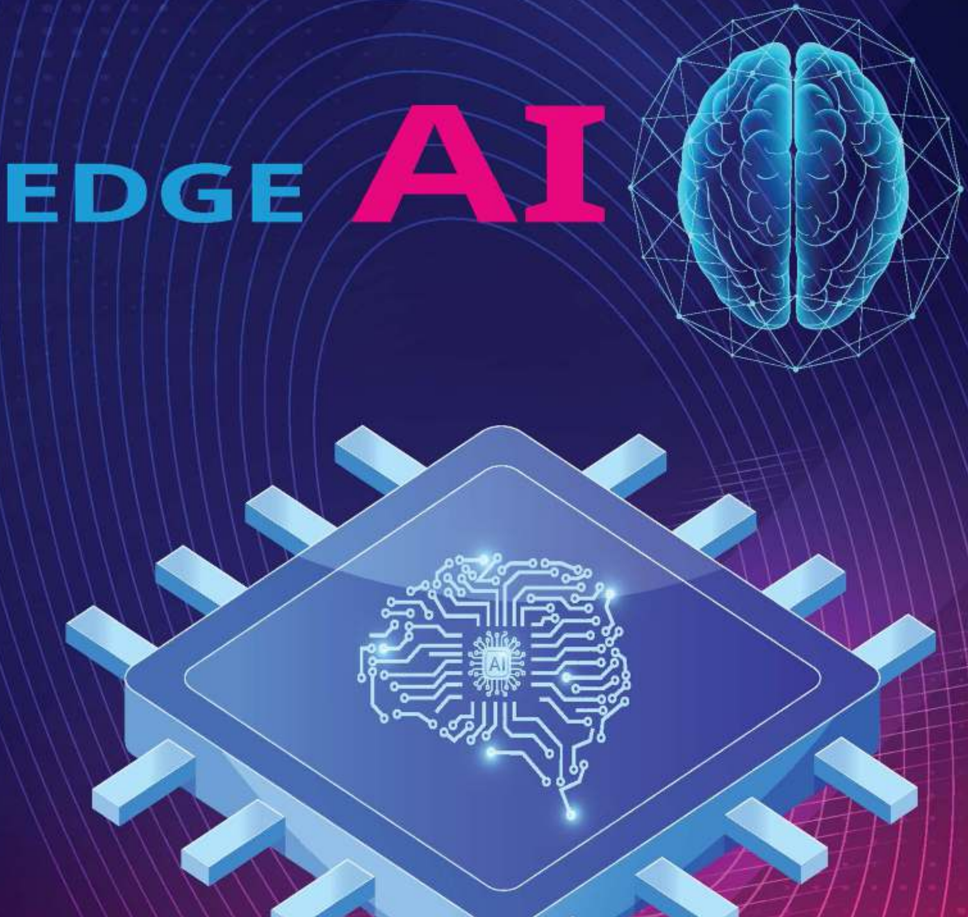
River Publishers Series in Communication and Networking

Charting the Intelligence Frontiers Edge AI Systems Nexus



Editors:
Ovidiu Vermesan
Alain Pagani
Paolo Meloni

River Publishers

Charting the Intelligence Frontiers
Edge AI Systems Nexus

Charting the Intelligence Frontiers Edge AI Systems Nexus

Editors

Ovidiu Vermesan
SINTEF, Norway

Alain Pagani
German Research Center for Artificial Intelligence, Germany

Paolo Meloni
University of Cagliari, Italy

River Publishers

Routledge
Taylor & Francis Group
NEW YORK AND LONDON

Published 2025 by River Publishers
River Publishers
Broagervej 10, 9260 Gistrup, Denmark
www.riverpublishers.com

Distributed exclusively by Routledge
605 Third Avenue, New York, NY 10017, USA
4 Park Square, Milton Park, Abingdon, Oxon OX14 4RN

Charting the Intelligence Frontiers Edge AI Systems Nexus / by Ovidiu Vermesan, Alain Pagani, Paolo Meloni.

ISBN: 978-87-4380-884-8 (hardback)
978-87-4380-887-9 (online)
978-87-4380-886-2 (master ebook)

DOI: 10.1201/9788743808862

Dedication

"Time stays long enough for those who use it."
– Leonardo da Vinci

"Pleasure in the job puts perfection in the work."
– Aristotle

"Success in the AI era will belong to those who adapt, learn, and innovate continuously."
– Anonymous

"It is not the strongest species that survive, nor the most intelligent, but the ones most responsive to change."
– Charles Darwin

Acknowledgement

The editors would like to thank all the contributors for their support in the planning and preparation of this book. The recommendations and opinions expressed in the book are those of the editors, authors, and contributors and do not necessarily represent those of any organizations, employers, or companies.

Ovidiu Vermesan
Alain Pagani
Paolo Meloni

Contents

Preface

A New Edge AI Reality

This book is the result of the rich exchanges of ideas and presentations at the European Conference on EDGE AI Technologies and Applications (EEAI) held on 21-23 October 2024 in Cagliari, Sardinia, Italy, offering a panoramic snapshot and a technical deep dive into the contemporary landscape of edge AI. With twenty selected chapters, it encapsulates the convergence of fundamental concepts, technical advancements, and real-world deployments that define the edge AI continuum.

Collectively, the book serves as a reference for the field, capturing the current state-of-the-art and anticipating future trends in hyperautomation, generative AI, connectivity, autonomy, and security mesh architectures. Whether you are seeking in-depth technical knowledge, inspiration for novel applications, or a strategic overview of the edge AI landscape, you will find invaluable insights from thought researchers and practitioners at the forefront of the field of edge AI.

A brief overview of each of the twenty chapters is provided below, highlighting the research and applications of edge AI that underscore the book's commitment to both technological and societal impact.

Edge AI Systems Verification and Validation: This chapter explores the challenges of verifying and validating complex edge AI systems, which integrate hardware, software, and data. It proposes a structured framework that combines model- and data-driven engineering to ensure these systems are reliable, robust, and meet regulatory standards.

Pioneering the Hybridization of Federated Learning: This work introduces a hybrid federated learning framework for human activity recognition, where some clients agree to share a portion of their data. The research assesses whether this partial data sharing can improve the overall classification accuracy of the collective model while maintaining user privacy.

Edge Intelligence Architecture for Distributed and Federated Learning: This chapter proposes a novel architecture for monitoring Electric Vehicles (EVs) by combining Federated Learning, Knowledge Distillation, and model compression. This approach enables the creation of efficient, privacy-preserving AI models that can be deployed on resource-constrained edge devices for applications like predictive maintenance.

Challenges and Performance of SLAM Algorithms on Resource-Constrained Devices: This study evaluates the performance of various visual-based SLAM (Simultaneous Localisation and Mapping) algorithms on resource-constrained hardware, such as the NVIDIA Jetson. It benchmarks several deep learning-based systems on metrics such as accuracy, energy consumption, and resource usage to assess their real-world viability.

Designing Accelerated Edge AI Systems with Model-Based Methodology: This chapter presents a Model-Based Cybertronic System Engineering (MBCSE) methodology for designing optimal edge AI systems with bespoke hardware accelerators. This approach enables a holistic analysis that balances performance, power, and cost, ensuring AI algorithms can be deployed effectively within tight system constraints.

Edge AI Acceleration for Critical Systems: Focusing on the demanding environment of satellites, this work discusses hardware solutions, such as FPGAs and CGRAs, for real-time, autonomous AI processing. The research addresses critical system challenges, including power constraints and radiation tolerance, and details the design of an FPGA-based GPU and an AI accelerator framework.

Model Selection and Prompting Strategies for LLM-Based Robotic Systems: This chapter examines the challenges of selecting and implementing Large Language Models (LLMs) in resource-constrained robotic systems. It highlights that changing model weights or precision often requires significant modifications to prompting strategies, complicating the development of modular, weight-agnostic systems.

Optimising ViT for Edge Deployment: This research presents a hybrid token reduction method, combining token merging and pruning, to make Vision Transformers (ViT) more efficient for semantic segmentation on edge devices. This approach significantly reduces computational complexity with only a minimal drop in accuracy, though it highlights challenges in exporting pruned models.

Recent Trends in Edge AI: This chapter provides a comprehensive overview of recent techniques for efficiently designing, training, and deploying machine learning models on edge devices. It covers scalable architectures, neural architecture search, and compression methods, such as quantisation and pruning, to enable energy-efficient AI in resource-limited environments.

Scalable Sensor Fusion for Motion Localization in Large RF Sensing Networks: This work addresses the challenge of accurate motion localisation in large-scale wireless sensing networks by using a probabilistic model. It demonstrates that variational Bayesian techniques offer a scalable solution for sensor fusion, enabling localised updates that model non-local effects efficiently.

Multi-Step Object Re-Identification on Edge Devices: This chapter proposes a pipeline for vehicle re-identification on edge devices using a multi-step feature extraction and matching process. The system detects an object, converts it to a vector embedding, and queries a database to find matches, achieving high precision in real-world camera network scenarios.

A TinyMLOps Framework for Real-World Applications: This work introduces a TinyMLOps framework to streamline the optimisation and deployment of AI models on microcontrollers. The framework uses cloud resources for intensive tasks while gathering real-time performance metrics from target devices, ensuring an accurate and scalable solution for deploying AI in constrained environments.

Transfer and Self-Learning in Probabilistic Models: This chapter explores the integration of transfer-learning and self-learning techniques within a single probabilistic model. The research finds that this synergy can be achieved through prior optimisation, enabling models to adapt across different environments where they are deployed.

A Novel Hierarchical Approach for On-Device Energy Efficient Fault Classification: This work proposes a hierarchical architecture utilising multiple smaller neural networks to perform energy-efficient fault classification directly on edge devices. By dividing the problem into smaller sub-tasks, the approach achieves a nine-fold reduction in energy consumption with comparable accuracy to a non-hierarchical model.

Discovering and Classifying Defects at the Edge: This chapter presents an AI-based optical inspection solution for detecting defects in digital and wooden industry products. Using YOLO and ResNet models deployed on edge devices, the system achieves high accuracy in identifying defect positions and classifying defect types, with explainability tools clarifying the model's decisions.

Conscious Agents Interaction Framework for Industrial Automation: This paper examines the integration of human cognitive models into industrial automation, aiming to create flexible, multi-agent systems where humans and machines collaborate as equal partners. Case studies in vertical farming and HVAC control demonstrate how agents can reason and negotiate to achieve both collective and individual goals.

Neuromorphic IoT Architecture for Efficient Water Management: This work proposes a neuromorphic IoT architecture inspired by biological systems to address the energy and communication challenges of traditional IoT networks. A case study on water management demonstrates how this event-driven, asynchronous approach can be realised with neuromorphic hardware to create a more efficient and responsive system.

Online AI Benchmarking on Remote Board Farms: This project aims to create a collaborative platform, dAIEdge - VLab, that enables researchers to benchmark AI models on a range of remote edge devices. This virtual laboratory will provide access to shared resources and tools, enabling users without deep-embedded expertise to conduct live AI experiments.

Optimising Neural Networks for Water Stress Prediction in Europe: This study compares various neural network architectures and optimisers to predict water stress, a key sustainability indicator accurately. The findings show that a three-layer architecture with an Adam optimiser provides the highest accuracy, offering a valuable tool for informed water resource management.

Decentralising Key Generation in CL-PKC with Traceable Ring Signatures: This chapter addresses a key vulnerability in Federated Learning by proposing a mechanism to decentralise key generation in Certificateless Public Key Cryptography. Using traceable ring signatures and blockchain infrastructure, the model provides accountability and disincentivises malicious behaviour among trusted authorities.

List of Figures

List of Tables

List of Contributors

Abidin, Aysajan, *COSIC, KU Leuven, Belgium*

Antonelli, Fabio, *Fondazione Bruno Kessler, Italy*

Antonini, Mattia, *Fondazione Bruno Kessler, Italy*

Antonio Coppola, Marcello, *STMicroelectronics, France*

Arents, Janis, *Institute of Electronics and Computer Science (EDI), Latvia*

Bahr, Roy, *SINTEF AS, Norway*

Bocchi, Tommaso, *University of Pisa, Italy*

Bublin, Mugdim, *University of Applied Science FH Campus Wien, Austria*

Bureka, Anzelika, *Institute of Electronics and Computer Science (EDI), Latvia*

Cancelliere, Francesco, *University of Catania, Italy*

Carnevale, Lorenzo, *University of Messina, Italy*

Ciabattini, Leonardo, *University of Bologna, Italy*

Dell'Acqua, Pierluigi, *University of Messina, Italy*

Deutel, Mark, *Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany*

Di Felice, Marco, *University of Bologna, Advanced Research Center for Electronic Systems, Italy*

Divernois, Margaux, *Haute Ecole Arc – HES-SO, Switzerland*

Dupertuis, Baptiste, *Haute Ecole Arc – HES-SO, Switzerland*

Eduards Zinars, Toms, *Institute of Electronics and Computer Science (EDI), Latvia*

Esposito, Alfonso, *University of Bologna, Italy*

Fanucci, Luca, *University of Pisa, Italy*

Faro, Robin, *Deepsensing SRL, Italy*

Frund, Robin, *Haute Ecole Arc – HES-SO, Switzerland*

Galagain, Calvin, *Université Paris-Saclay, CEA-List, ENSTA Paris, France*

Goulette, François, *ENSTA Paris, France*

Greitans, Modris, *Institute of Electronics and Computer Science (EDI), Latvia*

Grosu, Radu, *Technische Universität Wien, Austria*

Haroun, Karim, *Université Paris-Saclay, CEA-List, Université Côte d'Azur, France*

Hirner, Heimo, *University of Applied Science FH Campus Wien, Austria*

Huguenin, Maïck, *Haute Ecole Arc – HES-SO, Switzerland*

Jammal, Manal, *IoT Digital Innovation Hub, Spain*

Judvaitis, Janis, *Institute of Electronics and Computer Science (EDI), Latvia*

Klein, Russell, *Siemens EDA, USA*

Lanners, Antoine-Martin, *University of Applied Science FH Campus Wien, Austria*

Mallah, Maen, *Fraunhofer IIS, Fraunhofer Institute for Integrated Circuits, Germany*

Mishra, Varesh, *COSIC, KU Leuven, Belgium*

Moghbelan, Yasamin, *University of Bologna, Italy*

Monopoli, Matteo, *University of Pisa, Italy*

Montori, Federico, *University of Bologna, Advanced Research Center for Electronic Systems, Italy*

Nannipieri, Pietro, *University of Pisa, Italy*

Ovsiannikova, Polina, *Aalto University, Finland*

Pacini, Tommaso, *University of Pisa, Italy*

Pagani, Alain, *German Research Center for Artificial Intelligence (DFKI), Germany*

Parra-Domínguez, Javier, *IoT Digital Innovation Hub, Spain*

Pazos, Nuria, *Haute Ecole Arc – HES-SO, Switzerland*

Pijlman, Fetze, *Signify, Eindhoven University of Technology, The Netherlands*

Poreba, Martyna, *Université Paris-Saclay, CEA-List, France*

Preneel, Bart, *COSIC, KU Leuven, Belgium*

Proust, Mathilde, *Université Paris-Saclay, CEA-List, France*

Racinskis, Peteris, *Institute of Electronics and Computer Science (EDI), Latvia*

Sanz-Martín, Laura, *University of Salamanca, Spain*

Scheele, Stephan, *Ostbayerische Technische Hochschule Regensburg, Germany*

Solanti, Petri, *Siemens EDA, Germany*

Strano, Alessandro, *Deepsensing SRL, Italy*

Szczepanski, Michal, *Université Paris-Saclay, CEA-List, France*

Urlini, Giulio, *STMicroelectronics, Italy*

Vashishth, Devesh, *University of Applied Sciences, Heilbronn, Germany*

Vecchio, Massimo, *Fondazione Bruno Kessler, Italy*

Vermesan, Ovidiu, *SINTEF AS, Norway*

Villari, Massimo, *University of Messina, Italy*

Vismanis, Oskars, *Institute of Electronics and Computer Science (EDI), Latvia*

Vyatkin, Valeriy, *Aalto University, Finland; Luleå Tekniska Universitet, Sweden*

Wagner, Marco, *University of Applied Sciences, Heilbronn, Germany*

Wissing, Julio, *Fraunhofer IIS, Fraunhofer Institute for Integrated Circuits, Germany*

Zulberti, Luca, *University of Pisa, Italy*

Zutis, Tomass, *Institute of Electronics and Computer Science (EDI), Latvia*

Zyrianoff, Ivan, *University of Bologna, Italy*

List of Abbreviations

AHU	Air handling unit
AI	Artificial Intelligence
ANN	Artificial Neural Network
ASIC	Application-Specific Integrated Circuit
AXI	Advanced eXtensible Interface
BDI	Belief-desire-intention
CGRA	Coarse-Grained Reconfigurable Array
CNN	Convolutional Neural Network
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CPU	Central Processing Unit
CTS	Content-aware Token Sharing
DL	Deep Learning
DMA	Direct Memory Access
DNN	Deep Neural Network
DSE	Design Space Exploration
DSP	Digital Signal Processing
DToP	Dynamic Token Pruning
EI	Edge Intelligence
ESA	European Space Agency
EV	Electric Vehicle
FL	Federated Learning
FP	Floating-point
FPGA	Field Programmable Gate Array
FPS	Frames Per Second
FU	Functional Unit
FXP	Fixed-point
GB	Gigabyte
GEO	Geosynchronous Earth Orbit
GPGPU	General-Purpose Computing on Graphic Processing Units

GPU	Graphics Processing Unit
HCI	Human-computer interaction
HDL	Hardware Description Language
HMI	Human-machine interface
HVAC	Heating, ventilation, and air conditioning
IoT	Internet of Things
KD	Knowledge Distillation
LEO	Low Earth Orbit
LIB	LI-ion Battery
LUT	Look Up Table
MAC	Multiply And Accumulation
MAS	Multiagent systems
MCU	Microcontroller Unit
MDE	Modular Deep Learning Engine
MES	Manufacturing execution system
mIoU	Mean Intersection Over Union
ML	Machine Learning
MM	Memory-Mapped
mmseg	MMSegmentation, an open-source semantic segmentation toolbox
MPU	Microprocessor Unit
NN	Neural Network
NPU	Neural Processing Unit
OTA	Over-the-air
PA	Power/AreaRH Radiation-Hardened
RBf	Radial Basis Functions
RHBD	Radiation-Hardened by Design
RL	Reinforcement learning
RNN	Recurrent Neural Network
RT	Radiation-Tolerant
SAN	Stochastic Activity Network
SEL	Single Event Latchup
SEU	Single Event Upset
SLAM	Simultaneous Localization and Mapping
SoC	System-on-Chip
SoC	State of Charge
SoH	State of Health
SVD	Singular Value Decomposition
TCM	Tightly-Coupled Memory

TMR	Triple Modular Redundancy
VIO	Visual-Inertial Odometry
ViT	Vision Transformer
VO	Visual Odometry
VPU	Vision Processing Unit
VRAM	video random-access memory

1

Edge AI Systems Verification and Validation

**Ovidiu Vermesan[1], Alain Pagani[2], Roy Bahr[1],
Marcello Antonio Coppola[3], and Giulio Urlini[4]**

[1]SINTEF AS, Norway
[2]German Research Center for Artificial Intelligence (DFKI), Germany
[3]STMicroelectronics, France
[4]STMicroelectronics, Italy

Abstract

The integration of edge artificial intelligence (AI) into different complex systems presents unique challenges, particularly concerning their reliability, robustness, safety, and transparency. Edge AI systems must function as intended and meet regulatory and technical standards. Traditional verification and validation (V&V) methodologies, which are well-suited for conventional software (SW) and hardware (HW) systems, do not fully address the unique characteristics of edge AI-based systems that include hardware, software, elements of edge AI technology stack and data.

The chapter delves into the challenges and methodologies for edge AI verification and validation to identify the unique elements required to develop verifiable edge AI systems based on a structured verification and validation framework integrated with model- and data-driven engineering principles, assurance cases, and domain-specific requirements. It highlights the terminology and concepts for edge AI as a technology that integrates HW, SW, and edge AI technology and data while presenting the challenges of the convergence of these technologies in developing verification and validation solutions.

Keywords: edge AI, edge AI system, verification, validation, machine learning, deep learning, AI agents, agentic AI, system engineering, small language models.

DOI: 10.1201/9788743808862-1

1.1 Introduction and Background

Edge AI has become a cornerstone of innovation in various industries, driving advancements in automation, decision-making, and predictive analysis. Edge AI systems applying machine learning (ML), deep learning (DL), and data processing at the edge involving deep neural networks (DNN) present significant challenges for ensuring the reliability, safety, and effectiveness of intelligent embedded devices across the edge AI computing continuum, ranging from micro- to deep- and meta-edge. Edge AI can be either deterministic or non-deterministic, based on the typical application and design choices involved. Many edge AI applications prioritise real-time, deterministic behaviour for critical tasks, such as control algorithms. Other applications can leverage the non-deterministic nature of AI to deliver more adaptable and creative solutions as the non-deterministic nature of edge AI means it can offer different interpretations based on context. In real-time applications, edge AI systems require precise timing and consistent response times. This is demanded for tasks where milliseconds of delay can be critical. Deterministic edge AI is appropriate for applications that demand predictability and consistency, while non-deterministic approaches are advantageous for applications that require adaptability, creativity, and continuous learning. The choice between using a deterministic or non-deterministic approach finally depends on the detailed requirements of the application and the expected trade-offs among predictability, adaptability, and computational cost.

The advancement of edge AI technologies and the ubiquity of automated AI-based tools have created complex operational environments. Edge AI systems are evolving towards engineering advanced adaptive systems and require new concepts for verification and validation to address the challenging multidimensional integration of HW, SW, AI models, algorithms, datasets, and the multimodality of data.

The advantages of leveraging edge AI in many industrial applications include real-time processing, enhanced privacy and data security, reduced latency, optimised bandwidth, reliability, and scalability, as illustrated in Figure 1.1.

Edge AI technology stack combines AI and IoT with edge computing, allowing data processing and edge AI algorithm execution to occur directly on devices located at the edge of the network. By bringing AI closer to the source of data generation, edge AI enables more efficient and responsive decision-making across a wide range of applications.

AI systems, particularly those based on machine learning (ML), pose unique challenges that differ from traditional software. Unlike conventional

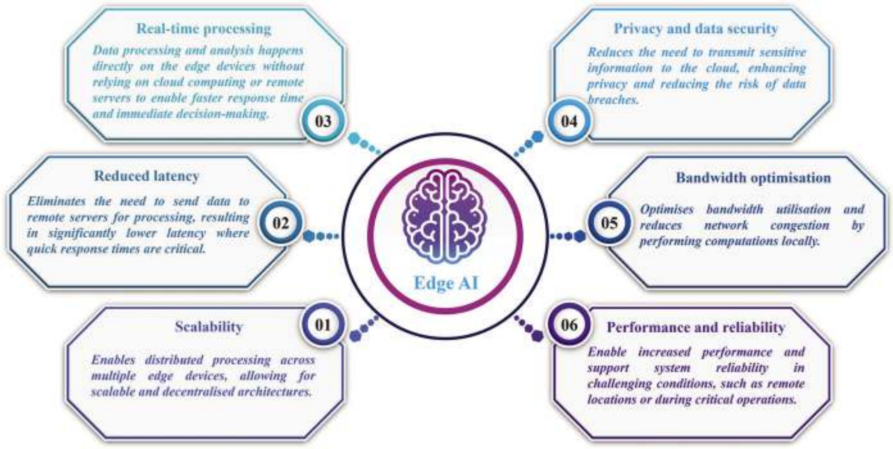

Real-time processing
Data processing and analysis happens directly on the edge devices without relying on cloud computing or remote servers to enable faster response time and immediate decision-making.
03
Privacy and data security
Reduces the need to transmit sensitive information to the cloud, enhancing privacy and reducing the risk of data breaches.
04
Reduced latency
Eliminates the need to send data to remote servers for processing, resulting in significantly lower latency where quick response times are critical.
02
Edge AI
Bandwidth optimisation
05
Optimises bandwidth utilisation and reduces network congestion by performing computations locally.
Scalability
01
Enables distributed processing across multiple edge devices, allowing for scalable and decentralised architectures.
06
Performance and reliability
Enable increased performance and support system reliability in challenging conditions, such as remote locations or during critical operations.

Figure 1.1 Edge AI advantages. ⏎

programs where behaviour is largely determined by explicit code, AI system behaviour often emerges from complex interactions between algorithms, vast datasets, and the operational environment [1].

Many advanced AI models, particularly deep neural networks, function as "black boxes," making their internal decision-making processes difficult to understand, predict, or inspect directly, hence difficult to validate [2]. This opacity, combined with potential determinism/non-determinism and sensitivity to data variations, complicates efforts to guarantee reliability, safety, and fairness [3].

A particularly demanding application domain of edge AI is real-time machine vision, which is critical in domains such as industrial robotics, autonomous navigation, and quality inspection. In these systems, the correctness and timeliness of visual perception directly influence physical actions, safety, and mission success. Their dependence on high-throughput, often noisy and non-reproducible visual data, and the need for ultra-low latency, makes their verification and validation particularly challenging under edge constraints.

The data-driven approach is based on systematically and algorithmically producing the best dataset to feed a given AI-based model, focusing on improving data quality and data governance to enhance the performance of a specific problem statement. Data-driven AI aims to improve data quality and outcomes by treating code as an unchangeable entity and dealing with labelling, augmenting, managing, and curating data. This is part of the data preprocessing, emphasising an iterative AI lifecycle consisting of data collection, model training, and error analysis.

The model-driven approach is based on producing the best model for a given dataset and aims to build new models and algorithmic improvements to enhance performance. The model-driven edge AI focuses on improving code reflecting the edge AI model or algorithm to achieve adequate results from fixed datasets. Edge AI developers view the training datasets from which the code, model, or algorithm is learning as a collection of reference labels. The edge AI model is made to fit that labelled training data and assumes the training data is external to the edge AI development process.

In model-driven edge AI, the focus is on optimising an edge AI model, whereas in data-driven edge AI, the focus is on data quality improvement. In model-driven edge AI, the aim is to find the most suitable edge AI model or an optimisation technique for a given problem, whereas, in data-driven edge AI, the aim is to find inconsistencies in the collected data for a given problem. The two approaches require specific verification and validation solutions.

Validation, in the context of edge AI systems, moves beyond the verification by checking if a system was built according to its technical specifications to seek confirmation that the edge AI system is fit for its intended purpose and effectively meet the actual needs and expectations of its users and stakeholders within its specific operational environment.

This necessitates the implementation of rigorous verification and validation processes, underscoring the responsibility and accountability in the development and implementation of edge AI systems.

The growing complexity and societal impact of AI, edge AI and generative AI demand a shift from purely technical verification towards a more holistic validation approach. This approach must encompass not only functional correctness but also usability, ethical alignment, fairness, robustness in real-world conditions, and overall effectiveness in achieving desired outcomes [6].

Before the adoption of AI agents and agentic AI with the use of large language models (LLMs), the development of autonomous and intelligent agents was deeply rooted in foundational paradigms of AI, such as multi-agent systems and expert systems, which emphasise social action and distributed intelligence [13][28].

Small language models (SLMs) are designed to offer capabilities similar to LLMs but scaled to edge computing capabilities, such as reduced size, processing requirements, and memory size. SLMs contain fewer parameters (e.g., hundreds of millions to one billion) while still providing strong performance for specific tasks.

Agentic AI is a class of systems that extends the capabilities of traditional AI agents by enabling multiple intelligent entities to collaborate on pursuing goals through shared memory [18][20], structured communication [24][22][26], and dynamic role assignment [21].

Ethical and legal aspects and the requirements on explainability and interpretability can lead to system development decisions that do not solely attempt to optimize functional requirements such as accuracy and robustness. In this case, system design choices rely on trade-offs that should ideally be made consciously by system developers.

Agentic AI systems pose challenges in explainability and verifiability due to their distributed, multi-agent architecture. While interpreting the behaviour of a single language model powered by the agent is already non-trivial, this complexity is multiplied when multiple agents interact asynchronously through loosely defined communication protocols. Each agent may possess its memory, task objective, and reasoning path, resulting in compounded opacity where tracing the causal chain of a final decision or failure becomes exceedingly difficult. The lack of shared, transparent logs or interpretable reasoning paths across agents makes it highly difficult, if not impossible, to determine why a particular sequence of actions occurred or which agent initiated a misstep. Compounding this opacity is the absence of formal verification tools tailored for agentic AI. In traditional software systems, model checking and formal proofs offer bounded guarantees, while there exists no widely adopted methodology to verify that a multi-agent system comprising multiple large language model agents collaborating on tasks will perform reliably across all input distributions or operational contexts.

Validation, therefore, serves as a cornerstone for building trustworthy edge AI, systems that stakeholders can confidently rely upon to operate safely, effectively, and responsibly [7]. It directly addresses the widening gap observed between accelerating edge AI capabilities and lagging safety protocols.

The AI verification standardisation efforts within the edge AI community underscores a fundamental challenge: establishing justified confidence, or trust, in edge AI systems whose behaviour often emerges unpredictably. This inherent uncertainty and the potential for significant negative impact necessitate rigorous V&V processes.

V&V encompasses activities designed to ensure that an edge AI system not only meets its specified requirements but also fulfils its intended purpose safely and reliably in its operational context. While drawing upon established V&V principles from software and systems engineering, edge AI verification and validation requires tailored approaches and methodologies to address its specific complexities.

The emphasis on “trustworthiness” in standards and frameworks like ISO/IEC TR 24028 directly reflects this imperative to build demonstrable confidence in AI systems [4][5].

This chapter provides a comprehensive overview of the verification and validation of edge AI systems. It examines definitions grounded in international standards, outlines the core elements subject to verification and validation, details the typical process steps involved, analyses the significant research challenges, explores contextual variations, discusses current research trends, and summarises future directions needed to advance the field.

1.2 Foundational Concepts and Edge AI Verification and Validation Taxonomy

In edge AI systems, the failure of an AI component can lead to overall system failure, highlighting the need for AI V&V. Components with AI capabilities are treated as subsystems. V&V is carried out both on the AI subsystem itself and on its interfaces with other parts of the overall system, just as with any other subsystem. That is, the high-level definitions of V&V remain unchanged for systems containing one or more AI components.

AI V&V challenges require approaches and solutions that go beyond those for conventional or traditional systems (those without AI elements). In the context of edge AI systems, AI components and subsystems need to be integrated into the systems engineering framework. This involves identifying the characteristics of AI subsystems that create challenges in their V&V, highlighting these challenges, and providing potential solutions while determining open areas of research in the V&V of edge AI subsystems.

Conventional SW/HW systems are engineered via three main phases, namely, requirements, design and V&V. These phases are applied to each subsystem and to the system under design.

Before the expansion of AI, ML, DL, and generative AI, research on V&V of neural networks addressed the adaptation of existing standards (e.g., IEEE Std 1012-Software Verification and Validation) and processed the augmentation of these standards to enable V&V and new techniques and lessons learned to solve the V&V issues for systems integrating AI components.

In all the adaptation and augmentation attempts, one of the challenges is data validation, as the data upon which AI depends should go through a form of V&V process. Data quality attributes that are important for edge AI systems include accuracy, currency and timeliness, correctness, consistency, usability, security and privacy, accessibility, accountability, scalability, lack of bias, and coverage and representativeness of the state space. Data validation steps can include file validation, transformation validation, import validation, domain validation, aggregation rule and business validation.

AI-based systems follow a distinct lifecycle compared to traditional systems. For edge AI systems learning lifecycle, V&V activities occur throughout the lifecycle, as illustrated in Figure 1.2. The requirements allocated to the edge AI subsystem encompass both hardware and software (HW/SW), as well as the AI models and data that flow up to the system from the edge AI subsystem.

Verification refers to the set of the activities that ensure that the edge AI system implements the specific function, and the system is built right according to requirements.

Edge AI system verification is the process of checking that the edge AI system achieves its goal without any bugs. It is the process to ensure whether the developed edge AI system is right or not. It verifies whether the developed product fulfils the requirements. Verification is static testing. Verification means answering the question: are we building the edge AI system, right?

Edge AI verification and validation require approaches and solutions at data, model and system level beyond those for cloud AI and conventional systems. Edge AI lifecycle workflows require to combine the SW/HW engineering methods with the data and system level analysis.

Data quality attributes like accuracy, timeliness, correctness, consistency, usability, security, privacy, accessibility, accountability, scalability, lack of

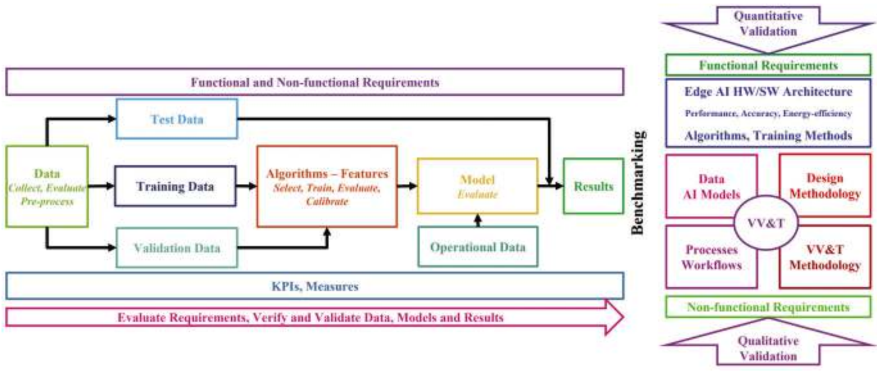


Figure 1.2 Edge AI verification and validation process. ⏎

bias, etc. are critical for edge AI. These data quality attributes are part of a larger edge AI non-functional requirements set.

Verification of edge AI systems involves systematically ensuring that AI models and their implementations fulfil specified requirements and intended purposes, as defined by recognised standards such as ISO/IEC 22989 (Information Technology - Artificial Intelligence - Concepts and Terminology). According to ISO, verification refers to the confirmation through objective evidence that specified requirements have been fulfilled. When applied to edge AI systems, verification processes ascertain that AI models and related software systems conform rigorously to technical and functional specifications, without necessarily validating the appropriateness of these specifications.

Principal elements involved in edge AI systems verification include:

- A formal requirements specification represents a crucial element, where clearly defined, unambiguous requirements serve as the foundational basis for verification. These specifications typically include functional requirements, performance criteria, safety constraints, security measures, and ethical guidelines.
- Model verification that entails evaluating AI models, including machine learning (ML) and deep neural networks (DNNs), ensuring their internal logic and behaviours align precisely with predefined specifications. Techniques employed in model verification include formal methods, theorem proving, model checking, and simulation-based testing.
- Software and hardware integration verification, which is vital, ensuring edge AI systems correctly interact with hardware components and software environments. It includes examining interface correctness, interoperability, real-time performance, and robustness under varying conditions and inputs.
- Rigorous test case generation and execution constitute essential verification steps. AI system verification employs automated test generation methods, including boundary value analysis, equivalence partitioning, and mutation testing, complemented by scenario-based testing to thoroughly assess compliance and performance under diverse and extreme operational conditions.
- Documentation and traceability processes involve detailed records demonstrating systematic compliance with verification steps, adherence to standards, and requirement fulfilment. Comprehensive documentation supports transparency and accountability and facilitates continuous improvement and iterative refinement processes.

In this context, the principal verification process involves several methodical steps:

- Requirement Analysis: Clearly define and document edge AI systems' functional, performance, and safety requirements.
- Verification Planning: Establishing a structured plan that details verification strategies, methods, criteria, and resources.
- Model and Code Inspection: Applying manual or automated inspections and formal verification techniques to analyse AI model structures and implementation code for correctness.
- Test Development: Generating extensive and varied test cases covering all possible usage scenarios, operational environments, and stress conditions.
- Verification Execution: Systematically conducting tests and verification activities, rigorously analysing outcomes against specified acceptance criteria.
- Reporting and Review: Documenting detailed verification outcomes, identifying discrepancies, and facilitating stakeholder review to ensure comprehensive verification coverage.
- Iterative Refinement: Addressing identified issues through iterative model adjustments, re-verification cycles, and continual improvement to achieve specified verification goals.

Validation refers to the set of the activities that ensure that the edge AI system that has been built is traceable to the requirements and the right edge AI system is built to meet user needs.

Validation is the process of checking whether the edge AI system is up to the mark or, in other words, if the product has high-level requirements. It is the process of checking the validation of the edge AI system, e.g., it checks if what we are developing is the right edge AI system. It is validation of the actual and expected edge AI systems. Validation is a form of dynamic testing. Validation means answering the question: are we building the right edge AI system?

Validation of edge AI systems is a critical and systematic process intended to ensure that the developed AI system meets stakeholders' and end-users' specific needs and expectations, as explicitly outlined in ISO/IEC 22989 (Information Technology — Artificial Intelligence — Concepts and Terminology). According to ISO standards, validation involves confirming through objective evidence that the requirements for a specific intended use or application have been fulfilled. In AI, validation goes beyond verifying compliance with technical specifications—it assesses whether the system performs suitably in real-world conditions and scenarios.

The principal elements involved in the validation of edge AI systems encompass several dimensions:

- The identification of intended use and user requirements is foundational. Clear articulation and comprehensive understanding of user needs, operational contexts, and usage environments are paramount. This involves gathering input from stakeholders and end-users to form a robust basis for subsequent validation activities.
- Operational scenario definition is critical. Edge AI systems must be validated within scenarios that accurately represent real-world operational contexts. Scenarios are typically derived from realistic usage conditions, including normal operational states, boundary conditions, and potential abnormal or edge cases.
- Performance evaluation under realistic conditions is essential. Validation combines simulated environments and real-world testing to ensure edge AI systems perform reliably and effectively. Performance metrics, such as accuracy, precision, recall, robustness, resilience, and usability, form the basis for evaluating system performance and alignment with stakeholder expectations.
- Human-machine interaction and usability assessment are integral to validation. Edge AI systems are validated to ensure effective and intuitive interactions with human operators or users. Usability testing, user experience assessments, and feedback loops with real users facilitate comprehensive evaluations of the AI system's ease of use and accessibility.
- Safety, security, and ethical considerations are central elements of the validation process. These assessments verify that edge AI systems function correctly and comply with safety standards, security protocols, data privacy laws, and ethical guidelines, aligning with international frameworks and societal expectations.

The edge AI validation process typically involves structured, methodical steps:

- Requirement and Expectation Definition: Establishing clear validation criteria and user expectations, documenting them rigorously.
- Validation Planning: Creating detailed validation plans that specify methodologies, scenarios, test environments, and acceptance criteria.

- Scenario Development: Defining realistic operational scenarios and selecting representative use-cases and edge-cases for comprehensive validation.
- Simulation and Real-world Testing: Controlled simulations are conducted, followed by real-world trials to evaluate AI system performance against established criteria.
- Performance and Usability Assessment: Analysing performance outcomes, usability data, and user feedback to ascertain compliance with expectations and user requirements.
- Safety, Security, and Ethical Evaluation: Systematically reviewing compliance with safety and security standards, data protection requirements, and ethical norms.
- Reporting and Continuous Improvement: Compiling comprehensive validation reports, documenting findings and recommendations, and establishing iterative cycles for continuous system refinement.

Verification and validation of AI and edge AI models and data are required in safety-critical applications to ensure the trustworthiness of edge AI-enabled systems (e.g., reliability, availability, maintainability, safety, security, resilience, connectability, explainability, interpretability, transparency, etc.) as illustrated in Figure 1.3 and Figure 1.4.

Dependable edge AI systems involve using systems and software engineering principles to systematically guarantee dependability during the edge AI system's construction, V&V, and operation and consider legal and normative requirements directly from the start.

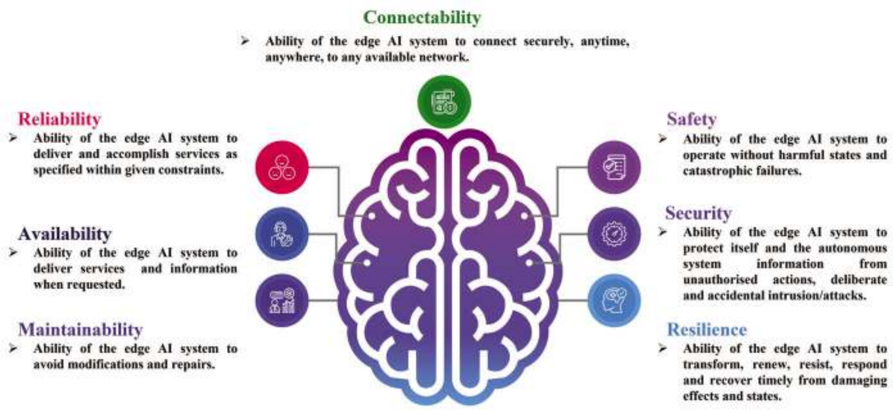

Connectability
➢ Ability of the edge AI system to connect securely, anytime, anywhere, to any available network.
Reliability
➢ Ability of the edge AI system to deliver and accomplish services as specified within given constraints.
Availability
➢ Ability of the edge AI system to deliver services and information when requested.
Maintainability
➢ Ability of the edge AI system to avoid modifications and repairs.
Safety
➢ Ability of the edge AI system to operate without harmful states and catastrophic failures.
Security
➢ Ability of the edge AI system to protect itself and the autonomous system information from unauthorised actions, deliberate and accidental intrusion/attacks.
Resilience
➢ Ability of the edge AI system to transform, renew, resist, respond and recover timely from damaging effects and states.

Figure 1.3 Edge AI dependability – Trustworthiness. ⏎

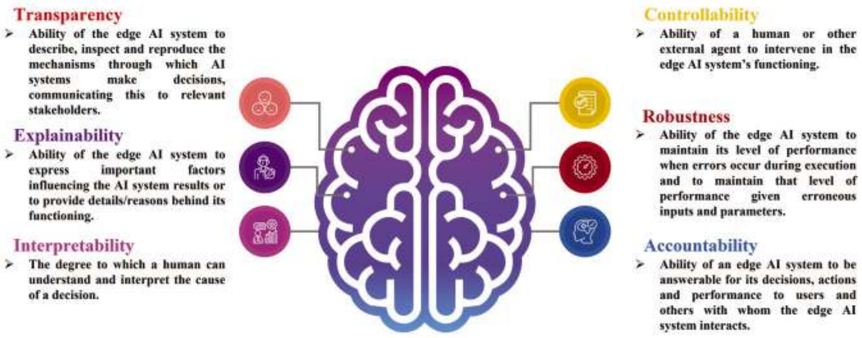

Transparency
➢ Ability of the edge AI system to describe, inspect and reproduce the mechanisms through which AI systems make decisions, communicating this to relevant stakeholders.
Explainability
➢ Ability of the edge AI system to express important factors influencing the AI system results or to provide details/reasons behind its functioning.
Interpretability
➢ The degree to which a human can understand and interpret the cause of a decision.
Controllability
➢ Ability of a human or other external agent to intervene in the edge AI system's functioning.
Robustness
➢ Ability of the edge AI system to maintain its level of performance when errors occur during execution and to maintain that level of performance given erroneous inputs and parameters.
Accountability
➢ Ability of an edge AI system to be answerable for its decisions, actions and performance to users and others with whom the edge AI system interacts.

Figure 1.4 Edge AI dependability - Trustworthiness extended properties. ⏎

The progress made in developing standards and regulatory frameworks for AI and edge AI aims to ensure the responsible use of AI in various applications.

The relevant standards for AI that can be applied to edge AI systems are ISO/IEC 42001 and ISO/IEC TR 24028:2020 that are described below.

The ISO/IEC 42001 standard, a management system for AI, focuses on building trust and dependability in AI systems. It provides a framework to establish, implement, maintain, and continually improve their AI management systems, ensuring the responsible development and use of AI. The standard emphasises trustworthiness, fairness, transparency, and accountability in AI systems [43][44].

The ISO/IEC TR 24028:2020 standard addresses topics related to trustworthiness in AI systems, including approaches to establish trust in AI systems through transparency, explainability, controllability, etc.; engineering pitfalls and typical associated threats and risks to AI systems, along with possible mitigation techniques and methods; and approaches to assess and achieve availability, resiliency, reliability, accuracy, safety, security and privacy of AI systems [5].

Traditional V&V workflows, such as the V-model, are insufficient for ensuring the accuracy and reliability of AI and edge models. As a result, transformations of these workflows occurred to better serve edge AI applications.

1.2.1 Agentic AI and AI Agents

The evolution of generative AI and the emergence of AI agents and agentic AI requires addressing them under the presentation of foundational concepts and edge AI verification and validation taxonomy by defining the concepts and their specific characteristics.

AI Agents can be defined as autonomous software entities engineered for goal-directed task execution within bounded digital environments. These agents are characterised by their ability to perceive structured or unstructured inputs, to reason over contextual information, and to initiate actions toward achieving specific objectives. The main characteristics of AI and edge AI agents are autonomy, task specificity, reactivity and adaptability, which enable the agents to operate as modular, lightweight interfaces between pre-trained AI models and domain-specific pipelines and workflows.

AI agents are the concrete instantiations of the agentic AI paradigm. An AI agent is a specific software or hardware entity that embodies the principles of agentic AI. It is a tangible system equipped with sensors to perceive its environment and effectors to act upon it. While agentic AI is the "what," the AI agent is the "how", the actual implementation that performs tasks, makes decisions, and interacts with external environments.

Agentic AI systems describe a paradigm shift from isolated AI agents to collaborative, multi-agent ecosystems capable of decomposing and executing complex goals [21]. These systems typically consist of orchestrated or communicating agents that interact via tools, APIs, and shared environments [23][14].

A key distinction between agentic AI and AI agents lies in their level of abstraction, as the agentic AI is a conceptual framework, whereas an AI agent is a functional system. An analogy can be drawn between the theory of computation and a physical computer. One provides the theoretical foundation and a model of what is possible, while the other is the practical machine that executes computations based on that theory.

Agentic AI reflects a broad paradigm in AI and edge AI centred on creating systems that can perceive their environment, reason about their observations, and act autonomously to achieve specific goals. It is the underlying philosophy and set of principles that guide the development of intelligent, goal-oriented systems. This concept emphasises proactivity, reactivity, and social ability, defining the potential for AI to operate as an independent actor rather than a passive tool. Agentic AI systems introduce internal orchestration mechanisms and multi-agent collaboration frameworks. Agentic AI extends the foundational architecture to support complex, distributed, and adaptive behaviours by integrating components such as specialised agents, persistent memory, orchestration and advanced reasoning and planning. Agentic AI introduces novel memory integration, communication paradigms, and decentralised control, paving the way for the next generation of adaptive workflow automation in autonomous systems, swarm robotics, and autonomous vehicles with scalable, adaptive intelligence.

In robotics and automation, agentic AI enables collaborative behaviour in multi-robot systems. Each robot operates as a task-specialised agent, such as a picker, transporter, or mapper, while an orchestrator supervises and adapts workflows. These architectures rely on shared spatial memory, real-time sensor fusion, and inter-agent synchronisation for coordinated physical actions. Use cases include warehouse automation, drone-based orchard inspection, and robotic harvesting [25].

Verification and validation of edge AI systems, based on AI agents and agentic AI components, must focus on ensuring the correctness, reliability, and robustness of autonomous decision-making in highly dynamic and constrained environments, which requires validating that the AI agents consistently perform their intended functions correctly under varying external environment conditions, including unexpected scenarios and adversarial inputs.

Due to the limited computational resources typical of edge devices, V&V must also confirm that the system meets stringent real-time performance requirements, ensuring timely responses to critical events despite hardware and network limitations.

Another aspect is assessing the resilience and safety of adaptive learning processes within these systems, particularly as they evolve in open environments. V&V efforts should capture how individual agents and collective multi-agent behaviours emerge and interact, verifying alignment with overall system objectives and preventing unsafe or unintended actions.

Additionally, transparency and trustworthiness are key elements that enable human oversight, offering clear traceability and checkability of decisions made by autonomous components at the edge.

1.3 Defining Verification and Validation per Standard

Several ISO standards offer consistent definitions for verification and validation, primarily within the context of quality management and systems/software engineering that can be applicable to AI and edge AI as presented below.

ISO 9000:2015 (Quality management systems - Fundamentals and vocabulary) provides the definition for verification as the “confirmation, through the provision of objective evidence, that specified requirements have been fulfilled" [29]. The focus is on confirming that the system or component conforms to its design specifications and requirements [31]. It answers the question: “Did we build the product right?” [32]. Verification is often viewed as an internal process comparing the outputs of a development phase against the inputs [31]. Validation is defined as the “confirmation, through the provision of objective evidence, that the requirements for a specific intended use or application have been fulfilled” [29]. The focus shifts to ensuring the system meets the needs of the user and fulfils its intended purpose in the actual context of use. It answers the question: “Did we build the right product?” [32]. Validation often involves testing under real or simulated use conditions and considers stakeholder needs [31].

ISO 9001:2015 (Quality management systems - Requirements), states that validation activities ensure the resulting products/services meet requirements for the specified application or intended use. Validation often involves acceptance testing with end-users and assessing fitness for purpose, making it frequently an external process, whereas verification is more often internal. Both verification and validation are essential components of quality management and are necessary for ensuring a dependable system [30].

ISO/IEC/IEEE 15288:2015 (Systems and software engineering - System life cycle processes) standard integrates V&V into the system lifecycle and considers that the **verification process** has as purpose “to provide objective evidence that a system or system element fulfils its specified requirements and characteristics” [34]. It involves activities comparing the system or element against requirements, design descriptions, and other required characteristics, confirming it was “built right” [35], while the **validation process** has as purpose “to provide objective evidence that the system, when in use, fulfils its business or mission objectives and stakeholder requirements, achieving its intended use in its intended operational environment” [33]. This process confirms that stakeholder requirements are correctly defined, and that the system meets its intended purpose in the context where it will operate [33].

ISO/IEC 22989:2022 (Information technology - Artificial intelligence - Artificial intelligence concepts and terminology) AI-specific standard defines **verification** as “confirmation, through the provision of objective evidence, that specified requirements have been fulfilled,” noting it assures conformance to specification [9]. While not explicitly defining validation in the same way, it defines **trustworthiness** as the “ability to meet stakeholder expectations in a verifiable way" [38]. This definition links the core goal of validation (meeting stakeholder expectations/needs) directly to the concept of trustworthiness in AI. The standard also incorporates a "verification and validation" phase within its depiction of the AI system lifecycle [39].

ISO/IEC TR 24028:2020 (Information technology - Artificial intelligence - Overview of trustworthiness in Artificial Intelligence) technical report further reinforces the link between validation and trustworthiness and defines **trustworthiness** as the "ability to meet stakeholder expectations in a verifiable way" [40]. This aligns the concept of trustworthiness directly with the objective of validation – confirming that stakeholder needs and intended use requirements are met [42]. The report discusses assessing and achieving key characteristics like reliability, safety, security, and privacy, all crucial aspects evaluated during validation [41].

ISO/IEC 42001:2023 (AI Management System) standard specifies requirements for establishing, implementing, maintaining, and continually improving an AI Management System (AIMS) within an organization [43]. An AIMS provides a structured framework for responsible AI governance, risk management, and operational control throughout the AI lifecycle [44]. Verification activities are integral to an AIMS, supporting risk assessment, impact assessment, performance evaluation, and ensuring compliance with policies and objectives [44]. Notably, ISO/IEC 22989 (providing the core AI terminology) is a normative reference for ISO/IEC 42001, highlighting the foundational role of clear definitions [45].

IEEE 1012-2016 (IEEE Standard for System, Software, and Hardware Verification and Validation) standard applies to systems, software, and hardware being developed, maintained, or reused (legacy, commercial off-the-shelf [COTS], non-developmental items) [91]. The term "software" also includes firmware and microcode. Additionally, each of the terms "system," "software," and "hardware" encompasses documentation. V&V processes include the analysis, evaluation, review, inspection, assessment, and testing of products. V&V processes are used to determine whether the development products of a given activity conform to the requirements of that activity and whether the product satisfies its intended use and user needs. V&V lifecycle process requirements are specified for different integrity levels. The scope of V&V processes encompasses systems, software, hardware, and their interfaces.

1.4 Key Elements for Edge AI Verification and Validation

The elements for verifying and validating edge AI may encompass operational aspects, system integration, AI models and human-machine interaction. Verification and validation are important to ensure reliability, performance and accuracy of complex systems.

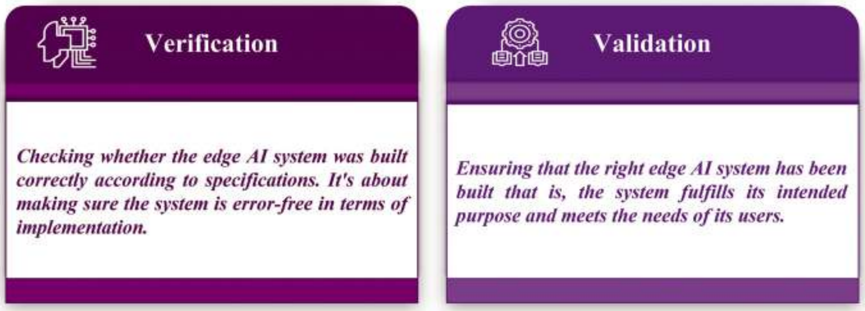

Verification
Checking whether the edge AI system was built correctly according to specifications. It's about making sure the system is error-free in terms of implementation.
Validation
Ensuring that the right edge AI system has been built that is, the system fulfills its intended purpose and meets the needs of its users.

Figure 1.5 Verification and validation.

Edge AI system verification and validation refers to the processes and methodologies used to ensure that an edge AI system is dependable, performs as expected, and meets certain standards before it is deployed.

These processes are crucial as the edge AI algorithms can work with high-stakes decision-making, various sizes datasets, learn and evolve over time. The processes are needed for ensuring that the edge AI systems do what they are supposed to do, without unintended consequences, biases, or errors.

AI and edge AI systems typically focus on the actual algorithms and models to ensure that they perform as intended under various conditions. In addition, edge AI systems focus on validating the systems performance on resource-constrained devices, network conditions and privacy in real-world scenarios.

It is critical to distinguish verification from validation. While verification checks conformance to specifications ("Did we build the system right?"), validation confirms that the system meets the needs of the customer and other stakeholders and fulfils its intended purpose in its operational environment ("Did we build the right system?") [30].

The introduction of AI in product and systems development has significantly increased the complexity of electronic components and systems (ECS), by integrating various technologies such as hardware, software, ML, DL, NNs, generative AI, and advanced data analytics. This complexity necessitates robust verification and validation frameworks and benchmarking to ensure these systems operate correctly and efficiently as illustrated in Figure 1.6. Complex edge AI models require verification and validation to ensure their predictions, decisions, and content generation outputs are reliable and accurate, which is critical for maintaining the trustworthiness of AI systems. Failures in edge AI-based ECS can have significant economic and business-critical consequences, including system failures, financial loss, and damage to infrastructure, making the dependability of edge AI systems paramount.

In machine vision, specific verification concerns arise from the need to ensure reliable object detection, tracking, segmentation, or pose estimation across a wide range of dynamic conditions. For example, verification must confirm that visual inference results remain stable under varying lighting, occlusion, and motion blur, common challenges in edge deployments like factory floors or drones.

Ensuring robustness and reproducibility in edge-based machine vision systems is inherently difficult due to the high variability and noise in visual data. Unlike structured tabular inputs, images and videos exhibit a vast range of intra-class variation—objects or actions belonging to the same class can appear drastically different depending on factors such as:

- Lighting conditions (e.g., shadows, reflections).
- Occlusions or partial views.
- Background clutter.
- Camera distortions, blur, or motion artifacts.
- Variability in object shape, colour, texture, or viewpoint.

A comprehensive V&V framework, presented in Figure 1.6, along with benchmarking of edge AI-based methods, frameworks, tools, and ECS, is essential to ensure performance and dependable system properties like security, reliability, robustness, and fairness.

Verification ensures that edge AI-based methods, frameworks, tools, and electronic components and systems are built correctly and meet specifications, while validation confirms they perform as intended in real-world scenarios.

In edge AI systems there is a need of creating a structured approach to defining and applying such a framework to edge AI-based tools and methods, ensuring ECS meet functional and non-functional requirements, quality, KPIs, and performance standards.

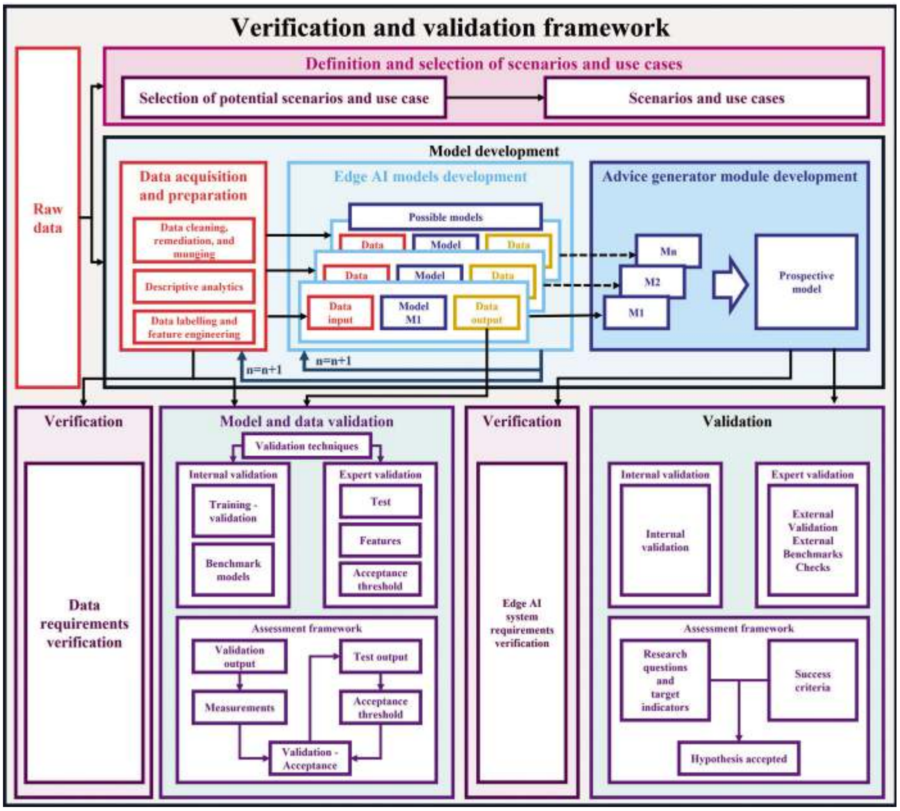


Figure 1.6 Verification and validation framework. ⏎

1.4.1 Core Elements for AI Verification

Verification activities in AI systems must address multiple facets, spanning data, models, system-level behaviour, and the processes governing development and deployment. Ensuring the integrity and appropriateness of each element is crucial for overall system trustworthiness.

1.4.1.1 Data Verification

Given that many AI and edge AI systems, particularly those based on ML, learn from data, verifying the data itself is paramount [3]. Key aspects include:

Data Quality: Assessing if the data meets predefined standards for accuracy, completeness, consistency, timeliness, and representativeness for the target domain [3]. This involves checking for errors, missing values, correct formatting, and ensuring the data is current and relevant [46]. Poor data quality directly impacts model performance and reliability. Machine vision applications often require extensive data augmentation and synthetic dataset generation for robustness. In this case, the validity of the augmented dataset needs to be verified for plausibility and compliance with conditions of the actual use of the system.

Data Bias: Identifying systemic skews or prejudices within the data that could lead to unfair or discriminatory outcomes [3]. Verification involves confirming that bias detection methods have been applied and that any mitigation steps align with fairness requirements or definitions. This includes checking for underrepresentation or imbalances across demographic groups [3].

Data Provenance and Lineage: Ensuring the origin and history of the data are understood and documented, including all transformations and processing steps [47]. Verification confirms traceability back to authorized sources and validates the integrity of the data pipeline.

Data Security and Privacy: Confirming that data collection, storage, and processing adhere to relevant privacy regulations like General Data Protection Regulation (GDPR), a law in the European Union aimed at safeguarding the data and privacy of EU residents or California Consumer Privacy Act (CCPA) a US state law that applies to for-profit businesses operating in California that collect personal information from California residents, and organizational security policies [3]. This includes verifying the implementation of techniques like anonymization, encryption, access controls, and proper consent management [48].

Data Labelling: For supervised learning, verifying the accuracy, consistency, and quality of labels applied to the training and testing data is crucial, as errors here directly impact model learning [3].

1.4.1.2 Model Verification

The AI and edge AI model itself, the core component that performs learning and prediction, requires rigorous verification:

Accuracy and Performance: Quantifying how well the model achieves its intended task according to predefined metrics (e.g., precision, recall, F1-score for classification; BLEU score for translation) evaluated on unseen test or validation datasets [41]. Verification confirms that the achieved performance meets the specified requirements or benchmarks.

Robustness: Evaluating the model's ability to maintain its performance level when faced with noisy data, adversarial perturbations, changes in data distribution (drift), or other unexpected conditions [49]. Verification checks if the model's resilience meets specified criteria under defined stress conditions.In machine vision models, robustness testing should also include tests for perceptual artifacts, such as camera motion blur or lens distortion, and adversarial perturbations that affect visual features. This is especially important for safety-critical applications like automated visual inspection or autonomous guidance.

Reliability: Assessing the consistency and predictability of the model's outputs under normal operating conditions over time [50]. Verification aims to confirm that the model behaves dependably within its specified operational domain.

Efficiency: Measuring the model's consumption of computational resources, such as processing time, memory usage, and energy [48]. Verification ensures the model operates within the constraints imposed by the deployment hardware or system requirements.

1.4.1.3 System-Level Verification

Verification must also extend to the AI and edge AI system, considering its interaction with its environment and users:

Safety: Confirming that the system operates without causing unacceptable levels of risk or harm to humans, property, or the environment [49]. This involves verifying adherence to specific safety requirements, standards (like ISO 26262 for automotive), and risk assessments. In vision-driven systems, safety verification must ensure that the interpretation of the visual scene cannot trigger unsafe behaviour due to false positives or misclassifications e.g., mis detecting a pedestrian or failing to recognise a hazard in the camera feed.

Security and Resilience: Checking the implementation and effectiveness of measures designed to protect the system against threats like unauthorized access, data breaches, model tampering, and adversarial attacks [3]. It also includes verifying the system's ability to withstand and recover from disruptions [51].

Fairness: Evaluating system outcomes across different demographic or user groups to ensure equity and the absence of harmful bias or discrimination, according to defined fairness metrics or criteria [3].

Privacy: Verifying that the system's operation, including data handling and output generation, complies with privacy principles and regulations throughout its use [52].

1.4.1.4 Process and Governance Verification

Beyond the technical components, the processes surrounding the AI system also require verification of:

Transparency: Assessing whether sufficient and appropriate information about the AI system (its purpose, data sources, model type, limitations, performance) is documented and made available to relevant stakeholders (developers, deployers, users, regulators) [53]. Verification checks if documentation and communication channels meet specified transparency requirements.

Explainability and Interpretability: Evaluating whether the system can provide understandable reasons or justifications for its outputs or decisions, tailored to the applications and users. Verification checks if the explanation mechanisms provided meet requirements for clarity, fidelity, and utility.

Accountability: Confirming that clear roles, responsibilities, governance structures, and mechanisms for oversight, audit, and redress are defined, documented, and effectively implemented [6]. Verification involves auditing these governance processes and structures against standards like ISO/IEC 42001.

These verification elements are deeply interconnected [3]. For instance, verifying fairness requires access to appropriate data and potentially explainability techniques to understand model behaviour. Verifying safety may depend on demonstrating model robustness and having transparent documentation of system limitations. An opaque model hinders the verification of its internal logic, making it difficult to assess its safety or fairness properties directly. This interdependence necessitates a holistic verification strategy rather than treating each element in isolation.

Furthermore, the emphasis placed on different verification elements naturally shifts depending on the type of AI system. For data-driven ML models, verification heavily scrutinizes data quality, bias, model performance, and robustness [3].

In contrast, for symbolic AI systems built on explicit rules and logic, verification may concentrate more on the consistency, correctness, and completeness of the knowledge base and the soundness of the reasoning engine [64].

Hybrid neuro-symbolic systems demand verification of both the neural and symbolic parts, as well as their complex interactions, representing a distinct verification challenge [64].

1.4.2 Core Elements Subject to AI Validation

Given validation's focus on fitness for purpose and meeting stakeholder needs, the elements assessed extend beyond traditional software checks. Validating AI systems requires evaluating a broader spectrum of characteristics that reflect their performance, usability, effectiveness, and impact within their socio-technical context [6]. The exact scope may vary based on the application domain, but the core elements subject to validation include:

1.4.2.1 Ensuring Fitness for Intended Purpose and Operational Context

This is a central element of validation. It involves confirming that the AI system effectively achieves its stated goals within the specific environment and conditions of its intended use [54]. This requires a clear definition of the intended purpose and the Operational Design Domain (ODD), the specific conditions under which the system is designed to function. However, defining and validating against these can be particularly challenging for adaptive AI systems or those designed for open-world environments where conditions are dynamic and unpredictable [3]. Validation must assess performance not just under nominal conditions but also under stress, edge cases, and potential environmental shifts or adversarial inputs [85]. Frameworks like the NIST AI Risk Management Framework (RMF) emphasize establishing context (Map function) as a foundational activity to inform subsequent measurement and management, including validation [55].

1.4.2.2 Meeting User Needs and Stakeholder Expectations

Validation explicitly confirms that the system satisfies the requirements and expectations of its end-users and other relevant stakeholders [30]. This extends beyond purely functional requirements to encompass aspects like usability, user satisfaction, ease of integration into existing workflows, and alignment with business objectives [56]. Because AI systems can impact a wide range of individuals and groups, validation should involve engagement with diverse stakeholders, including end-users, domain experts, potentially affected communities, and regulators, to capture a comprehensive set of needs and expectations [90]. Addressing the challenge that these needs might be implicit, diverse, or even conflicting is a key part of the validation process [57].

1.4.2.3 Assessing Real-World Effectiveness and Outcomes

Validation must measure how the AI and edge AI system performs in practice, assessing its actual effectiveness in achieving desired outcomes within realistic scenarios [59]. This moves beyond performance metrics derived solely from laboratory settings or curated test datasets. It involves evaluating the system's impact on relevant Key Performance Indicators (KPIs), operational efficiency, safety records, cost savings, or other context-specific measures of success [58]. Initiatives like NIST's Assessing Risks and Impacts of AI (ARIA) program are specifically focused on developing methodologies to measure these real-world impacts under controlled conditions [61]. This assessment typically requires methods such as Operational Testing (OT), field testing, pilot deployments, and continuous performance monitoring after deployment [6]. In edge machine vision systems, this includes validating that visual perception models continue to perform accurately when deployed with quantized weights, compressed inputs, or on hardware that introduces latency jitter. This real-world validation should account for degradation due to environmental variables and resource limitations.

1.4.2.4 Evaluating Usability and Human-AI Interaction

For edge AI and AI systems that interact with or support humans, validation must assess the quality and effectiveness of this interaction [60]. This includes evaluating usability (ease of use, learnability, efficiency), the clarity and utility of the interface, the cognitive load imposed on the user, and overall user satisfaction [63]. Particularly for human-AI collaboration or teaming scenarios, validation needs to assess the effectiveness of the partnership, the safety of the interaction, the appropriateness of trust levels (avoiding over-trust or under-trust), and the degree of shared understanding between human and AI [61]. This requires human-centered evaluation methods, such as usability studies, task analyses involving representative users, and systematic collection of user feedback [62].

1.4.2.5 Validating Ethical Alignment and Societal Impact

A critical dimension of edge AI validation involves assessing the system's alignment with ethical principles and societal values [6]. This includes validating characteristics like fairness, accountability, and transparency in practice [55]. Methodologies such as Ethical Impact Assessments (EIAs) are emerging to help proactively identify, assess, and mitigate potential negative ethical and societal consequences before and during deployment [61]. A key focus is validating fairness and non-discrimination, moving beyond simple dataset metrics to assess the actual impact on different demographic groups in real-world deployment contexts [90]. This also involves considering broader societal implications related to employment, environmental sustainability, and the functioning of democratic processes [7].

1.4.2.6 Data Quality and Suitability

High-quality data ensures that models are trained effectively and can make accurate predictions in real-world scenarios [36]. As AI and edge AI systems become more complex and are deployed in diverse environments, the challenges associated with data quality and suitability have become increasingly significant. Considering the specific requirements for various AI and edge AI systems, challenges for data quality and suitability in AI and edge AI validation include:

Relevance and Representativeness: the data used for training and validation is relevant and representative of the real-world environment in which the AI and edge AI systems operate. Data must reflect the diversity of conditions, contexts, and populations that the system will encounter. If the training data is biased or unrepresentative, the model's performance may deteriorate when applied to actual situations.

Volume and Availability: Considered very important, especially in scenarios where data may be generated at high velocity. Obtaining enough high-quality data for training and validation can be difficult. In many cases, developers may struggle to gather sufficient diverse data from edge devices, leading to models that are not well-trained for all possible situations they may encounter in deployment.

Label Quality: important for supervised learning, as it directly impacts model accuracy. Inaccurate or inconsistent labelling can mislead the training process and result in poor performance in operational environments. Ensuring the reliability of labels, especially when data is labelled manually or derived from semi-automated processes, can be a significant extra work.

Bias and Fairness: the biases in learning and training of data, can lead to outcomes that are unfair when models are deployed. AI and edge AI systems trained on biased data may perpetuate existing stereotypes or discriminate against certain classes and groups. Addressing data bias and ensuring fairness in model predictions is key to building trustworthy AI and edge systems that serve all stakeholders equitably.

Data Drift: refers to the shifts in data distributions over time, which can degrade model performance. As the underlying data evolves, models may become less accurate or irrelevant. Ongoing monitoring and adaptation of models are necessary to mitigate the effects of data drift, making it a continuous challenge for AI and edge AI validation.

Data Preprocessing: is a critical step in data management, particularly for edge AI systems with limited resources. Cleaning and transforming data into suitable formats can be challenging, when using with diverse data sources and formats. This preprocessing must be efficient to ensure real-time performance while maintaining data accuracy and integrity.

Synthetic Data: helps augment training datasets and has several limitations. The effectiveness of synthetic data depends on its ability to mimic real-world scenarios accurately. If synthetic data does not accurately represent the complexities of real-world environments, it may lead to models that underperform when applied to actual data.

Edge-Specific Challenges: these are related to data collection from distributed edge devices, considering elements like latency, bandwidth constraints, and intermittent connectivity, which can complicate the data validation process. Ensuring data quality in these scenarios requires innovative approaches to data management and model training.

1.5 The Edge AI Verification and Validation Lifecycle

According to the OECD recommendation on artificial intelligence [10], an AI system is a machine-based framework that, driven by either explicit or implicit goals, deduces from the input it receives how to produce outputs such as predictions, content, recommendations, or decisions that may impact physical or virtual environments. The levels of autonomy and adaptability of different AI systems can vary after they are deployed. The lifecycle of an AI system generally encompasses multiple stages, which include planning and design; data collection and processing; model development and/or adaptation of existing models for specific tasks; testing, evaluation, verification, and validation; deployment for use; operation and monitoring; and retirement or decommissioning. These stages often occur iteratively and are not strictly linear. The choice to retire an AI system can be made at any time during the operation and monitoring stage.

AI and edge AI systems are distinct from other system types, which can influence the processes of the lifecycle model, such as [9]:

- Most SW systems are designed to operate in exactly defined manners dictated by their requirements and specifications. In contrast, AI and edge AI systems that utilize ML rely on data-driven training and optimization techniques to address a wide range of inputs.
- Traditional SW applications tend to be predictable, whereas this is less frequently true for AI and edge AI systems.
- Additionally, traditional SW applications are generally verifiable, while evaluating the performance of AI and edge AI systems often necessitates statistical methods, making their verification more complex.
- AI and edge AI systems usually require numerous iterations of enhancement to reach satisfactory performance levels.

The edge AI development lifecycle outlines the stages involved in creating and operationalizing edge AI systems. It starts with problem definition, functional, non-functional requirements and data collection, followed by data preparation and feature engineering. Model selection and architecture design precede the training phase, where algorithms learn from the prepared dataset. Validation and testing ensure model performance and generalization. Iterative refinement optimizes the model based on results. Deployment integrates the AI system into production environments. Monitoring and maintenance track performance, address drift, and update the model as needed.

Embedding AI and generative AI into the system design requires shifting from the current V-model, which addresses the HW and SW development cycle, to a W-model superimposed on the V-model to account for data and AI-specific artifacts. This includes the AI model development and data into the AI system's lifecycle development, as illustrated in Figure 1.7 [92].

When superimposed on the traditional V-model used in HW and SW development, the W-model AI development lifecycle creates a comprehensive framework that addresses the distinct yet interrelated processes of AI data development and HW/SW engineering. This approach ensures that AI models and supporting systems are developed in a cohesive, iterative, and validated manner. This approach aligns existing tools and methods with AI technologies. The extension into the W-model structures represents the development workflow of AI systems comprising HW, SW, AI stack, and data components.

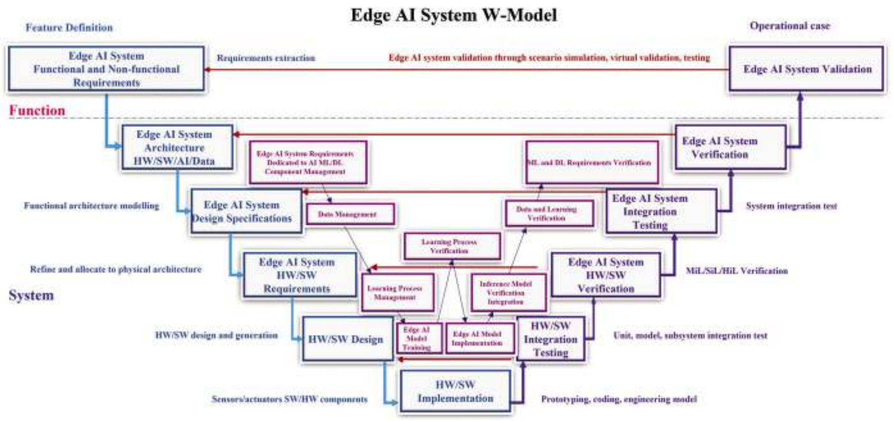

Edge AI System W-Model
Feature Definition
Operational case
Edge AI System Functional and Non-functional Requirements
Requirements extraction
Edge AI system validation through scenario simulation, virtual validation, testing
Edge AI System Validation
Function
Edge AI System Architecture HW/SW/AI/Data
Edge AI System Requirements Dedicated to AI ML/DL Component Management
ML and DL Requirements Verification
Edge AI System Verification
Functional architecture modelling
Edge AI System Design Specifications
Data Management
Data and Learning Verification
Edge AI System Integration Testing
System integration test
Learning Process Verification
Refine and allocate to physical architecture
Edge AI System HW/SW Requirements
Learning Process Management
Inference Model Verification Integration
Edge AI System HW/SW Verification
MiL/SiL/HiL Verification
System
HW/SW design and generation
HW/SW Design
Edge AI Model Training
Edge AI Model Implementation
HW/SW Integration Testing
Unit, model, subsystem integration test
Sensors/actuators SW/HW components
HW/SW Implementation
Prototyping, coding, engineering model

Figure 1.7 Edge AI system W-Model (adapted from [92]) ⏎

The inner W part of the model represents the AI-enabled processes and workflows integrated into the conventional model. The AI system W-model emphasises systematic validation and verification at each stage of the AI development process, helping ensure the robustness, reliability, and performance of AI systems. The W-model addresses the specific design and development requirements of edge AI systems, distinguishing them from traditional HW/SW and computing paradigms. The novelty in the model lies in the fact that the data required for development and AI, ML/DL, and generative AI model training is integrated into the development cycle, superimposed on the traditional V-model, and follows the algorithm selection and training in each lifecycle stage.

The edge AI system W-model emphasises that AI and generative AI are integral to the lifecycle development processes of any AI-based product or service.

As presented in the AI system W-model at the start of the development lifecycle, developers can utilise AI and generative AI to understand domain requirements and design architecture. The design captures both functional and non-functional requirements for embedded computing systems, such as those in automotive control or industrial units, considering hardware constraints and real-time performance needs. Challenges concerning edge AI requirements and AI requirements engineering are extensive and due in part to the practice by some to treat the AI element as a “black box”. Formal specification has been attempted and has proven to be difficult for tasks that are hard to formalise, requiring decisions on the use of quantitative or Boolean specifications, as well as the incorporation of data and formal requirements. The challenge is to design effective methods for specifying both desired and undesired properties of systems that utilise AI- or ML-based components.

When considering the broader principles of agentic AI and AI agents, their development must be integrated into the edge AI system W-model as a specific part of the lifecycle development processes of any AI-based product or service. As a result, the agentic AI and AI agents V&V extends beyond the individual agent, focusing on validating the system's autonomy and the ability to achieve long-term goals without unexpected consequences by assessing the alignment of the AI agent's goals with the overall objectives of the edge AI system and ensuring that its learning and adaptation mechanisms do not lead to unsafe or undesirable states over time.

1.6 Failure Case Behaviour in Edge-based Machine Vision Systems

In the context of edge-based machine vision systems, the study of failure case behaviour is a critical component of any robust verification and validation framework. These systems are increasingly deployed in real-world, safety-critical environments, ranging from autonomous vehicles and industrial robotics to surveillance and medical diagnostics, where failures can result in substantial consequences. While conventional validation focuses on average-case performance metrics such as accuracy or mean average precision, these metrics often obscure rare but consequential failure modes. An edge AI system that performs well under ideal conditions may fail unexpectedly in the presence of visual distortions, environmental variability, or edge hardware constraints.

Failures in machine vision models frequently arise in conditions that deviate from the data distribution seen during training. Examples include poor lighting, motion blur, occlusions, scale variation, or visual clutter. In edge deployments, such conditions are not only likely but expected, and the consequences of misclassification or missed detection can be severe. Furthermore, edge systems often operate with limited fallback options, and they must respond in real time, leaving little margin for error recovery. Understanding and characterizing these failure scenarios is therefore essential for both safety assurance and iterative model improvement.

One important strategy for investigating failure modes involves deliberate stress testing through visual perturbations. By applying controlled transformations—such as adding noise, blurring, shifting brightness, or introducing occlusions—it becomes possible to evaluate how resilient a vision model is to real-world distortions. These tests often reveal brittle model behaviours that are not apparent during standard validation.

In addition, targeted scenario-based testing using simulation tools or recorded video sequences enables systematic exploration of edge cases. This is especially valuable for applications involving dynamic environments, such as autonomous navigation, where rare events (e.g., unexpected pedestrian appearance or sensor occlusion) may not be captured adequately in available datasets. Scenario replay or simulation also supports reproducibility of observed failures, which is often a challenge in field deployments.

Another aspect of failure case analysis is the examination of uncertainty and confidence levels in model predictions. Machine vision systems may produce incorrect predictions with unjustified confidence, especially when faced with unfamiliar or out-of-distribution inputs. Monitoring softmax confidence, prediction entropy, or Bayesian uncertainty estimates can help identify instances where the model is likely to fail. These signals may be used in runtime monitoring or to trigger fail-safe mechanisms.

Post-hoc explainability methods, such as saliency maps or activation heatmaps, also play an important role in understanding failure behaviour. By visualising the regions of an input image that contributed most to a model's prediction, one can diagnose whether a failure was due to the model focusing on irrelevant or misleading features. This insight often reveals underlying dataset biases or spurious correlations that were inadvertently learned. By combining scenario condition variables and the predictions as features in probabilistic frameworks (e.g., Bayesian networks) is also a method for modelling the uncertainty in predictions.

Hardware-specific issues also need to be considered in failure analysis. For instance, the quantization of weights and activations required for execution on edge hardware (e.g., FPGAs or ASICs) can introduce numerical inaccuracies that degrade model performance in subtle ways. Testing the consistency between floating-point reference models and their hardware-deployed counterparts is essential to identify precision-induced errors. Similarly, real-time system profiling can reveal frame drops, synchronization mismatches, or input-output latency violations that lead to perceptual failures.

Finally, insights gained from the analysis of failure cases should feed back into the design and development process. Difficult or misclassified examples can be incorporated into retraining pipelines, synthetic data can be generated to increase robustness, and system architectures can be adapted to detect and respond to high-uncertainty inputs. Ideally, safety envelopes are defined at design time to formally capture the operational conditions under which the system is guaranteed to function correctly. This creates a closed-loop process that not only identifies but also mitigates and prevents known failure patterns.

The systematic study of failure case behaviour is indispensable for building trustworthy machine vision systems on edge platforms. It enables developers to move beyond average-case performance toward comprehensive assurance of correctness, robustness, and safety under realistic and adverse conditions.

1.7 Research Challenges in Edge AI Verification and Validation

Edge AI represents a cutting-edge computing approach that seeks to relocate the training and inference of ML models to the network's edge [12]. However, implementing intelligence at the edge presents several significant challenges, such as the necessity to limit model architecture designs, ensuring the secure distribution and execution of the trained models, and managing the considerable network load needed to disseminate the models and the data gathered for training.

Edge AI systems that incorporate continuous learning involve the gradual updating of the models within the systems during production and test runs operations [9]. The data input into a system during these operations, is not only evaluated to generate an output but is also concurrently utilized to modify the model, aiming to enhance it based on the production data. Depending on the design of the continuous learning system, certain human interventions may be necessary, such as data labelling, validating the application of specific incremental updates, or monitoring the performance of the edge AI system. Continuous learning can address the limitations of the initial training data and assist in managing data drift and concept drift, but it also presents significant challenges in ensuring the edge AI system operates correctly while learning. It is essential to verify the system in production and to capture the production data to be able to include them as part of the training dataset in future system updates.

Catastrophic interference (and catastrophic unlearning) occurs when the training for new tasks disrupts the model's comprehension of previous tasks [11]. As new information supersedes earlier learning, the model forfeits its capability to manage its initial tasks. Given the risk of catastrophic interference, continuous learning necessitates the capability to learn over time by integrating new observations from current data while preserving prior knowledge [9]. Numerous ML algorithms excel at learning tasks only when the data is provided in a single batch. As a model is trained on a specific task, its parameters are modified to effectively tackle that task. However, when new training data is introduced, the adjustments made for these new inputs can erase the knowledge the model had previously gained. In the context of neural networks, this occurrence is regarded as one of their key limitations.

The combination of edge AI, IoT and Cyber-Physical Systems (CPS) marks a significant transformation in data processing by bringing it closer to the origin. This strategy minimizes latency, improves real-time decision-making, and lessens the load on centralized cloud resources. In CPS, control logic is utilized to process input from sensors, through actions of actuators and thus affecting processes occurring in the physical world [9]. This is particularly evident in robotics, where sensor data is directly employed to manage the robot's operations and execute tasks in the physical world. Typically, robots are equipped with sensors at the edge to evaluate their current conditions, processors to facilitate control through analysis and action planning, and actuators to implement those actions.

In contrast to industrial robots, which are consistently repeating the same trajectories and actions without deviations, service robots or collaborative robots must adapt to evolving situations and dynamic environments [9]. Programming this adaptability presents significant challenges due to the inherent variability. Components of edge AI systems can play a role in the control software and planning processes through the "Sense-Plan-Act" framework, allowing robots to modify their actions in response to obstacles or changes in the location of target objects. The integration of robotics and edge AI system components facilitates automated physical interactions with objects, environments, and individuals.

In machine vision-based robotics, the visual processing pipeline itself must be verified and validated not only for accuracy but also for real-time responsiveness. Edge V&V must ensure that latency from image acquisition to action initiation does not exceed application-specific safety thresholds. Techniques like real-time trace logging and FPGA-based image path profiling can support this validation.

The challenges and appropriate methodologies for AI verification are not uniform; they vary significantly depending on the type of AI model employed and the application domain's risk profile.

Model-Specific Challenges and Verification Focus

Deep Learning (DL) / Sub-Symbolic AI:

- *Challenges:* The primary verification challenges stem from their inherent opacity (making internal logic inscrutable) [64], strong data dependency (performance tied to training data quality and representativeness) [76], difficulty in formal specification of complex learned behaviours [64], susceptibility to adversarial examples, and challenges in generalization beyond training data distributions [76]. Scalability of verification methods is a major bottleneck due to the vast number of parameters and high-dimensional inputs [52]. Non-determinism can also arise during training or inference [70].
- *Verification Focus:* Emphasis is placed on empirical performance evaluation using diverse test datasets, extensive robustness testing against perturbations and adversarial attacks, fairness audits to detect biases learned from data, applying explainable AI (XAI) techniques (like LIME, SHAP, saliency maps) and interpretable AI (IAI) to gain insights into model decisions [74][75] and, where feasible, formal verification of specific, localised properties such as robustness bounds around specific inputs [69].

Symbolic AI / Rule-Based Systems:

- *Challenges:* These systems often suffer from **brittleness**, meaning they struggle to handle situations not explicitly covered by their predefined rules or knowledge base [73]. Creating and maintaining large, consistent, and complete knowledge bases can be labour-intensive and requires significant domain expertise [72]. They typically lack the ability to learn directly from raw, unstructured data. A computer vision model trained to detect stop signs may misclassify a slightly occluded or weathered sign because it hasn't seen enough variation in training. Traditional software can also exhibit brittleness, i.e. they both struggle but in different forms.
- *Verification Focus:* Verification centres on the logical integrity of the system. This includes checking the consistency of the rule set and knowledge base (absence of contradictions), analysing completeness (do the rules cover the intended domain?), formally verifying logical properties like soundness and validity of reasoning steps [64] and ensuring the traceability of outputs back to specific rules, which provides inherent explainability [72].

Neuro-symbolic AI:

- *Challenges:* This hybrid approach aims to combine the strengths of DL and symbolic AI but verifying the interaction and ensuring consistency between the neural (learning) and symbolic (reasoning) components is a key challenge [64]. Developing unified V&V frameworks that can handle both paradigms simultaneously is an active area of research [64].
- *Verification Focus:* Requires a multi-pronged approach: verifying the neural components using DL-specific techniques, verifying the symbolic components using logic-based methods, and crucially, verifying the interface and the correctness of the combined system's behaviour. A major research direction involves leveraging the symbolic part to constrain, explain, or formally verify aspects of the neural part's behaviour [64].

Domain-Specific Challenges and Verification Focus

Safety-Critical Systems (e.g., Automotive, Aerospace, Medical, Industrial Control):

- *Requirements:* These domains demand high levels of reliability, safety, robustness, and predictability [49]. System failures can have catastrophic consequences, including loss of life, severe injury, or significant environmental damage [49].
- *Challenges:* The need for provable guarantees clashes with the opacity and non-determinism of many AI components [65]. Meeting stringent regulatory standards (e.g., ISO 26262, IEC 62304, DO-178C) requires extensive evidence and documentation, which is difficult for AI/ML [68]. Managing the complexity of interaction with the physical world and ensuring safety across a vast range of operational scenarios is extremely challenging [71]. Exhaustive testing is typically infeasible due to the combinatorial explosion of possibilities [47]. Achieving deterministic replay for debugging and analysis is crucial but difficult [78].
- *Verification Focus:* Emphasis on rigorous methodologies, including formal methods where applicable, extensive simulation-based testing covering edge cases and failure modes, hardware-in-the-loop and real-world testing, fault tolerance analysis, adherence to domain-specific safety standards, meticulous documentation, and end-to-end requirements traceability [65]. Building a robust safety case with sufficient evidence is paramount [78].

Business or mission critical:

- *Requirements:* Business or mission-critical edge AI systems refer to applications that utilise AI to enable real-time decision-making and enhance operational efficiency. These domains demand high levels of scalability, reliability, safety, robustness, and interoperability. Due to their critical nature, they require special attention to ensure performance.
- *Challenges:* Deployment challenges arise from network reliability, as edge devices may operate in environments with unstable connections, affecting data synchronisation and model updates. The diversity of hardware platforms can cause compatibility issues and necessitate tailored solutions. Software challenges include the need for model optimisation, as AI models must be adjusted for edge deployment to balance accuracy and resource utilisation. Environmental conditions also pose risks, as edge devices must withstand various factors that can influence hardware performance and reliability. Regulatory and compliance challenges require navigating global data protection regulations to ensure that AI systems adhere to legal standards.
- *Verification Focus:* Verifying the effectiveness of edge AI systems involves establishing rigorous processes to ensure AI models meet performance standards under diverse conditions. Performance evaluation includes conducting real-time benchmarks to assess the responsiveness, accuracy, and resource utilisation of AI models deployed on edge devices. Interoperability ensures that edge AI solutions operate and communicate effectively within existing ecosystems and various hardware. Compliance verification requires regular audits to ensure that edge AI systems adhere to data privacy laws and industry regulations. Robustness verification involves stress-testing models against adversarial attacks and unexpected inputs to confirm their resilience in real-world scenarios. Lifecycle management strategies are necessary for overseeing the entire lifecycle of edge AI systems, from development and deployment to decommissioning.

Consumer Applications (e.g., E-commerce, social media, entertainment):

- *Requirements:* Often prioritize performance (e.g., accuracy of recommendations, speed of response), user experience, scalability, and cost-effectiveness. While direct physical safety risks are typically lower, significant concerns exist around fairness, bias, privacy, security (e.g., data breaches), misinformation, and ethical use [66].

- *Challenges:* Managing bias and fairness effectively across large, diverse user populations [66]. Protecting user privacy in data-hungry applications. Detecting and mitigating the generation or spread of harmful content or misinformation [67]. Preventing user manipulation (e.g., prompt injection in chatbots) [67]. Understanding and mitigating potential large-scale societal impacts [79][80].
- *Verification Focus:* Often relies more heavily on empirical testing, such as testing for performance, user studies for usability and acceptance, large-scale fairness and bias audits, privacy impact assessments and compliance checks, evaluation of content safety filters, and robustness testing against common failure modes or attacks. While formal verification might be used for specific critical components (e.g., payment processing), the overall verification rigor may be less intense than in safety-critical domains, unless specific high-risk functions are involved.

The fundamental difference in verification approaches between these contexts stems from the level of acceptable risk and the potential severity of failure consequences. Safety-critical domains operate with extremely low risk tolerance, demanding the highest levels of assurance and necessitating the use of more rigorous, often formal, verification techniques, alongside adherence to strict regulatory frameworks [65]. Consumer applications, while facing significant ethical and societal risks, typically have a higher tolerance for certain types of failures (e.g., a poor recommendation vs. a medical misdiagnosis), allowing for a greater reliance on empirical testing and monitoring. Addressing these domain-specific challenges in business and mission-critical edge AI systems is key for ensuring their reliability and effectiveness.

The inherent difficulties in verifying both pure DL (opacity) and pure symbolic AI (brittleness) have spurred interest in hybrid neuro-symbolic approaches [64]. By integrating the pattern-recognition strengths of neural networks with the explicit reasoning and transparency of symbolic methods, these approaches offer a potential pathway to building edge AI systems that are more amenable to verification and trust, particularly for complex tasks [77]. The verification of hybrid systems introduces its own set of research questions regarding the interaction and consistency between the different components [64].

For validation a one-size-fits-all approach to edge AI is ineffective due to the diversity of edge AI technologies and their application domains. The specific validation focus, methods, metrics, and acceptance criteria must be tailored to the type of AI system and the context in which it operates, particularly considering the nature of user interaction and potential real-world consequences.

Validation Nuances Across AI Types

Generative AI (e.g., LLMs, SLMs, VLMs, image generators): Validation priorities include assessing factual accuracy (mitigating “hallucinations”), ensuring content safety (detecting toxicity, bias, harmful content), preventing malicious use (e.g., generating disinformation or deepfakes), and evaluating output quality attributes like coherence, relevance, and creativity, which often lack objective metrics [66]. The inherent non-determinism is a key challenge, requiring validation strategies that assess output distributions or use human evaluation and red-teaming [81]. Defining and validating the “intended purpose” for highly flexible generative models is complex [87].

Agentic AI and AI agents: The AI agent act as a deterministic component with limited scope, while agentic AI reflects distributed intelligence, characterised by goal decomposition, inter-agent communication, and contextual adaptation, demonstrating key characteristics of the modern agentic AI frameworks. Agentic AI systems define an emergent class of intelligent architectures in which multiple specialised agents collaborate to achieve complex, high-level objectives utilising collaborative reasoning and multi-step planning [17]. V&V of edge AI systems employing AI agents focuses on the reliability and safety of the agent’s actions in its operational environment to ensure the agent’s decision-making logic is robust and predictable under a broad range of inputs, especially unexpected or anomalous sensor data. This involves rigorous testing of the agent’s software, hardware, edge AI algorithms and data components to confirm they meet design specifications and performance benchmarks. Another aspect of V&V for edge AI systems that must be considered is the formal verification of the agent’s reasoning processes, which involves creating mathematical models of the agent and its environment to demonstrate that specific critical properties, such as safety, robustness, and resilience, are met. For complex, learning-based agents, this can be supplemented with extensive simulation-based testing to explore the vast state space and identify potential failure modes before deployment in the domain applications.

Autonomous Systems (e.g., autonomous vehicles, industrial robots): Validation overwhelmingly focuses on safety, reliability, and robustness within complex and dynamic physical environments [7]. Key challenges include achieving sufficient test coverage across a vast space of potential scenarios (combinatorial explosion), bridging the gap between simulation and real-world performance, validating perception systems, and ensuring safe decision-making under uncertainty [78]. Validation heavily relies on extensive simulation, structured scenario-based testing, field testing, formal methods for safety-critical properties, and potentially runtime verification/monitoring [78]. Validating human oversight mechanisms is also critical, especially in military or safety-critical contexts [71].

Decision Support Systems (e.g., medical diagnosis aids, credit scoring tools): Validation emphasizes accuracy, reliability, fairness, explainability, and the system's impact on human decision-making and outcomes [7]. Challenges include validating against potentially imperfect or subjective ground truth, ensuring edge AI recommendations are beneficial and not misleading, rigorously assessing and mitigating bias across different user groups, and providing sufficient transparency to enable user trust and accountability. Validation typically requires domain-specific performance metrics, evaluation by domain experts, user studies assessing impact on decisions, and thorough bias and fairness audits [85].

Domain-Specific Considerations

The application domain significantly shapes validation priorities and methods due to differing risk profiles, regulatory requirements, and stakeholder concerns:

Healthcare: Extremely high stakes due to direct impact on patient safety and well-being [82]. Validation must adhere to regulatory frameworks (e.g., FDA regulations for medical devices, HIPAA for privacy, EU AI Act classifying medical AI as high-risk). Key validation elements include clinical efficacy (proven through clinical evaluation/trials), safety, reliability, data privacy, mitigation of bias in diverse patient populations, usability for clinicians, and explainability to support clinical judgment and trust. Frameworks like FUTURE-AI offer specific guidance for trustworthy AI in healthcare [86].

Finance: Focus on regulatory compliance (e.g., financial conduct authorities, anti-discrimination laws), fairness and bias mitigation in areas like credit scoring and loan applications, accuracy in fraud detection, model risk management, robustness against market volatility, security against financial attacks, and explainability for audits and customer inquiries [83]. Validation involves rigorous back testing, stress testing under various market conditions, comprehensive bias audits using relevant fairness metrics, security penetration testing, and checks for regulatory adherence.

Transportation (especially Autonomous Vehicles): Safety is the absolute priority [82]. Validation must demonstrate safe operation under a vast range of environmental conditions (weather, lighting, road types) and interactions (other vehicles, pedestrians, cyclists). This involves validating perception systems (sensor fusion, object detection/classification), prediction models, and planning/control algorithms [71]. Validation relies heavily on extensive simulation covering millions of virtual miles, structured scenario-based testing (including edge cases and failure modes), real-world road testing, and the development of robust safety cases supported by evidence [78]. Formal verification methods may be applied to critical safety properties [71].

Social media / Content Platforms: Key concerns involve mitigating the spread of misinformation and harmful content, addressing algorithmic bias in content ranking and recommendation, ensuring fairness in content moderation, protecting user privacy, and managing the impact on user well-being and societal discourse [84]. Validation is challenging due to the massive scale, the dynamic nature of content and user behaviour, the subjectivity involved in defining "harmful" or "fair," and the difficulty in measuring long-term societal impacts. Validation methods often include large-scale testing, human content review and rating, analysis of user engagement and feedback data, and monitoring metrics related to bias, toxicity, and content diversity.

This context-dependency highlights that effective AI validation requires not only technical expertise, but also deep domain knowledge [86]. Generic validation checklists are insufficient; protocols must be tailored to the specific AI type, its intended application, the operational environment, the relevant risks, and the specific needs and values of the stakeholders in that domain [88].

1.8 Trends and Methodologies in Edge AI Verification and Validation

The field of edge AI verification is rapidly evolving, driven by the increasing capabilities and deployment of AI systems, alongside growing concerns about their trustworthiness and potential risks. The unique challenges of edge AI validation are driving significant research and development into new methodologies, techniques, and tools. These efforts aim to provide more rigorous, scalable, and comprehensive ways to ensure edge AI systems are fit for their intended purpose.

Formal methods are advancing with a growing interest in applying, mathematically rigorous techniques, to the verification and validation of edge AI systems. Techniques like model checking, theorem proving, abstract interpretation, and reachability analysis are being adapted to prove specific properties of edge AI components, especially neural networks, concerning safety, robustness against perturbations (e.g., adversarial examples), and fairness. Major challenges remain in scaling these methods to handle the high dimensionality and complexity of edge AI models and in formally specifying properties for systems operating under uncertainty or with incomplete requirements. Research focuses on developing more scalable algorithms, better abstraction techniques, and methods for probabilistic verification.

Explainable and interpretable AI for V&V are techniques that are increasingly explored as tools for validation and verification. By providing insights into why a model makes a certain prediction (e.g., identifying important input features using SHAP or LIME, visualizing attention mechanisms, generating counterfactual explanations), AI explainability and interpretability can help validators assess whether the model's reasoning aligns with domain knowledge, requirements, or certain rules or principles (e.g., ethical). The techniques can aid in debugging unexpected behaviours, identifying reliance on spurious correlations, and verifying compliance with constraints (e.g., fairness). This helps address the "black box" challenge for validation purposes. The reliability and interpretation of explanations themselves require validation, and research is ongoing to understand the effectiveness and limitations of using AI explainability and interpretability for V&V tasks. In the context of edge machine vision, lightweight explainability methods can help assess whether the model's attention aligns with relevant image features. These methods assist in verifying that edge vision models respond to semantically appropriate cues and not to background artifacts or compression noise.

Neuro-Symbolic AI combines the strengths of data-driven neural networks (sub-symbolic AI) with rule-based logical reasoning (symbolic AI). The symbolic component can represent explicit domain knowledge, constraints, or reasoning rules, potentially making the hybrid system more interpretable, data-efficient, and robust. From a validation perspective, neuro-symbolic approaches offer promise by potentially enabling formal verification of the symbolic reasoning part, using symbolic knowledge to constrain or validate the neural network's outputs, and providing more transparent explanations for system behaviour. Research is actively exploring different integration architectures and their implications for validation.

Agentic AI and AI agents brings new challenges required the advancements of research focusing on developing new V&V techniques tailored to the dynamic nature of agentic AI, including advancing methods in runtime monitoring and formal verification that can cope with learning-based components and non-determinism. Creating simulation platforms that can model complex, real-world physics and multi-agent interactions will be crucial for testing edge systems exhaustively before deployment. The use of digital twin and immersive triplet environments could enable the safe exploration of an agent's behaviour under a wide range of standard and adverse conditions, helping to identify potential failure modes early. In this context, based on the technology trends research should address the system-level and collaborative aspects of agentic AI at the edge by creating frameworks for validating not only individual agents but also the collective, emergent behaviour of multi-agent systems. Developing techniques to ensure that the goals of individual agents remain aligned with the overall system objectives, even as they adapt and learn, is paramount. Research into explainable XAI and IAI for edge devices is required, as it will enable human operators to understand, trust, and effectively manage the decisions of autonomous agents, ensuring safe and predictable operation in complex, real-world scenarios.

1.9 Conclusion

The rapid advancement and deployment of edge AI necessitate a parallel evolution in the designers' ability to ensure that edge AI systems are safe, reliable, fair, and aligned with human values. Verification, as defined by standards such as ISO/IEC 22989, is the assurance through objective evidence that specified requirements have been fulfilled, forming a cornerstone of building essential trust. It provides the rigorous checks needed to confirm that AI systems are built according to their intended design and specifications.

The unique characteristics of AI, particularly its potential opacity, non-determinism, complex data dependencies, and difficulty in formally specifying requirements for emergent behaviours, pose significant challenges to traditional V&V approaches. The black-box nature of many models hinders direct inspection, scalability limitations restrict the application of formal methods, and the dynamic nature of edge AI systems and their environments demands continuous evaluation beyond design-time checks. Addressing conceptual challenges related to fairness, value alignment, and adversarial robustness requires ongoing fundamental research, and significant progress is being made. International standards provide common terminology (ISO/IEC 22989), frameworks for trustworthiness (ISO/IEC TR 24028), and management systems for responsible AI governance (ISO/IEC 42001). Methodologically, an increasing number of researchers are adopting formal methods for specific AI and edge AI verification tasks, developing advanced testing techniques (e.g., metamorphic and adversarial testing).

Metamorphic testing techniques are used to verify the behaviour of AI models, when predicting the exact output for a given input is challenging or impossible. The metamorphic testing techniques focus on identifying relationships between inputs and outputs, known as metamorphic relations, that act as logical rules or properties that should hold true when inputs are modified. Adversarial testing is a technique in which inputs are intentionally designed to expose weaknesses or flaws in a system, thereby identifying scenarios where the system produces harmful or biased outputs. This enables testers to identify vulnerabilities and ensure the system responds safely and effectively [37].

Generative AI excels at pattern recognition, classification, and predictive analytics, generates new patterns and multimodal content (e.g., text, sound, images) and plays a dual role in the verification and validation process, for example, as part of an edge AI system that has to be verified and validated and as a technology that supports the V&V processes by generating V&V requirements, specifications and automatically performing the V&V.

The V&V of emerging edge AI agents face challenges arising from the inherent autonomy and the dynamic environments in which these agents operate. The agents can rely on machine learning models that can produce non-deterministic outputs, making the behaviour difficult to predict and formally verify. Continuous interaction with external environments introduces an extensive and unpredictable operational space, where unforeseen events can lead to emergent behaviours that were not anticipated during the design and testing phases, posing risks to safety and reliability.

Further challenges for the V&V processes are the unique constraints of the edge environment itself. Edge AI systems must operate within the limitations of computational power, memory, and energy, which can impact the performance and consistency of their decision-making processes. Edge AI agents must make real-time decisions, where latency is a critical factor. Validating that an edge AI agent responds correctly and within strict time constraints, especially when facing intermittent connectivity or degraded sensor input, is a significant hurdle that requires novel testing methodologies beyond traditional software V&V.

The process of V&V agentic edge AI systems requires addressing the interaction between human and AI agent components to ensure that the agent's behaviour is understandable and transparent to human users, which allows for efficient oversight and human intervention when necessary. The V&V process must confirm that the edge system can communicate its state and intentions evidently and that its autonomous actions are auditable, interpretable and explainable.

This supports explainable AI techniques, implementing runtime verification for operational assurance, and opening the use of neuro-symbolic architectures to bridge the gap between learning and reasoning. Neuro-symbolic AI is a type of AI that integrates neural and symbolic AI architectures to address the weaknesses of each, providing a robust AI capable of reasoning, learning, and cognitive modelling.

Continued research is essential to develop more scalable and robust verification techniques that can handle the complexity of new edge AI systems. Addressing foundational AI safety problems, enhancing automated and human-centric V&V approaches, and building comprehensive, trustworthy AI frameworks that integrate technical verification with ethical considerations and governance are key priorities. Achieving verifiably trustworthy AI requires a holistic perspective, acknowledging the interplay between hardware, software, AI models, data, systems, processes, and the technical, application, and environmental contexts [40].

Achieving the goal of trustworthy AI and edge AI systems that are demonstrably beneficial and responsibly integrated into society requires elevating validation beyond a mere technical, end-of-phase check. It demands a holistic, continuous, and lifecycle-integrated approach [54].

This approach must rigorously integrate technical validation (ensuring robustness, reliability, and security) with user-centric validation (confirming usability, fitness for purpose, meeting needs), ethical validation (assessing fairness, accountability, and value alignment), and real-world effectiveness monitoring [90].

Success requires multidisciplinary collaboration, bringing together AI/ML experts, software engineers, domain specialists, human factors engineers, ethicists, social scientists, legal experts, end-users, and regulators. International standard bodies like ISO and organisations like NIST provide essential frameworks, common terminology, and guidance (e.g., ISO 9000, ISO/IEC/IEEE 15288, ISO/IEC 22989, ISO/IEC TR 24028, ISO/IEC 42001, NIST AI RMF) [89].

The rapidly evolving nature of AI means that significant ongoing research and innovation in validation techniques are imperative to address the challenges effectively.

Edge AI system validation is a dynamic and increasingly critical field. As AI capabilities continue to advance and these systems become more deeply embedded in our lives, the methods used to ensure they are fit for purpose, safe, and aligned with human values must also evolve.

The focus is shifting from static, pre-deployment checks towards continuous, adaptive, and context-aware validation processes that span the entire edge AI lifecycle. Addressing the complex technical, ethical, and societal challenges associated with edge AI validation requires sustained research, multidisciplinary collaboration, and international cooperation.

Continued innovation in validation methodologies and tools will be essential to harness the transformative potential of AI responsibly and build a future where edge AI systems are trustworthy and integrated into industrial and business processes.

Acknowledgements

This publication has received funding through the projects Chips JU EdgeAI and HE dAIEDGE. The Chips JU EdgeAI "Edge AI Technologies for Optimised Performance Embedded Processing" project is supported by the Chips Joint Undertaking and its members including top-up funding by Austria, Belgium, France, Greece, Italy, Latvia, Netherlands, and Norway under grant agreement No 101097300. The HE dAIEDGE "A network of excellence for distributed, trustworthy, efficient and scalable AI at the Edge" project is supported under grant agreement No 101120726. Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the Chips Joint Undertaking. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] A. Lavin et al., "Technology readiness levels for machine learning systems," Nature Communications, vol. 13, no. 1, p. 6039, Oct. 2022, https://doi.org/10.1038/s41467-022-33128-9.

[2] S. Mahmud, S. Saisubramanian, and S. Zilberstein, "Verification and Validation of AI Systems Using Explanations," *Proceedings of the AAAI Symposium Series*, vol. 4, no. 1, pp. 76–80, Nov. 2024, https://doi.org/10.1609/aaaiss.v4i1.31774.

[3] "Verification and Validation of Systems in Which AI is a Key Element - SEBoK," *sebokwiki.org*. https://sebokwiki.org/wiki/Verification_and_Validation_of_Systems_in_Which_AI_is_a_Key_Element.

[4] "ISO/IEC TR 24028:2020 – Overview of trustworthiness in artificial intelligence," *BSI*, 2020. https://www.bsigroup.com/en-IN/training-courses/isoiec-tr-240282020--overview-of-trustworthiness-in-artificial-intelligence/.

[5] "ISO/IEC TR 24028:2020 - Information technology - Artificial intelligence - Overview of trustworthiness in artificial intelligence" *ISO*. https://www.iso.org/standard/77608.html.

[6] NIST, "AI Risk Management Framework," *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*, vol. 1, Jan. 2023, https://doi.org/10.6028/nist.ai.100-1.

[7] J. Jeon, "Standardization Trends on Safety and Trustworthiness Technology for Advanced AI", 2024, https://arxiv.org/abs/2410.22151.

[8] T. R. McIntosh, T. Susnjak, T. Liu, P. Watters, and M. N. Halgamuge, "Inadequacies of Large Language Model Benchmarks in the Era of Generative Artificial Intelligence," *arXiv (Cornell University)*, Feb. 2024. Available at: https://doi.org/10.48550/arxiv.2402.09880

[9] "ISO/IEC 22989:2022 – Information technology – Artificial intelligence – Artificial intelligence concepts and terminology," Edition 1, 2022. https://www.iso.org/standard/74296.html

[10] "Recommendation of the Council on Artificial Intelligence," OECD Legal Instruments, 2025. https://legalinstruments.oecd.org/en/instruments/oecd-legal-0449

[11] "What is catastrophic forgetting?" IBM, April 2025. https://www.ibm.com/think/topics/catastrophic-forgetting

[12] T. Meuser, et.al. "Revisiting Edge AI: Opportunities and Challenges". IEEE Internet Computing, vol. 28, July-August 2024. https://www.computer.org/csdl/magazine/ic/2024/04/10621659/1Z5lGDb639C

[13] Z. Ren and C. J. Anumba, "Multi-agent systems in construction–state of the art and prospects," Automation in Construction, vol. 13, no. 3, pp. 421–434, 2004, https://doi.org/10.1016/j.autcon.2003.12.002.

[14] G. Papagni, J. de Pagter, S. Zafari, M. Filzmoser, and S. T. Koeszegi, "Artificial agents' explainability to support trust: considerations on

timing and context," AI & Society, Vol. 38, No. 2, pp. 947–960, 2023. https://doi.org/10.1007/s00146-022-01462-7

[15] P. Wang and H. Ding, "The rationality of explanation or human capacity? Understanding the impact of explainable artificial intelligence on human-AI trust and decision performance," Information Processing & Management, Vol. 61, No. 4, p. 103732, 2024. https://doi.org/10.1016/j.ipm.2024.103732.

[16] F. Sado, C. K. Loo, W. S. Liew, M. Kerzel, and S. Wermter, "Explainable Goal-driven Agents and Robots - A Comprehensive Review", ACM Computing Surveys, Volume 55, Issue 10, pp. 1–41, 2023. https://doi.org/10.1145/3564240.

[17] R. Sapkota, K. I. Roumeliotis, and M. Karkee, "AI Agents vs. Agentic AI: A Conceptual Taxonomy, Applications and Challenge," arXiv.org, 2025. https://arxiv.org/abs/2505.10468.

[18] C. Riedl and D. De Cremer, "AI for collective intelligence," Collective Intelligence, vol. 4, no. 2, Apr. 2025, https://doi.org/10.1177/26339137251328909.

[19] F. Piccialli, D. Chiaro, S. Sarwar, D. Cerciello, P. Qi, and V. Mele, "AgentAI: A comprehensive survey on autonomous agents in distributed AI for industry 4.0," Expert Systems with Applications, vol. 291, p. 128404, Oct. 2025, https://doi.org/10.1016/j.eswa.2025.128404.

[20] W. Xu, Z. Liang, K. Mei, H. Gao, J. Tan, and Y. Zhang, "A-MEM: Agentic Memory for LLM Agents," arXiv.org, 2025. https://arxiv.org/abs/2502.12110.

[21] D. B. Acharya, K. Kuppan and B. Divya, "Agentic AI: Autonomous Intelligence for Complex Goals—A Comprehensive Survey," in IEEE Access, vol. 13, pp. 18912-18936, 2025, https://www.doi.org/10.1109/ACCESS.2025.3532853.

[22] R. Zhang et al., "Toward Agentic AI: Generative Information Retrieval Inspired Intelligent Communications and Networking," arXiv.org, 2025. https://arxiv.org/abs/2502.16866.

[23] M. Gridach, J. Nanavati, K. Zine, L. Mendes, and C. Mack, "Agentic AI for Scientific Discovery: A Survey of Progress, Challenges, and Future Directions," arXiv.org, 2025. https://arxiv.org/abs/2503.08979.

[24] E. Miehling et al., "Agentic AI Needs a Systems Theory," arXiv.org, 2025. https://arxiv.org/abs/2503.00237.

[25] S. Hong et al., "MetaGPT: Meta Programming for Multi-Agent Collaborative Framework," arXiv.org, Aug. 07, 2023. https://arxiv.org/abs/2308.00352.

[26] U. M. Borghoff, P. Bottoni, and R. Pareschi, “Human-artificial interaction in the age of agentic AI: a system-theoretical approach,” Frontiers in Human Dynamics, vol. 7, May 2025, https://doi.org/10.3389/fhumd.2025.1579166.

[27] J. Heer, “Agency plus automation: Designing artificial intelligence into interactive systems,” Proceedings of the National Academy of Sciences, Vol. 116, No. 6, pp. 1844–1850, 2019. https://doi.org/10.1073/pnas.1807184115.

[28] E. Oliveira, K. Fischer, and O. Stepankova, “Multi-agent systems: which research for which applications,” Robotics and Autonomous Systems, vol. 27, no. 1-2, pp. 91–106, 1999, https://doi.org/10.1016/S0921-8890(98)00085-2.

[29] Validation - Glossary | CSRC - NIST Computer Security Resource Center, https://csrc.nist.gov/glossary/term/validation

[30] Verification and validation - Wikipedia, https://en.wikipedia.org/wiki/Verification_and_validation

[31] Design Review, Verification and Validation - Quality Gurus, https://www.qualitygurus.com/design-review-verification-and-validation/

[32] Verification Versus Validation – What’s the Difference? - Climedo, https://climedo.de/en/blog/verification-versus-validation-whats-the-difference/

[33] System Validation - SEBoK, https://sebokwiki.org/wiki/System_Validation

[34] Implementing ISO 15288 V&V Processes using the V&V Studio - The Reuse Company, https://www.reusecompany.com/wp-content/uploads/2021/02/VV-Studio-Webinar-Jan-2021.pdf

[35] Verification (glossary) - SEBoK, https://sebokwiki.org/wiki/Verification_(glossary)

[36] “Sapien’s AI Glossary of Data Terms, Definitions & Insights,” Sapien.io, 2025. https://www.sapien.io/glossary/all

[37] A. Pande, “Metamorphic and adversarial strategies for testing AI systems,” Ministry of Testing, Jan 14, 2025. https://www.ministryoftesting.com/articles/metamorphic-and-adversarial-strategies-for-testing-ai-systems

[38] ISO and IEC Make Foundational Standard on Artificial Intelligence Publicly Available, https://www.holisticai.com/news/iso-iec-22989-foundational-standard-on-ai-open-source

[39] The foundational standards for AI | JTC 1, https://jtc1info.org/wp-content/uploads/2022/06/03_08_Paul_Milan_Wei_The-foundational-standards-for-AI-20220525-ww-mp.pdf

[40] E. Manziuk, O. Barmak, I. Krak, O. Mazurets, and T. Skrypnyk, "Formal Model of Trustworthy Artificial Intelligence Based on Standardization," CEUR-WS.org, https://ceur-ws.org/Vol-2853/short18.pdf
[41] ISO/IEC TR 24028:2020 - Information technology - Artificial intelligence - OECD.AI, https://oecd.ai/en/catalogue/tools/isoiec-tr-240282020-information-technology-artificial-intelligence-overview-of-trustworthiness-in-artificial-intelligence
[42] Exploring the landscape of trustworthy artificial intelligence: Status and challenges, https://content.iospress.com/articles/intelligent-decision-technologies/idt240366
[43] ISO 42001 Artificial Intelligence Management System - Amazon Web Services (AWS), https://aws.amazon.com/compliance/iso-42001-faqs/
[44] ISO 42001 - AI Management System - BSI, https://www.bsigroup.com/en-US/products-and-services/standards/iso-42001-ai-management-system/
[45] ISO/IEC 22989:2023 Understanding AI Concepts and Definitions Training Course | BSI, https://www.bsigroup.com/en-ID/training-courses/isoiec-229892023-understanding-ai-concepts-and-definitions-training-course/
[46] AI Compliance Audit: Step-by-Step Guide - Dialzara, https://dialzara.com/blog/ai-compliance-audit-step-by-step-guide/
[47] R. Prieto, "Verification and Validation of Project Management Artificial Intelligence Key Points," Jun. 2020. https://www.researchgate.net/publication/342452507_Verification_and_Validation_of_Project_Management_Artificial_Intelligence_Key_Points
[48] AI audit checklist (updated 2025) | Complete AI audit procedures | Technical evaluation framework | System reliability guide | Compliance checklist | Lumenalta, https://lumenalta.com/insights/ai-audit-checklist-updated-2025
[49] Y. Wang and S. H. Chung. "Artificial intelligence in safety-critical systems: a systematic review," Industrial Management & Data Systems,| Emerald Insight, Dec. 2021. https://www.emerald.com/insight/content/doi/10.1108/imds-07-2021-0419/full/html
[50] Trustworthy AI - AI@UCSF - University of California San Francisco, https://ai.ucsf.edu/trustworthy
[51] AI Risks and Trustworthiness - NIST AIRC - National Institute of Standards and Technology, https://airc.nist.gov/airmf-resources/airmf/3-sec-characteristics/

[52] S. A. Seshia, D. Sadigh, and S. Shankar Sastry. Toward Verified Artificial Intelligence. Communications of the ACM, July 2022. https://cacm.acm.org/research/toward-verified-artificial-intelligence/
[53] AI, Opacity, and Personal Autonomy, https://d-nb.info/1275205275/34
[54] Measure - NIST AIRC - National Institute of Standards and Technology, https://airc.nist.gov/airmf-resources/playbook/measure/
[55] Understanding the NIST AI RMF: What It Is and How to Put It Into Practice - Secureframe, https://secureframe.com/blog/nist-ai-rmfy
[56] AI Life Cycle Core Principles - CodeX - Stanford Law School, https://law.stanford.edu/2023/03/17/ai-life-cycle-core-principles/
[57] Ethical and societal implications of algorithms, data, and artificial intelligence: a roadmap for research - Nuffield Foundation, https://www.nuffieldfoundation.org/sites/default/files/files/Ethical-and-Societal-Implications-of-Data-and-AI-report-Nuffield-Foundat.pdf
[58] Messages on "When using AI systems, what are some best practices for ensuring the results you receive are accurate, relevant, and aligned with your original goals?" - ProjectManagement.com, https://www.projectmanagement.com/discussion-topic/203772/when-using-ai-systems--what-are-some-best-practices-for-ensuring-the-results-you-receive-are-accurate--relevant--and-aligned-with-your-original-goals-?sort=asc&pageNum=39
[59] A Framework for the Verification and Validation of Artificial Intelligence Machine Learning Systems - JagWorks@USA - University of South Alabama, https://jagworks.southalabama.edu/theses_diss/137/y
[60] Trustworthy AI - The Data Science Institute at Columbia University, https://datascience.columbia.edu/news/2020/trustworthy-ai/
[61] NIST launches ARIA program to assess societal impacts, ensure trustworthy AI systems, https://industrialcyber.co/ai/nist-launches-aria-program-to-assess-societal-impacts-ensure-trustworthy-ai-systems/
[62] User Acceptance Testing (UAT): Definition, Process, and Tools - LambdaTest, https://www.lambdatest.com/learning-hub/user-acceptance-testing
[63] Human-AI Interaction and User Satisfaction: Empirical Evidence from Online Reviews of AI Products - ResearchGate, https://www.researchgate.net/publication/390142284_Human-AI_Interaction_and_User_Satisfaction_Empirical_Evidence_from_Online_Reviews_of_AI_Products/download

[64] J. Renkhoff, K. Feng, M. Meier-Doernberg, A. Velasquez, and H. H. Song, “A Survey on Verification and Validation, Testing and Evaluations of Neurosymbolic Artificial Intelligence,” IEEE transactions on artificial intelligence, pp. 1–15, Jan. 2024, https://doi.org/10.1109/tai.2024.3351798.

[65] A. E. Goodloe, “Assuring Safety-Critical Machine Learning-Enabled Systems: Challenges and Promise,” *Computer*, vol. 56, no. 9, pp. 83–88, Sep. 2023, https://doi.org/10.1109/mc.2023.3266860

[66] A. Woodie, “Top 10 Challenges to GenAI Success,” BigDATAwire, Jan. 22, 2024. https://www.bigdatawire.com/2024/01/22/top-10-challenges-to-genai-success/

[67] C. Bronsdon, “AI Safety Metrics: How to Ensure Secure and Reliable AI Applications” - Galileo AI, 2025, https://www.galileo.ai/blog/introduction-to-ai-safety

[68] R. Camacho, “A Practical Guide for AI in Safety-Critical Embedded Systems - Parasoft, 2025, https://www.parasoft.com/blog/ai-in-safety-critical-embedded-systems/

[69] Y. Y. Elboher et., al “Formal Verification of Deep Neural Networks for Object Detection,” Arxiv.org, 2023. https://arxiv.org/html/2407.01295

[70] What are non-deterministic AI outputs? - Statsig, 2024, https://www.statsig.com/perspectives/what-are-non-deterministic-ai-outputs-

[71] K. Leahy et al., “Grand Challenges in the Verification of Autonomous Systems,” arXiv (Cornell University), Nov. 2024, https://doi.org/10.48550/arxiv.2411.14155.

[72] Symbolic AI vs. Deep Learning: Key Differences and Their Roles in AI Development, https://smythos.com/ai-agents/agent-architectures/symbolic-ai-vs-deep-learning/

[73] Symbolic AI vs. Machine Learning: A Comprehensive Guide - SmythOS, https://smythos.com/ai-agents/ai-tutorials/symbolic-ai-vs-machine-learning/

[74] O. Vermesan, V. Piuri, F. Scotti, A. Genovese, R. D. Labati, and P. Coscia, “Explainability and Interpretability Concepts for Edge AI Systems,” River Publishers eBooks, pp. 197–227, Feb. 2024, https://doi.org/10.1201/9781003478713-9.

[75] What Is Explainable AI (XAI)? Palo Alto Networks, https://www.paloaltonetworks.com/cyberpedia/explainable-ai

[76] Deep Learning’s Challenges and Neurosymbolic AI’s Solutions - AskUI, 2024. https://www.askui.com/blog-posts/deep-learnings-challenges-and-neurosymbolic-ais-solutions

[77] V. Musanga, S. Viriri, and C. Chibaya, "A Framework for Integrating Deep Learning and Symbolic AI Towards an Explainable Hybrid Model for the Detection of COVID-19 Using Computerized Tomography Scans," Information, vol. 16, no. 3, p. 208, Mar. 2025, https://doi.org/10.3390/info16030208.

[78] X. Zhang, "Apex.OS: Breaking Barriers in Autonomous Verification & Validation", 2024, https://www.apex.ai/post/apex-os-breaking-barriers-in-autonomous-verification-validation

[79] J. Szarmach, NIST: Reducing Risks Posed by Synthetic Content An Overview of Technical Approaches to Digital Content Transparency, 2025, https://www.aigl.blog/nist-reducing-risks-posed-by-synthetic-content-an-overview-of-technical-approaches-to-digital-content-transparency/

[80] NIST Trustworthy and Responsible AI - NIST AI 100-4. Reducing Risks Posed by Synthetic Content. An Overview of Technical Approaches to Digital Content Transparency. https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-4.pdf

[81] NIST. The United States Artificial Intelligence Safety Institute: Vision, Mission, and Strategic Goals, 2024, https://www.nist.gov/document/aisi-strategic-vision-document

[82] S. Dahaweer, "How AI is going to revolutionize safety in critical applications", Alithya, 2023, https://www.alithya.com/en/insights/blog-post/how-ai-going-revolutionize-safety-critical-applications

[83] The Top 10 Unsolved Challenges in AI: A 2024 Retrospective, gekko, 2024, https://gpt.gekko.de/unsolved-challenges-in-ai-2024/

[84] Risks from AI - An Overview of Catastrophic AI Risks, CAIS - Center for AI Safety, https://www.safe.ai/ai-risk

[85] R. Lubecki, "Verifying and Validating AI/ML" - UpCity, 2021, https://upcity.com/experts/verifying-and-validating-ai-ml/

[86] K. Lekadir et al., "FUTURE-AI: International consensus guideline for trustworthy and deployable artificial intelligence in healthcare," BMJ, vol. 388, p. e081554, Feb. 2025, https://doi.org/10.1136/bmj-2024-081554.

[87] Traditional AI vs. Generative AI: What's the Difference? - College of Education, Illinois, 2024, https://education.illinois.edu/about/news-events/news/article/2024/11/11/what-is-generative-ai-vs-ai

[88] N. Ahsan, "Why Enterprises Are Adopting Domain-Specific AI Agents", Vidizmo, 2025, https://vidizmo.ai/blog/why-domain-specific-ai-agents-are-key-to-business-success

[89] Future of AI Research – AAAI – Association for the Advancement of Artificial Intelligence 2025. Available: https://aaai.org/wp-content/uploads/2025/03/AAAI-2025-PresPanel-Report-FINAL.pdf

[90] Methods For Verifying AI Reliability to Ensure AI Safety, IAI – Institute for AI Transformation, August 2024. Available: https://www.leadersinaisummit.com/insights/methods-for-verifying-ai-reliability-to-ensure-ai-safety

[91] IEEE 1012-2016. IEEE Standard for System, Software, and Hardware Verification and Validation. https://webstore.ansi.org/standards/ieee/ieee10122016?source=blog

[92] O. Vermesan, K. De Bosschere, T. Vardanega, S. Azaiez, M. Duranton, R. Badia, and D. Lezzi (Eds.). "Distributed Computing and Swarm Intelligence - Developing a Vision for Transatlantic Collaboration," Zenodo, Feb. 2025, https://doi.org/10.5281/zenodo.14940197

2

Pioneering the Hybridization of Federated Learning in Human Activity Recognition

Alfonso Esposito[1], Yasamin Moghbelan[1], Ivan Zyrianoff[1], Leonardo Ciabattini[1], Federico Montori[1,2], and Marco Di Felice[1,2]

[1]University of Bologna, Italy
[2]University of Bologna, Advanced Research Center for Electronic Systems, Italy

Abstract

The Internet of Things (IoT) nowadays greatly benefits from Artificial Intelligence (AI) algorithms implemented in the edge, because of their efficiency and the reduction of costs that they imply. The advent of Federated Learning (FL) has made possible the combination of the advantages of edge-AI, among which the privacy of users, as their data is not shared with the cloud, with the collective intelligence. However, FL is known to have worse performance compared to its centralized counterpart, which may not be tolerable in certain cases. In this paper, we propose a hybrid framework for FL, imagining a number of clients that are willing to share part of their data. We envisioned two types of Hybridization: vertical and horizontal. The goal of this paper is to assess whether a small hybridization can bring advantages to the overall performance of the whole FL procedure in terms of classification accuracy.

Keywords: Internet of Things (IoT), deep learning (DL), federated learning (FL), human activity recognition (HAR), performance evaluation.

2.1 Introduction and Background

The growth of the Internet of Things (IoT) has enabled novel monitoring systems capable of gathering data from heterogeneous and pervasive devices, supporting smart city, healthcare and industrial domains. This data

DOI: 10.1201/9788743808862-2

is processed using advanced Deep Learning (DL) techniques for system state forecasting, contributing to the integrated paradigm of the Artificial Intelligence of Things (AIoT) [1]. However, it is well known that advanced DL techniques, particularly those used in computer vision applications, require large datasets with adequate number of instances for each system state or class to be predicted. In many cases, building reliable DL models requires aggregating data from multiple heterogeneous users or organizations performing decentralized data collection for a common task. This is the case, for instance, of most of Human Activity Recognition (HAR) applications that utilize DL models trained from wearable or camera-based IoT data gathered from multiple volunteers or via crowdsensing techniques [2].

State-of-the-art AIoT systems often rely on cloud-based centralized architectures, which facilitate the aggregation of IoT data from diverse clients. While these solutions enable easy deployment and scalable computational and storage resources, they also raise significant privacy concerns due to the inherent risks associated with data sharing. Federated Learning (FL), first proposed in 2018 [3], has emerged as a privacy-preserving alternative to centralized machine learning, enabling cooperative training in a distributed environment. In a typical FL setup, multiple clients independently train models on their private datasets and only share the trained weights with a central server, which aggregates these weights and sends the updated model back to the clients. This approach ensures that no raw data is exchanged among clients, although concerns about the trustworthiness of the centralized server remain [4]. Moreover, variations in data quality and quantity across clients can create challenges in ensuring fair evaluation of each client's contribution and achieving a high-quality global model [5]. To address the issue of non-i.i.d. (non-independent and identically distributed) data across clients, various solutions have been proposed, such as clustering clients with similar data distributions or adopting weight-based model aggregation techniques [6].

In this paper, we aim to bridge the gap between centralized and Federated Learning (FL) methodologies, in order to address distributed IoT use cases where privacy requirements vary from client to client. For instance, some clients may be willing to share raw data, while others may not, due to differences in client nature (e.g., public vs. private organizations), varying perceptions of privacy—often shaped by social factors [7]—or monetization strategies, where some clients are incentivized to sell their data. We refer to this scenario as *hybrid FL* and describe this *hybridization* approach for data gathering and model building at the server side, which offers a novel interpretation compared to other studies [4]. Specifically, this paper explores two types of FL hybridization: *vertical* and *horizontal*. In the vertical hybridization, a portion of clients shares raw data with the centralized server, while others only share deep learning model coefficients generated from local training. In the horizontal case, all clients share a variable portion of their raw data (e.g., corresponding to the amount for which they have been compensated) in addition to the DL model coefficients trained on the complementary data. We evaluate the performance of both strategies using two benchmark datasets (related to Human Activity Recognition and image classification) and analyze the impact of different levels of hybridization compared to pure centralized and FL-based approaches. Our results demonstrate that hybridization can be a powerful tool for improving the accuracy of federated systems, even when applied slightly.

To summarize, the key contributions of our paper are the following:

- Introduction of hybrid FL as a new strategy for privacy-adaptive learning in IoT scenarios.
- Proposal of two versions of hybrid FL, respectively horizontal and vertical, based on distinct methods of integrating raw data and deep learning weights at the server.
- Evaluation of proposed strategies across varying degrees of hybridization, using two widely adopted benchmark datasets in the DL community.

The rest of the paper is structured as follows. Section 2 introduces the revised FL architecture supporting the vertical and horizontal hybridization. Section 3 describes the evaluation methodology, datasets and metrics. Section 4 presents some evaluation results. Section 5 concludes the work and discusses future research steps.

2.2 Hybrid FL Architecture

We consider the scenario depicted in Figure 2.1, composed of two main actors: N distributed clients and the server. Each client c_i possesses its own dataset D_i gathered through its local sensors, and a DL network topology, denoted as m in the following. In a classical FL application, each client c_i performs local training rounds of model m on D_i and then shares the list of weights W_i with the server: the latter is responsible for aggregating the weights, for instance through the FedAvg [3] algorithm, and sending the updated values back to the clients.

In the proposed hybrid FL architecture, the server performs additional storage and computational tasks, basically behaving as a special client device.

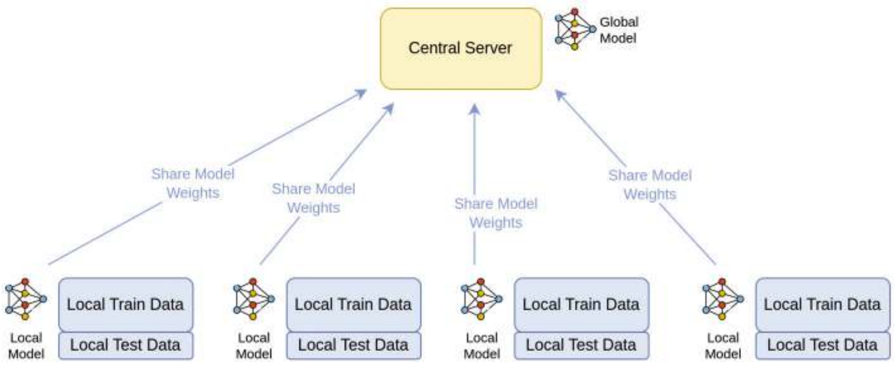


Figure 2.1 Typical Federated Learning Architecture ⏎

Indeed, it possesses a local dataset D_s that is built from clients' datasets or portions of them. More formally, we have that $D_s = \bigcup_{i=0}^{n} D_i^*$ where $n \leq$N and $D_i^* \subseteq D_i$. The server trains the m model on D_s getting a local version of weights W_S that is later averaged with the weights W_i received by clients at each round. We denoted this process as **FL hybridization**, distinguishing between two modes for creating the local dataset D_s:

- **Vertical Hybrid FL**. In such case, we have that: $n < N$ and $D_i^* = D_i$. In other words, only a subset of the client nodes shares its own raw data with the server. This setup may model two different use-cases: (*i*) only n clients have been compensated with monetary rewards to share their data and/or (*ii*) n clients do not consider their local datasets privacy-sensitive. We indicate with $r_v = \frac{n}{N}$ the rate of vertical hybridization. Clearly, for $n = N$ and $r_v = 1$, the system is equivalent to centralized learning. Vice versa, for $n = 0$ and $r_v = 0$, the system is equivalent to a pure FL approach. We investigate the impact of varying r_v configurations in the overall DL performance in Section 4. We show in Figure 2.2 an overview on the conceptual architecture of Vertical Hybrid FL.
- **Horizontal Hybrid FL**. In such case, we have that: $n = N$ and $D_i^* \subset D_i$. In other words, all clients share only a portion of their local datasets with the server. This setup may model a practical use-case where each client shares with the server an amount of raw data proportionally to the monetary reward it received. We indicate with $r_h = mean_{0 \leq i < N}\left(\frac{|D_i^*|}{|D_i|}\right)$ the rate of vertical hybridization. Clearly, for $D_i^* = D_i$, the system becomes a centralized learning. Vice versa, if $D_i^* = \emptyset \; \forall c_i$ the system is

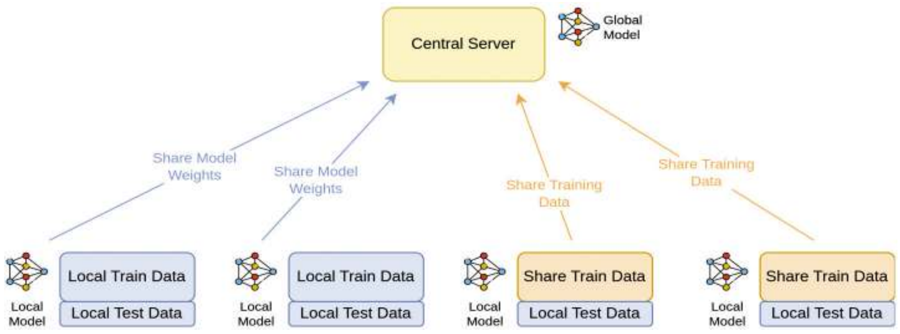


Figure 2.2 Vertical Hybrid Federated Learning Architecture ⏎

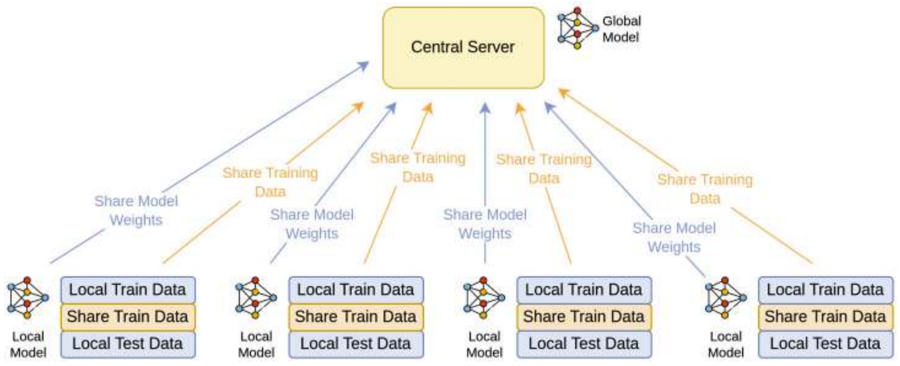


Figure 2.3 Horizontal Hybrid Federated Learning Architecture ⏎

equivalent to a pure FL approach. We investigate the impact of varying r_h configurations in the overall DL performance in Section 4. We show in Figure 2.3 an overview on the conceptual architecture of Vertical Hybrid FL.

We further highlight that the hybridization rates determine the amount of required privacy preservation. The maximum value (1) corresponds to when all clients share their local data with the server. Vice versa, the minimum value (0) forbids any transmission of raw data outside the clients' devices.

2.3 Evaluation Methodology and Metrics

In this section, we describe the experiments that we performed to investigate the performances of the two hybrid approaches explained in the previous section. First, we will describe the methodology used to test the effectiveness of HFL across two datasets: FEMNIST and UCI HAR.

The widely recognized University of California Irvine (UCI) HAR dataset [8] was created using data from smartphone accelerometer and gyroscope sensors, which were used to classify six types of human activities. It was gathered from 30 volunteers, each carrying a smartphone while performing six distinct activities: walking, walking upstairs, walking downstairs, sitting, standing, and lying down. The dataset consists of time-series data captured across the three axes of both sensors, along with the corresponding activity labels. It has been extensively used in research for developing and assessing HAR models using machine learning techniques. Each record in the dataset contains a vector of 561 features, derived from both time and frequency domain calculations.

The FEMNIST dataset [9] is an adaptation of the extended version of the MNIST dataset that has been modified to be suitable for FL tasks. The MNIST dataset contains 28 by 28 pixels images of handwritten digits and characters (62 classes in total), and the goal of the DL model is to guess the actual character represented. The FEMNIST dataset groups the elements on top of the user that actually performed the handwriting, producing a number of sub-datasets each of them with a different style of writing. The number of users of the FEMNIST dataset is 3500, however, for the purpose of our experiment, we considered only 30 users, in order to make experiments comparable between the two datasets.

For UCI HAR we employed, as a base local model, a simple feed-forward neural network, while for FEMNIST we adopted a convolutional network. Each of the local sub-datasets is split into training and test set using a stratified split with a 70%-30% ratio.

We performed federated classification experiments by employing 6 epochs and 20 rounds of federation, recording the accuracy score at the end of the last round.

We tested both vertical and horizontal hybridization, by setting alternately r_v and r_h to values spanning from 0% to 100% with a 10% step.

The experiments were implemented in Python using the Flower framework (https://flower.readthedocs.io/en/latest/). Since Flower does not support the implementation of hybridization, we adopted the two following methods to simulate the two hybridization methods (as Figure 2.4 suggests):

- Vertical Hybridization was simulated by aggregating all the clients that share their whole dataset instead of the weights into a single client.

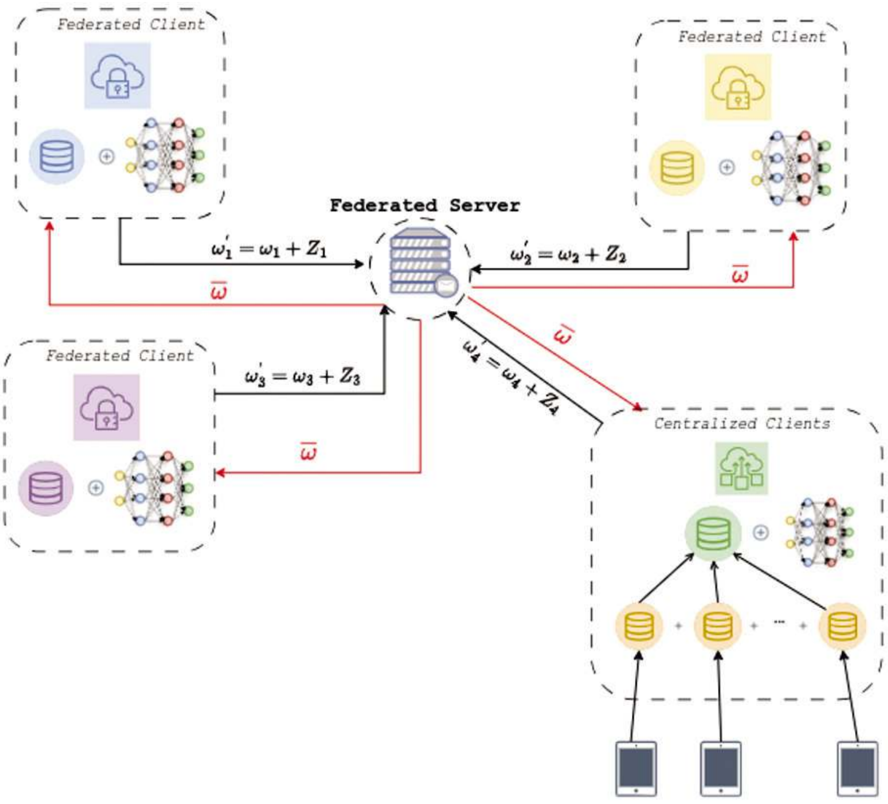

Federated Client
Federated Client
Federated Client
Federated Server
Centralized Clients
$\omega_1' = \omega_1 + Z_1$
$\omega_2' = \omega_2 + Z_2$
$\omega_3' = \omega_3 + Z_3$
$\omega_4' = \omega_4 + Z_4$
$\overline{\omega}$

Figure 2.4 Implementation of the Vertical Hybridization in Flower ⏎

- Horizontal Hybridization was simulated by extracting from each client the portion of the training dataset that they aim to share and assigning it to a new "sink" client.

Each experiment was then repeated 20 times, by randomizing the clients or the dataset portions to be shared. This ensures scientific rigor and smooths out certain corner situations that may arise.

2.4 Evaluation Results

This section presents the outcome of the experiments presented in the previous section. The results are aimed at evaluating the HFL approaches on the datasets. We investigate how different levels of data sharing in vertical and horizontal hybridization affect model performance. We first discuss the results for the UCI HAR dataset, followed by FEMNIST, to uncover any dataset-specific trends and performance differences. A general overview of the results shows, as expected, an overall improvement in model performance as the degree of hybridization increases, which is notable for both hybrid approaches. The UCI HAR dataset performed best. Even with the relatively small model size, it consistently achieved excellent results, maintaining accuracy above 90% and reaching almost 95% in experiments with higher levels of hybridisation. This is shown in Figures 2.5 and 2.6, where the increase in model performance as the level of hybridisation increases is evident, with an increase of 2 percentage points already at a low level of horizontal hybridization (20%).

The FEMNIST is the dataset where the performance improvements from sharing data is most noticeable. As we can observe in the Figures 2.7 and 2.8, the sharing of a small number of data points could lead to a significant improvement in performance. Specifically, sharing 10% of the data led to an improvement of approximately 5% in accuracy, while sharing 20% led to an additional improvement of approximately 2/3, resulting in a final performance of 88%. This is comparable to the 90% accuracy obtained through centralized training. Beyond a data sharing rate of 30%, the improvements

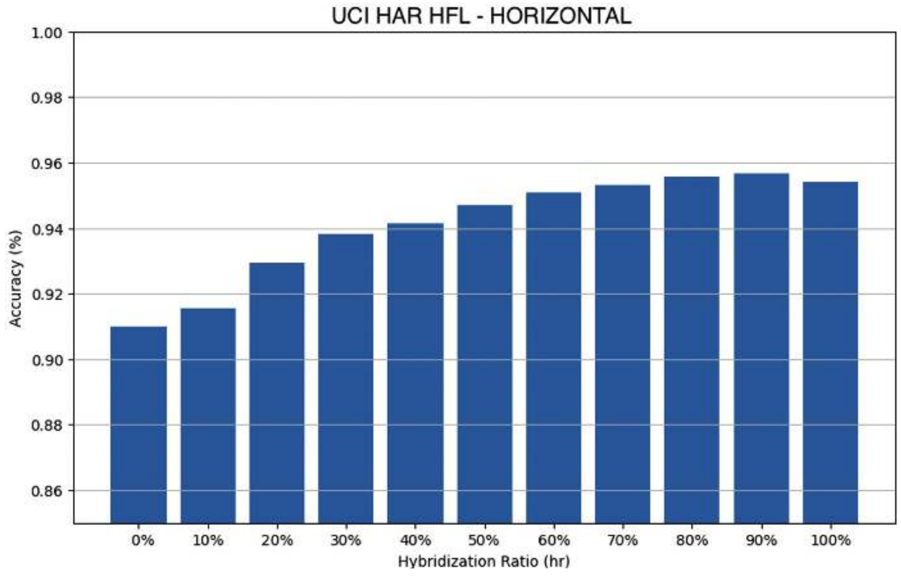


Figure 2.5 Horizontal Hybridization Results for UCI HAR ⏎

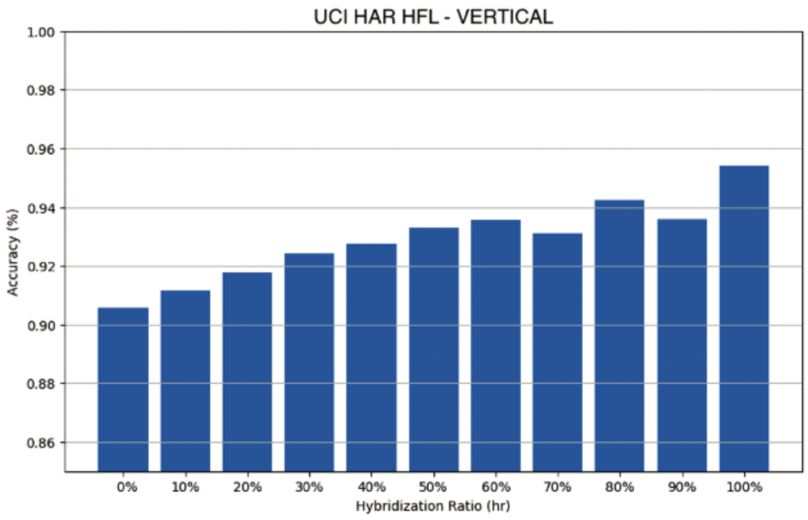


Figure 2.6 Vertical Hybridization Results for UCI HAR ⏎

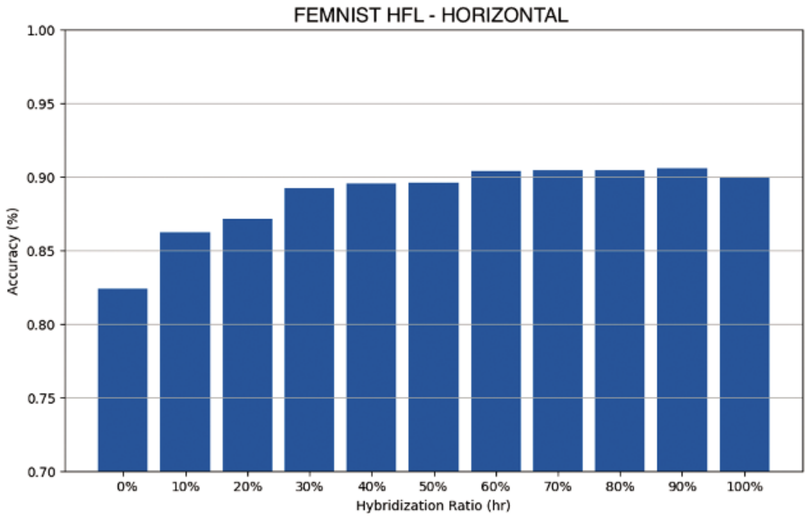


Figure 2.7 Horizontal Hybridization Results for FEMNIST ⏎

obtained are increasingly marginal, with a maximum of 1%. This suggests that a data sharing rate of 20% represents an optimal balance between performance and data sharing.

The results demonstrate that the sharing of a portion of the dataset has a significant positive impact on the model's performance. This effect was

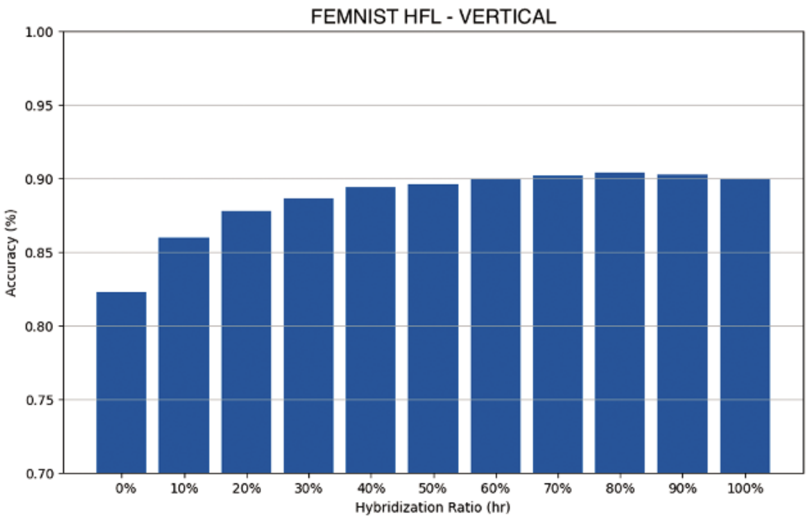


Figure 2.8 Vertical Hybridization Results for FEMNIST ⏎

observed across two distinct datasets, UCI HAR and FEMNIST, indicating that the improvement is not specific to any problem or model. The performance enhancement was especially evident in the case of the FEMNIST dataset.

2.5 Conclusion and Future Works

In this paper we examined the effects of hybridization for Federated Learning scenarios. We specifically directed our research towards Human Activity Recognition, imagining scenarios in which certain clients would be willing to share (part of) their data with the central server for a reward, penalizing their privacy to an extent. Results showed that a minimal amount of hybridization does provide an increase in the performance. The extent to which the privacy of the users is compromised by this is a future work. We aim to study how to select carefully data in a way in which the privacy is minimally affected, as well as to blend the two hybridization techniques, to select the best configuration.

Acknowledgements

This research is funded by the "Progetto Casa delle Tecnologie Emergenti" - Comune di Bologna - PSC MISE 2014-2020.

References

[1] K. S. Awaisi, Q. Ye and S. Sampalli, "A Survey of Industrial AIoT: Opportunities, Challenges, and Directions," in IEEE Access, vol. 12, pp. 96946-96996, 2024. https://doi.org/10.1109/ACCESS.2024.3426279

[2] S. Ankalaki, "Simple to Complex, Single to Concurrent Sensor-Based Human Activity Recognition: Perception and Open Challenges," in *IEEE Access*, vol. 12, pp. 93450-93486, 2024. https://doi.org/10.1109/ACCESS.2024.3422831

[3] H. Brendan McMahan, E. Moore, D. Ramage, S. Hampson, B. Agüera y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data", arXvi, https://arxiv.org/abs/1602.05629

[4] J. Cui, H. Zhu, H. Deng, Z. Chen, and D. Liu, "Fearh: Federated machine learning with anonymous random hybridization on electronic medical records," Journal of Biomedical Informatics, vol. 117, p. 103735, 2021. https://doi.org/10.1016/j.jbi.2021.103735

[5] M. Moshawrab, M. Adda, A. Bouzouane, H. Ibrahim, and A. Raad, "Reviewing federated learning aggregation algorithms; strategies, contributions, limitations and future perspectives," Electronics, vol. 12, no. 10, p. 2287, May 2023. https://doi.org/10.3390/electronics12102287

[6] H. Lee and D. Seo, "FedLC: Optimizing Federated Learning in Non-IID Data via Label-Wise Clustering," in *IEEE Access*, vol. 11, pp. 42082-42095, 2023. https://doi.org/10.1109/ACCESS.2023.3271517

[7] D. Ibdah, N. Lachtar, S. M. Raparthi and A. Bacha, "Why Should I Read the Privacy Policy, I Just Need the Service": A Study on Attitudes and Perceptions Toward Privacy Policies," in IEEE Access, vol. 9, pp. 166465-166487, 2021. https://doi.org/10.1109/ACCESS.2021.3130086

[8] Anguita, Davide, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. "A public domain dataset for human activity recognition using smartphones." In Esann, vol. 3, p. 3. 2013. https://www.esann.org/sites/default/files/proceedings/legacy/es2013-84.pdf

[9] Caldas, Sebastian, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. "Leaf: A benchmark for federated settings." arXiv preprint arXiv:1812.01097. https://arxiv.org/abs/1812.01097

3

Edge Intelligence Architecture for Distributed and Federated Learning Systems

Pierluigi Dell'Acqua, Lorenzo Carnevale, and Massimo Villari

University of Messina, Italy

Abstract

In recent years, deep neural networks have achieved success in Electric Vehicles (EVs) monitoring, primarily due to their scalability with large-scale data and numerous model parameters. However, EVs rely on resource-constrained edge devices that struggle with complex models, and data privacy concerns prevent sharing data outside the owning device. Federated Learning (FL) and Knowledge Distillation (KD) have emerged as key solutions, enabling model simplification and distributed training on private data. FL allows models to be trained locally on edge devices, addressing privacy concerns while keeping data decentralized and avoiding central server dependencies. This approach requires lightweight models optimized for edge intelligence deployment. To address this challenge, we propose architectural solutions leveraging FL, KD and model compression techniques to create simplified Artificial Neural Networks (ANNs) suitable for edge devices in EVs. The proposed architecture integrates these methods into a federated environment, ensuring distributed training while maintaining computational efficiency for EV monitoring and predictive maintenance applications. By combining FL, KD, and model compression, our approach enables efficient and privacy-preserving Machine Learning (ML) models, enhancing Edge Intelligence (EI) for EV monitoring in resource-constrained settings.

DOI: 10.1201/9788743808862-3

Keywords: knowledge distillation, federated learning, edge computing, IoT, edge intelligence.

3.1 Introduction

The automotive industry has witnessed a substantial expansion in both the scale and intricacy of electrical and electronic system architectures. In this regard, the EVs production is becoming increasingly prevalent in the field. Therefore, the challenge of predicting diagnosing faults and improving the LI-ion Battery (LIB) lifetime in EVs becomes progressively more demanding.

Modern monitoring systems approach the battery state parameters maintenance, such as the State of Charge (SoC), State of Health (SoH), State of Power, Remaining Useful Life, within safe limits, safeguarding the battery's safety. These are based on ML and Artificial Intelligence (AI) models, which can adapt the analysis to the specific technology. Furthermore, the computational trend is moving the data elaboration to the edge, with no requiring EVs producers to share their own diagnostic data as training datasets, preserving the industrial secrets and users' privacy. Actually, edge devices are characterized by low capacity and low performance that generally do not allow complex operations. For this reason, model simplification becomes necessary.

This scenario can be intended as a specific EI scenario, where AI models and algorithms are deployed and executed on resource-constrained edge devices. EI refers to the integration of AI capabilities directly on edge devices, enabling real-time data processing, decision-making and autonomy at the edge of the network. In such contexts, the need to balance computational demands with limited hardware resources is critical. Therefore, FL strategies, that aim to train models in a distributed manner and keep data locally on users' devices to preserve the privacy, can be adopted. The basic idea of FL unfolds in several key stages: i) a model based on an ANN is centrally initialized and subsequently disseminated to various peripheral devices; ii) these devices independently train the model using their locally available data, sending back for aggregation [17] to the central server only the outcomes of this localized training, such as the model weights.

This work aims to define a comprehensive and unified EI architecture designed to the specific demands of EVs monitoring systems. The goal is to leverage the potential of edge computing and distributed AI strategies to improve the real-time diagnosis, prediction, and overall health management, specifically of LIBs. Within this framework, the FL strategy is a cornerstone, providing an effective solution for decentralized learning. By enabling model training on local devices FL ensures that sensitive diagnostic data remains on the edge, thereby enhancing both privacy and security by preserving industrial know-how and user-specific data.

Moreover, to meet the computational efficiency requirements inherent to resource-limited edge devices, the architecture also incorporates complementary compression methods. Among these, KD not only facilitates the transfer of knowledge from a larger, more complex "teacher" model to a smaller, more efficient "student" model—making it ideal for deployment on devices with limited computational power—but also mitigates a well-known limitation of FL: KD strategies can effectively address both heterogeneous data and heterogeneous models within an FL environment [18].

In summary, this architecture proposes the integration of FL, KD, and compression techniques to create a scalable, efficient, and privacy-preserving solution for enhancing the monitoring and lifetime management of EVs within an EI scenario.

The remaining of the paper is organized as follow. Section 1.2 reports the EI background and discuss the recent advance in the literature. Section 1.3 determines the scenario where the proposed architecture may be deployed. That architecture is, therefore, discussed in Section 1.4. Finally, Section 1.5 summarizes the paper's insights and brings light on the future research activities.

3.2 Related Works

3.2.1 Edge Intelligence

In most of the cases, the computation on edge devices leverages data owned by them-self, data not shared with others for privacy reasons. In this context, EI leverages data generated at the edge of the network by applying AI directly to it. As stated in [28], there is not a formal definition of EI, though it is commonly used to describe the execution of AI models on edge devices. Therefore, they define EI as the efficient utilization of data in a cooperative edge-cloud system, where both inference and training can occur across all devices. This prospective is outlined by a six-level framework that categorizes where applications are executed, as reported in figure 3.1:

- Cloud Intelligence: a model (e.g., Deep Neural Network) is fully trained and executed in the cloud.
- Level 1 — Cloud–Edge Coinference and Cloud Training: a Deep Neural Network (DNN) model is trained in the cloud, but inference is performed through a cooperation between the cloud and the edge. In this case, part of the data is offloaded (e.g., migrated) to the cloud.
- Level 2 — In-Edge Coinference and Cloud Training: the DNN model is trained in the cloud, but inference is carried out at the network edge. In this setup, inference is handled by edge nodes or nearby devices, with data either fully or partially offloaded to the edge using device-to- device (D2D) communication.
- Level 3 — On-Device Inference and Cloud Training: the DNN model is trained in the cloud, but inference is performed entirely on the device itself, with no data being offloaded.

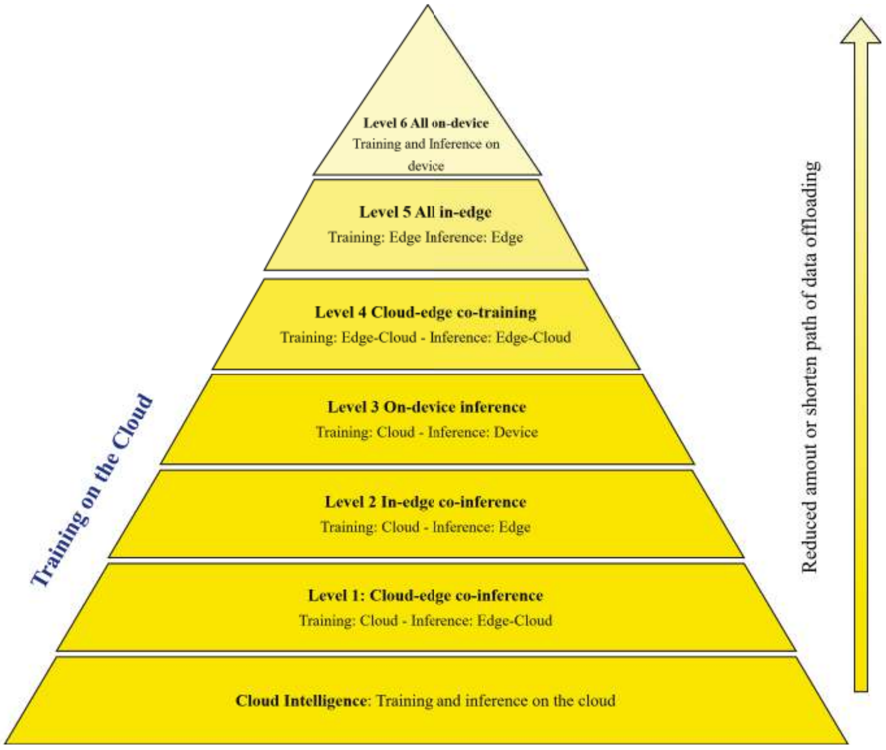

Level 6 All on-device
Training and Inference on device
Level 5 All in-edge
Training: Edge Inference: Edge
Level 4 Cloud-edge co-training
Training: Edge-Cloud - Inference: Edge-Cloud
Level 3 On-device inference
Training: Cloud - Inference: Device
Level 2 In-edge co-inference
Training: Cloud - Inference: Edge
Level 1: Cloud-edge co-inference
Training: Cloud - Inference: Edge-Cloud
Cloud Intelligence: Training and inference on the cloud
Training on the Cloud
Reduced amout or shorten path of data offloading

Figure 3.1 Six-level rating for EI described in [28]. ⏎

- Level 4 — Cloud–Edge Cotraining and Inference: Both the training and inference of the DNN model are con- ducted in a cloud-edge cooperative manner.
- Level 5 — All In-Edge: Both training and inference of the DNN model occur at the network edge.
- Level 6 — All On-Device: Both training and inference are handled entirely on the device.

As the level of EI increases, the amount and distance of data offloading decrease. This leads to lower data transmission latency, improved privacy and reduced Wide Area Network (WAN) bandwidth costs. However, this comes at the expense of increased computational latency and higher energy consumption. Therefore, there is not universally "best" level of EI. The optimal level is application-dependent and should be determined by considering multiple factors, such as latency, energy efficiency, privacy and WAN bandwidth cost.

3.2.2 Federated Learning

Among the different technologies designed to implement training in edge devices (e.g., Aggregation Frequency Control, Gradient Compression, Gossip Training), FL is often adopted where the learning process involves a federation of devices or nodes that do not need to share their own data but only their optimized model's parameters.

FL progresses through key stages, as reported in figure 3.2. Firstly, a neural network model is initialized in the central server and distributed to peripheral clients. Therein, the model is trained with local data only, sampled by the device itself. Importantly, this training happens autonomously, without sharing raw data with a central server. Only the outcomes of the training, which are the optimized model weights, are sent back to the central server, which aggregates them using an aggregation strategy (e.g., Federated Average [17], others [20] that implements the adaptive FedAvg, Lazy and Quantizatized gradients [4], [22]). This process maintains data privacy as raw data stays on local devices. This iterative process allows the model to evolve and improve over time without necessitating the aggregation of raw data in a central repository (Figure 3.2). The DFedAvgM framework [23] operates without a central server aggregator. It's implemented on clients connected through an undirected graph, where each client performs stochastic gradient descent with momentum and communicates only with its neighbors.

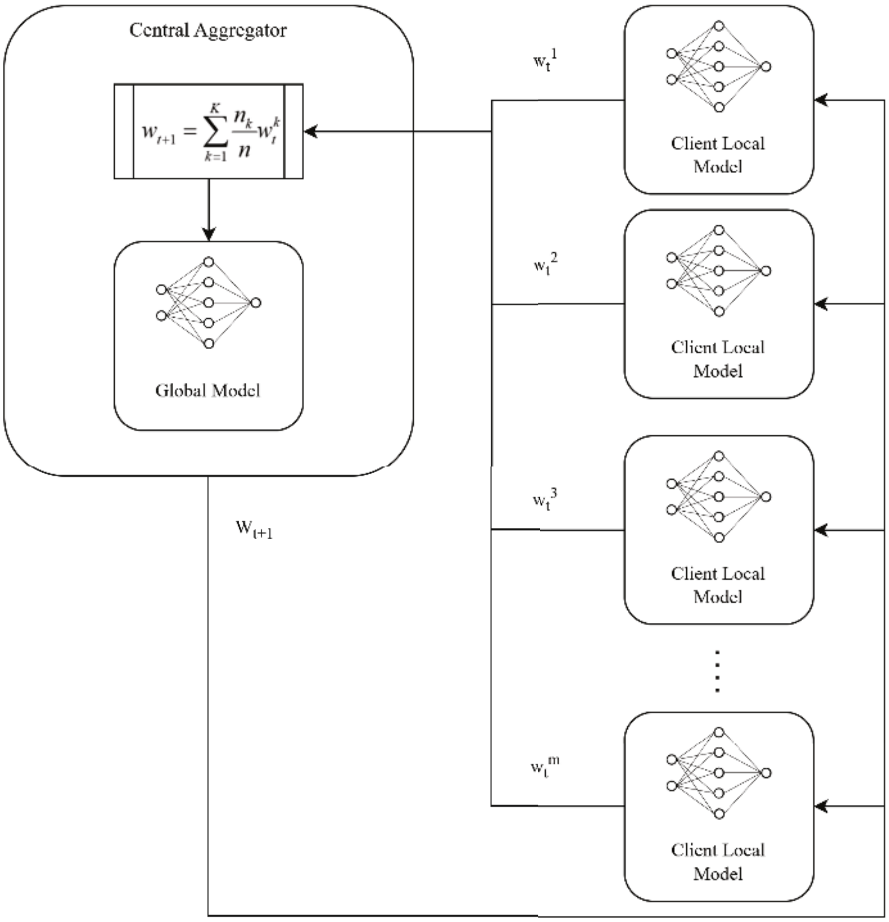


Figure 3.2 In a Federated Learning scenario, each client trains its model leveraging its own private data and sends its model parameters to a central server. The central server aggregates the parameters received from each client to enhance the performance of the central global model, which is then sent back to the clients. ⏎

This reduces communication costs and enhances privacy protection. The authors introduce DFedAvgM and its quantized algorithms offering extensive numerical verification of its performance.

3.2.3 Model Compression

In most of the cases, peripheral resource-constraint devices, with low computational capabilities, are not able to handle complex AI models because they cannot guaranty acceptable performance in terms of energy consumption, memory footprint and latency, i.e., in inference processes.

The main idea behind quantization is to convert the weights and activation values of a neural network model from high precision to lower precision, thereby reducing memory usage and latency [15]. Typically, quantization is applied to a pre-trained neural network, a process known as post-training quantization, without any fine-tuning [3], [6].

As an alternative, the Pruning methods [8] aim to remove non-essential components from DNN models while minimizing the impact on performance. Over time, pruning techniques evolved into two categories: i) the structured pruning as channels that serve as the primary pruning units; while ii) the unstructured pruning employs heuristic techniques to eliminate insignificant parameters, such as low-weight values, gradients or Hessian statistics [15].

Authors in [28] categorize EI technologies into training and inference technologies, as shown in Table 3.1. We have enhanced this table by incorporating KD, which serves as a strategy that bridges the two categories. While KD is implemented during the training phase, its primary goal is to produce a compressed model. The basic concept of KD [10] lies in training a simpler model (e.g., the student model) to imitate the behavior of the original, larger model (e.g., the teacher model). It produces a more efficient and quicker model to be executed. The teacher-student model is reported in figure 3.3.

In [7], different orthogonal distillation techniques classifications have been proposed. For example, the KD schema differentiates offline from online. Methods falling in the first category mainly focus on improving different parts of the knowledge transfer, including the design of knowledge and the loss functions. The [25] defines the distilled knowledge as the flow in the problem resolution trajectory, [13] uses Singular Value Decomposition (SVD) and Radial Basis Functions (RBf) for accuracy enhancement and minimizing computational costs, [19] incorporates distillation loss into the training of a smaller student network whose weights are quantized. On the other hand, the online distillation strategies aim to train both teacher and student models in a unified training scheme or, in vary common scenarios, to deal with lack of teacher network. The [26] proposes a collaborative learning strategy, [12] builds a multi-branch variant of a specified network by adding auxiliary branches, [24] leverages an additional classifier facilitating collaborative learning [21] and mutual learning without the need for pre-training a high- capacity teacher model. In many real-world applications,

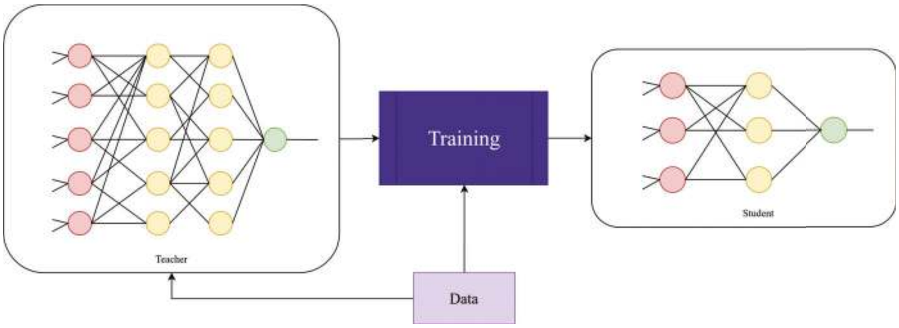


Figure 3.3 The schema illustrates the fundamental concept of KD: during the training of a simplified neural network, knowledge from a larger network is transferred to the smaller one. ⏎

Table 3.1 From Train to Inference technologies. ⏎

Technology	**Highlights**
Federated Learning	Training data remains on individual devices, while a shared model is trained centrally by aggregating locally computed updates, ensuring data privacy
Aggregation Frequency Control	Optimize the balance between local updates and global parameter aggregation, all within the given resource constraints
Gradient compression	Involves converting each element of the gradient vectors to a low-precision, finite-bit value. Gradient sparsification, on the other hand, reduces communication overhead by transmitting only a subset of the gradient vector values.
DNN Splitting	Select a splitting point to reduce latency as much as possible
Knowledge Distillation	Transfer the learnings of a large pre-trained model, the “teacher model,” to a smaller “student model”
Model Compression	Pruning and quantization methodologies
Model Partition	Computation tasks are offloaded to edge servers or mobile devices, with a focus on optimizing both latency and energy efficiency for better performance
Model Early-Exit	Accuracy-aware
Edge Caching	Fast response towards reusing the previous results of the same task
Input Filtering	Detecting difference between inputs, avoiding abundant computation
Model Selection	Inputs-oriented optimization and accuracy-aware
Support for Multi-Tenancy	Scheduling multiple DNN-based task

FL meets limitations and several KD frameworks are designed to overcome them. For example, the [9] improves communication efficiency reducing communication among clients, [5] adopts a grouping strategy to group clients that share homogeneous resources improving communication efficiency and balancing computing resources, [1], [2], [16] focus mainly on handling model and data heterogeneity.

3.2.4 Beyond the State of the Art

Actually, the scientific literature is challenging proposing a standard architecture for EI. Many methodologies may be involved both for training and inference. FL, KD, quantization and pruning are examples of enabling key technologies available for the exploitation of AI models into the edge of the network. This work aims to propose an architecture that includes all the mentioned methodologies on a simple workflow that optimizes the deployment and execution of EI solutions for EV.

3.3 Use Case

The use case focuses on the development of health monitoring systems for LIBs lifetime and contextual risk assessment in EVs. Among the various monitoring strategies, ML-based methods provide high accuracy, although they require large datasets for effective training [14]. Due to the complexity and critical nature of managing LIBs in EVs, deploying ML systems on resource-constrained microcontroller-based platforms is crucial. This integration enables real-time monitoring and predictive maintenance, optimizing battery performance and extending operational life.

Although cloud computing offers incomparable performance that can be leveraged for centralized activities (e.g., initial training, FL central aggregation), three main reasons lead to the need to push AI computation to the edge:

- *Sensing data*: the sensing layer within EVs produces a continuous and rich flow of data that cannot be transferred to the cloud to prevent bandwidth saturation.
- *Real-time response*: in a monitoring context, it is desirable to have real-time alerts rather than waiting for a stable connection with the central processor.

- *Privacy concerns*: manufacturers are not inclined to share their own diagnostic data with cloud data centers where big data are collected and processed.

The basic flow begins with defining and training an AI model in the cloud, potentially utilizing a more complex model in a KD scenario [10]. Once trained, the model may undergo optimization techniques such as quantization or pruning to reduce its size and improve inference time. After these optimizations, the model is converted into formats compatible with microcontroller-based devices (e.g., ONNX, TinyML), enabling deployment on distributed clients within EVs.

Once each client runs its own optimized AI model, real- time monitoring takes place locally on the EV. However, as new data continuously flows from the vehicle's sensing layer, the model can be further refined. These improvements can be shared with the central cloud, as well as with other clients, when stable connection is available, in a FL environment. This decentralized learning process allows each client to contribute to the overall model improvement without the need to share raw data, thus preserving privacy while enabling continual learning and adaptation (Figure 3.4).

This use case can be classified as level 4 within the EI framework proposed by [28], where both cloud and edge devices collaborate to perform training and inference tasks. However, in scenarios where the cloud is unable to handle training (e.g., due to the lack of a training dataset), the use case

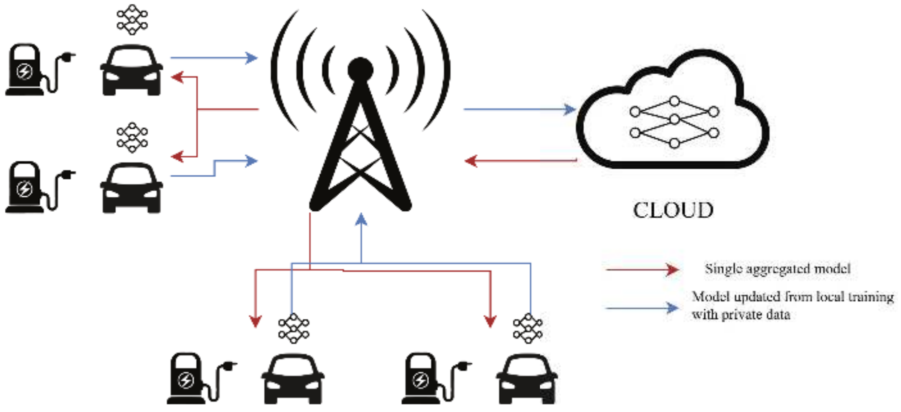


Figure 3.4 Use case scenario. ⏎

shifts to a level 5, where the edge devices will be engaged for both training and inference.

3.4 Architecture Proposal

In the context of EVs monitoring, leveraging data-driven approaches based on ML and AI algorithms is a critical point because of the limited computational and energy capacity of the machines. In the following sections, we will build the final architecture step-by-step, describing each involved component.

3.4.1 Assumption

The proposed architecture shares computation responsibility between cloud and edge computing, exploiting EI method- ologies for training and inference, such as the FL, KD, quantization and pruning. The cloud is adopted as much as possible for high-computation activities, such as serving as i) complex AI model training with existing datasets and ii) FL central node aggregator. Most of the inference and a relevant part of the training is intended to be executed on the edge.

The EVs are equipped with low-performance edge devices capable of handling models that are generally not too complex. These devices will perform monitoring and diagnostic tasks on their own sensing data, thereby preserving user privacy constraints. Moreover, since clients do not maintain a continuous connection either with the central cloud or among themselves, communication efficiency should be ensured. Clients will be responsible for performing training and sharing their model parameters to contribute to global model improvements.

3.4.2 Cluster Aggregator

The Cluster Aggregator, depicted in figure 3.5, is deployed in the cloud and primarily functions as the central aggregator for the FL system. This role is facilitated by several key modules responsible for specific tasks to ensure smooth and efficient operations within the federated framework.

1) *Model Aggregation Module*: This module is going to execute the FL core function. It generates the global model by aggregating (e.g., Federated Average [17]) trained models from the peripheral devices. This process ensures that the central model continuously improves by integrating

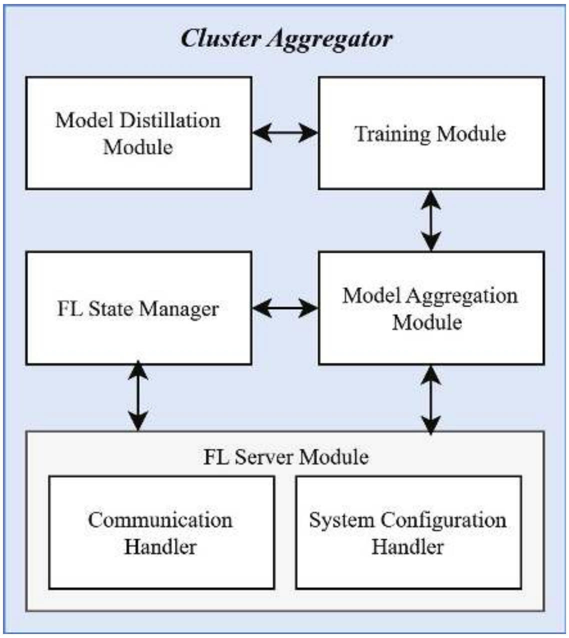


Figure 3.5 Cluster Aggregator schema designed to handle FL central aggregator tasks and to implement a distillation framework adaptable during the training process. ⏎

knowledge from distributed nodes without directly accessing their raw data, preserving privacy. This module is designed to handle different forms of model updates and can incorporate advanced techniques, such as weighted averaging, depending on the specific characteristics of the distributed models.

2) *FL State Manager*: It plays a critical role in maintaining synchronization between the cloud-based aggregator and peripheral devices. It tracks the status of each client, ensuring that the aggregation process considers only those clients that have successfully completed their local model training. The state manager keeps a record of which devices are actively participating in each round of FL, their connectivity status, and whether their contributions are valid for aggregation. This component also manages potential failures or delays in communication, ensuring the system can handle interruptions and continue functioning smoothly. Its role becomes even more significant in EVs scenario, where each EV changes its position very frequently and a stable connection cannot be guaranteed.

3) *System Configuration Handler*: The handler is tasked with managing the configuration of the entire system. This module ensures that the software environment is correctly set up, with all dependencies and configurations aligned for optimal performance. Additionally, it handles dynamic updates to system settings, such as changing communication protocols or modifying the aggregation frequency. Moreover, it guarantees that all components are correctly initialized and maintained throughout the lifecycle of the FL process.
4) *Communication Handler*: It manages the communication between the cloud-based aggregator and peripheral de- vices. It sets up and oversees the data transfer channels, ensuring that the communication is both efficient and secure. Given the distributed nature of FL, reliable communication is crucial for transmitting model updates, hyperparameters, and any other necessary metadata between clients and the cloud. The Communication Handler also implements protocols to minimize latency, reduce communication overhead, and ensure data integrity during transfer.
5) *Training and Model Distillation Modules*: The Training Module on the cloud side is activated when the global model is initialized and trained using an existing dataset. Once this initial training phase is completed, the global model is shared with the peripheral clients to begin the federated learning process. The clients use the global model as a starting point, performing local training on their own data and subsequently sharing their model updates with the central aggregator.

In addition to the standard training workflow, the Model Distillation Module plays a crucial role by implementing one or more KD strategies [7], [10], [19] during the training process. These strategies can be employed for several reasons:

- When the global model does not achieve an acceptable level of performance, KD can be used to refine it further by leveraging smaller, more efficient models that capture the key patterns of the original data
- Mainly in regression problem, different KD strategies make up for the lack of the dataset used for training
- [11], [27], allowing the transfer of knowledge from the pre-trained model to the global model without needing access to the original data

- When the FL process is subject to constraints, such as heterogeneous models and/or non-IID data across clients, KD can help align the learning processes [18].

By integrating KD into the training pipeline, the system can enhance the robustness and flexibility of the FL process, improving model performance in scenarios where traditional FL might face limitations.

All Together, the modules described above form a robust infrastructure that supports efficient and secure FL and their combination enables distributed training and aggregation while preserving user privacy, ensuring system reliability and maintaining overall system integrity.

3.4.3 Cloud Components

Although the Cluster Aggregator is the main component deployed on the Cloud, other elements need to be integrated to simplify model training and ANN-based model simplification, making them deployable on resource-constrained devices embedded in EVs (Figure 3.6).

1) *Compression Server:* It is introduced to reduce model complexity and size, addressing the challenges posed by the low computational performance and limited storage capacity of edge devices. It utilizes various handlers (i.e., software components capable of managing specific functionalities) to apply common compression techniques such as quantization, pruning and sparsification. These techniques are crucial for optimizing models to efficiently run on edge devices with constrained resources.

   Quantization reduces the precision of the numerical values used to represent the model's parameters, thereby decreasing both the model's size and its computational requirements. This allows edge devices to process models more efficiently without compromising significant accuracy.

   Pruning involves removing redundant or non-contributing weights from the model, which not only reduces its complexity but can also enhance performance by simplifying the model's structure. This results in a leaner, faster model that is more suitable for deployment in resource-limited environments.

   Sparsification, on the other hand, introduces sparsity into the model by setting insignificant weights to zero. This spar- sity can be leveraged by specialized hardware to accelerate computations, further enhancing the model's performance on edge devices.

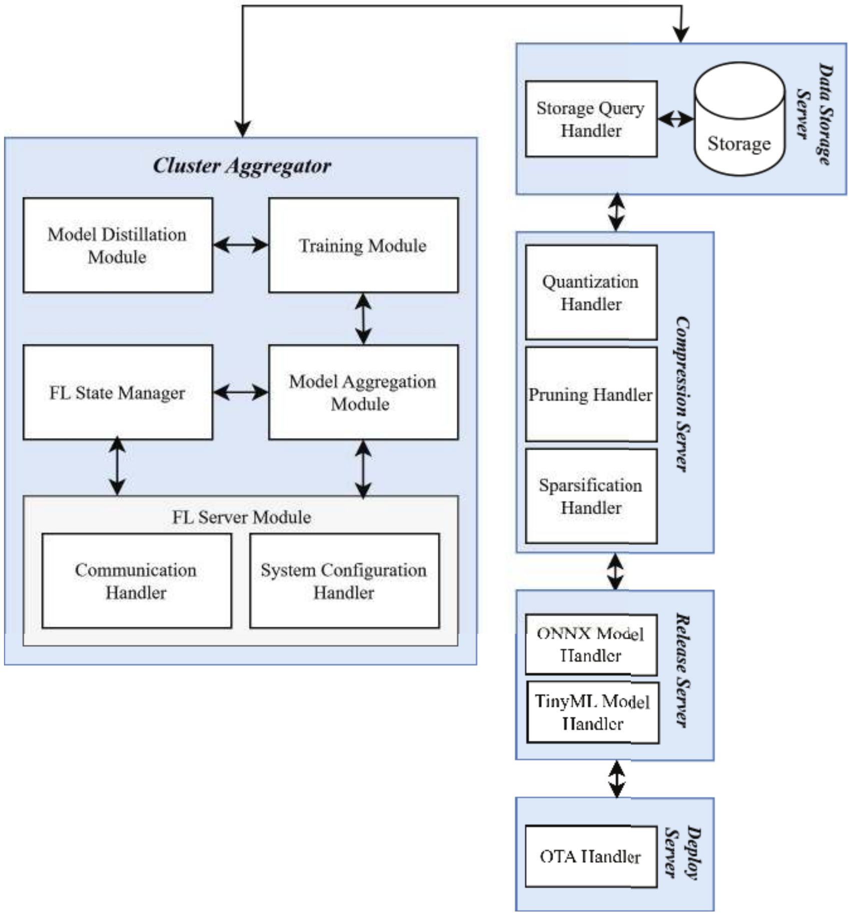


Figure 3.6 Software components deployed in the Cloud. ⏎

Together, these compression techniques enable the deploy- ment of complex models on edge devices, ensuring efficient operation while maintaining the balance between performance and resource utilization.

2) *Release Server*: The server is responsible for preparing ANN-based models in specific formats, such as ONNX, to enable seamless integration and deployment across a wide range of platforms and devices. The ONNX format, in particular, is highly valued for its interoperability between different machine learning frameworks, allowing models to be trained in one framework and deployed in another with minimal

conversion effort. This flexibility is crucial in environments where multiple frameworks are in use, ensuring that models can be efficiently transferred and utilized without compatibility issues. Additionally, the Release Server plays a critical role in the context of TinyML, where models must be optimized for execution on ultra-low-power devices such as microcontrollers and sensors. In this scenario, the server supports the conversion of models into highly compact formats suitable for deployment on resource-constrained edge devices. By integrating model compression techniques and optimizing for reduced memory and power consumption, the Release Server ensures that even complex ANN-based models can run efficiently in embedded systems.

3) *Deploy Server*: AI models will be deployed leveraging the Over-the-air (OTA) protocol that allows remote updates on microcontrollers without requiring physical access or direct connections. This approach is particularly beneficial for IoT and embedded systems where devices are often distributed in locations that are difficult to reach.

   The OTA deployment process begins with a central server preparing the new firmware or software update. The micro- controller periodically checks for updates via a secure wireless communication channel, such as Wi-Fi or cellular networks. When an update is available, the microcontroller downloads the update package and performs integrity checks. If the update passes the verification process, it is stored in a dedicated memory partition on the device. Finally, the microcontroller reboots and switches to the new firmware version, ensuring minimal downtime and continuous operation.

   Security plays a critical role in the OTA update process. To prevent unauthorized or malicious updates, encryption methods, authentication protocols, and secure boot mechanisms are often employed to ensure the integrity and authenticity of the update process.

4) *Data Storage Server*: Storage space provides crucial functionality. As its name implies, this server is responsible for storing large datasets required for the training process. It ensures that data is readily accessible and managed efficiently, supporting the extensive data requirements of modern machine learning algorithms. The Data Storage Server also handles data preprocessing and augmentation tasks, preparing the data in a suitable format for training.

These integrated components work in tandem to create a robust and efficient pipeline for deploying ANN-based models on resource-constrained devices embedded in EVs. By addressing the challenges of model size, complexity, and data management, the system ensures that high-performance models can operate effectively even in environments with limited computational resources.

3.4.4 Distributed Agent

The FL framework is intrinsically a distributed framework. In the proposed architecture (figure 3.7), a Distributed Agent is hosted by each peripheral client, which resides on an edge device within an EV.

1) *FL Client Module*: Within the agent, this module is responsible for establishing communication with the central cluster, receiving the global model's weights, and sending back the updated weights after performing local training on its local dataset. This module plays a crucial role in the federated learning process, ensuring that the client's contributions are incorporated into the global model. It is important to emphasize the complexity of the task managed by the Communication Handler. This component must work in coordination with the FL Participation Handler to address the asynchronicity of communication. Due to the intermittent nature of connectivity between each EV and the central aggregator, as well as among the EVs themselves, there is no guarantee of continuous communication. This sporadic connectivity necessitates robust mechanisms to ensure that updates are transmitted accurately and efficiently whenever a connection becomes available, thereby maintaining the integrity and effectiveness of the federated learning process.
2) *Inference Module*: The local ANN-based model will be utilized in inference tasks to implement the monitoring process. This module takes the trained model and applies it to real-time data gathered from the EV, enabling functions such as predictive maintenance, performance optimization, and anomaly detection. By leveraging the local model, the EV can make intelligent decisions without relying on constant cloud connectivity, thus enhancing the system's reliability and responsiveness. Additionally, the architecture ensures data privacy and security, as the FL approach allows data to remain on the edge device. Only the model updates, which are less sensitive than raw data, are shared with the central cluster. This decentralized approach not only enhances privacy but also reduces the bandwidth required for data transmission, which is critical in mobile and resource-constrained environments like EVs.

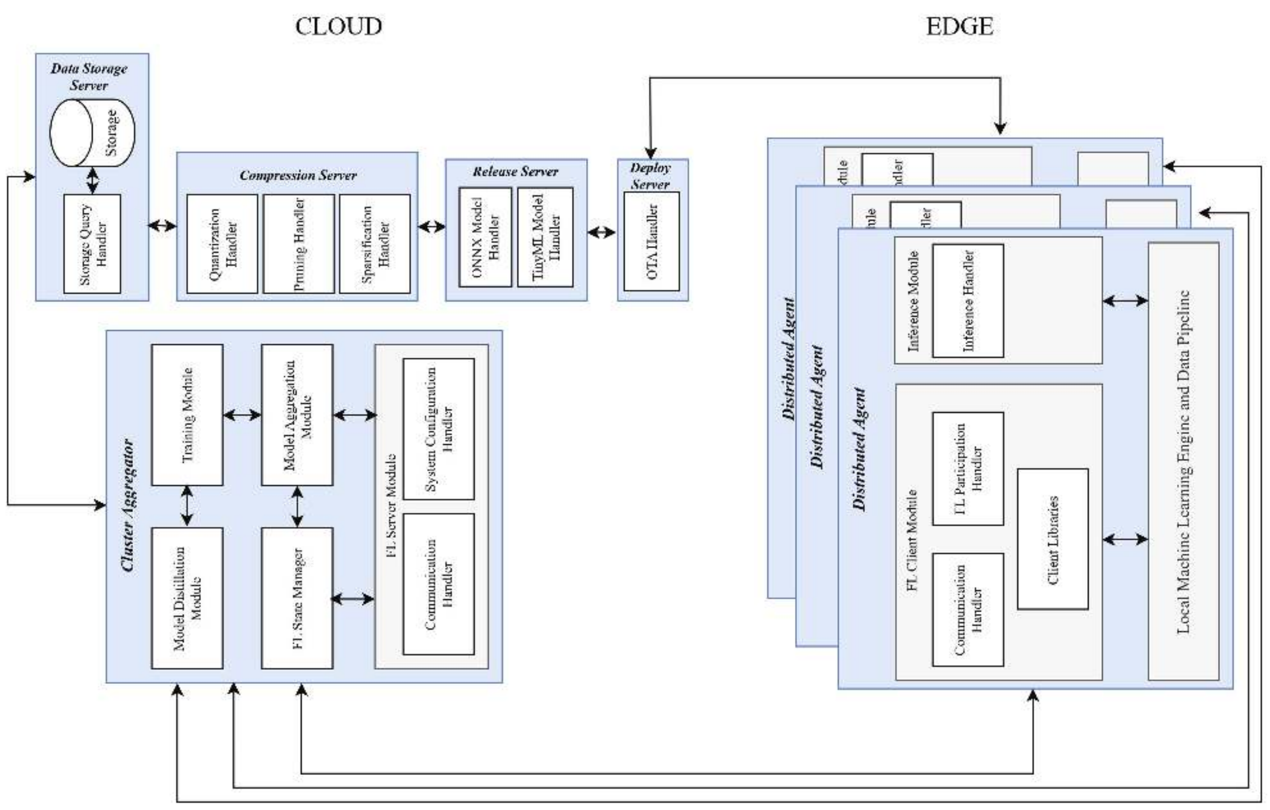


Figure 3.7 The final architecture includes a Cluster Aggregator, deployed in the Cloud, and Distributed Agents, deployed on resource-constrained edge devices. ⏎

Overall, the proposed architecture provides a scalable and efficient solution for deploying FL in EVs. By addressing communication challenges, ensuring data privacy, and enabling local inference, the system enhances the capabilities of EVs to perform complex tasks autonomously and effectively.

3.5 Challenges

The implementation of distributed artificial intelligence architecture presents several key challenges:

- *Communication among devices with limited Internet connectivity and energy capacity*: The intermittent availability of network connections, coupled with restricted energy resources and with the mobility of the clients, poses significant difficulties for effective data transmission and distributed computing and FL state maintenance.
- *Generalization of neural network architectures, AI frameworks*: A critical challenge lies in ensuring that AI solutions can generalize across diverse neural network models, software frameworks, and hardware platforms, enabling broad applicability and scalability.
- *Deployment on several architectures*: Deploying AI models on microcontrollers, each with different architectures and processing capabilities, adds a layer of complex- ity. Ensuring compatibility and performance optimization across these heterogeneous systems requires careful consideration of both software adaptation and hardware constraints.
- *Trade-offs in training and compression methodologies*: Striking the right balance between various training approaches and data compression techniques is essential to optimize performance while managing the limitations of resource-constrained devices

3.6 Conclusion

In this work, we designed and proposed an architecture aimed at implementing an Edge Intelligence scenario. Thanks to its modularity, each EI rating layer, from 1 to 5, can be realized.

Among different technologies, Federated Learning has been identified as a distributed training strategy to prevent data sharing and preserve privacy. Since edge devices typically have low-performance hardware, we also included modules for AI model compression and simplification.

Although the architecture has been designed for a generic EI scenario, we identified the monitoring system of Electric Vehicles as a potential use case where the proposed architecture could be applied after development. Furthermore, a list of interesting challenges and open points has been identified and is presented in this paper.

Acknowledgements

This work is partially supported by the Horizon Europe "Open-source deep learning platform dedicated to Embedded hardware and Europe" project (Grant Agreement, project 101112268 - NEUROKIT2E).

References

[1] S. Ahmad and A. Aral, "*FedCD: Personalized Federated Learning via Collaborative Distillation*," 2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC), Vancouver, WA, USA, 2022, pp. 189-194, https://doi.org/10.1109/UCC56403.2022.00036.

[2] Ilai Bistritz, Ariana J. Mann, and Nicholas Bambos. 2020. *Distributed distillation for on-device learning*. In Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS '20). Curran Associates Inc., Red Hook, NY, USA, Article 1894, 22593–22604.

[3] Y. Cai et al., "*ZeroQ: A Novel Zero Shot Quantization Framework*," in 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 2020, pp. 13166-13175, doi: 10.1109/CVPR42600.2020.01318.

[4] Tianyi Chen, Georgios B. Giannakis, Tao Sun, and Wotao Yin. 2018. *LAG: lazily aggregated gradient for communication-efficient distributed learning*. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Curran Associates Inc., Red Hook, NY, USA, 5055–5065.

[5] Z. Chen, P. Tian, W. Liao, X. Chen, G. Xu and W. Yu, "*Resource-Aware Knowledge Distillation for Federated Learning*," in IEEE Transactions on Emerging Topics in Computing, vol. 11, no. 3, pp. 706-719, 1 July-Sept. 2023, doi: 10.1109/TETC.2023.3252600.

[6] Jun Fang, Ali Shafiee, Hamzah Abdel-Aziz, David Thorsley, Georgios Georgiadis, and Joseph H. Hassoun. *Post-training piecewise linear quantization for deep neural networks*. page 69–86, Berlin, Heidelberg, 2020. Springer-Verlag.

[7] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. 2021. \. Int. J. Comput. Vision 129, 6 (Jun 2021), 1789–1819. https://doi.org/10.1007/s11263-021-01453-z.

[8] Stephen Joset' Hanson and Lorien Y. Pratt. *Comparing biases for minimal network construction with back-propagation*. NIPS'88, page 177–185, Cambridge, MA, USA, 1988. MIT Press.

[9] Chaoyang He, Murali Annavaram, and Salman Avestimehr. 2020. *Group knowledge transfer: federated learning of large CNNs at the edge*. In Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS '20). Curran Associates Inc., Red Hook, NY, USA, Article 1180, 14068–14080.

[10] Hinton, G.E., Vinyals, O., & Dean, J. (2015). *Distilling the Knowledge in a Neural Network. ArXiv, abs/1503.02531.*

[11] Myeonginn Kang and Seokho Kang. *Data-free knowledge distillation in neural networks for regression*. Expert Systems with Applications, 175:114813, 2021.

[12] Xu Lan, Xiatian Zhu, and Shaogang Gong. *Knowledge distillation by on-the-fly native ensemble*. NIPS'18, page 7528–7538, Red Hook, NY, USA, 6 2018. Curran Associates Inc.

[13] Seung Hyun Lee, Dae Ha Kim, and Byung Cheol Song. 2018. *Self-supervised Knowledge Distillation Using Singular Value Decomposition*. In Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8–14, 2018, Proceedings, Part VI. Springer-Verlag, Berlin, Heidelberg, 339–354. https://doi.org/10.1007/978-3-030-01231-1_21.

[14] Yi Li, Kailong Liu, Aoife M. Foley, Alana Zuĺlke, Maitane Berecibar, Elise Nanini-Maury, Joeri Van Mierlo, and Harry E. Hoster. *Data-driven health estimation and lifetime prediction of lithium-ion batteries: A review*. Renewable and Sustainable Energy Reviews, 113:109254, 2019.

[15] Li, Z.; Li, H.; Meng, L. *Model Compression for Deep Neural Networks: A Survey*. Computers 2023, 12, 60. https://doi.org/10.3390/computers12030060.

[16] Tao Lin, Lingjing Kong, Sebastian U. Stich, and Martin Jaggi. 2020. *Ensemble distillation for robust model fusion in federated learning*. In Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS '20). Curran Associates Inc., Red Hook, NY, USA, Article 198, 2351–2363.

[17] McMahan, H.B., Moore, E., Ramage, D., Hampson, S., & Arcas, B.A. (2016). *Communication-Efficient Learning of Deep Networks from Decentralized Data.* International Conference on Artificial Intelligence and Statistics.

[18] Alessio Mora, Irene Tenison, Paolo Bellavista, and Irina Rish. *Knowledge distillation in federated learning: A practical guide.* In Kate Larson, editor, Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24, pages 8188–8196. International Joint Conferences on Artificial Intelligence Organization, 8 2024. Survey Track.

[19] Polino, A., Pascanu, R., & Alistarh, D. (2018). Model compression via distillation and quantization. *ArXiv, abs/1802.05668.*

[20] Sashank Reddi et al. *Adaptive Federated Optimization.* arXiv e-prints, page arXiv:2003.00295, February 2020.

[21] Guocong Song and Wei Chai. 2018. *Collaborative learning for deep neural networks.* In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Curran Associates Inc., Red Hook, NY, USA, 1837–1846.

[22] Jun Sun, Tianyi Chen, Georgios B. Giannakis, and Zaiyue Yang. 2019. Communication-efficient distributed learning via lazily aggregated quantized gradients. Proceedings of the 33rd International Conference on Neural Information Processing Systems. Curran Associates Inc., Red Hook, NY, USA, Article 303, 3370–3380.

[23] T. Sun, D. Li and B. Wang, "*Decentralized Federated Averaging,*" in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 45, no. 4, pp. 4289-4301, 1 April 2023, https://doi.org/10.1109/TPAMI.2022.3196503.

[24] Wu, Guile & Gong, Shaogang. (2020). *Peer Collaborative Learning for Online Knowledge Distillation.* 10.48550/arXiv.2006.04147.

[25] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. *A gift from knowledge distillation: Fast optimization, network minimization and transfer learning*. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 7130–7138, 2017.

[26] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu. *Deep mutual learning*. In 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 4320–4328, Los Alamitos, CA, USA, jun 2018. IEEE Computer Society.

[27] Tianxun Zhou, Keng-Hwee Chiam. *Synthetic data generation method for data-free knowledge distillation in regression neural networks.*

Expert Systems with Applications, Volume 227, 2023, 120327, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2023.120327.

[28] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo and J. Zhang, "*Edge Intelligence: Paving the Last Mile of Artificial Intelligence with Edge Computing*," in Proceedings of the IEEE, vol. 107, no. 8, pp. 1738-1762, Aug. 2019, https://doi.org/10.1109/JPROC.2019.2918951.

4

Challenges and Performance of SLAM Algorithms on Resource-constrained Devices

Calvin Galagain[1, 2], Martyna Poreba[1], and François Goulette[2]

[1]Université Paris-Saclay, CEA-List, France
[2]ENSTA Paris, France

Abstract

Evaluating the performance of Simultaneous Localization and Mapping (SLAM) algorithms is essential for the progress of robotic systems. However, conducting a comprehensive assessment of SLAM systems in the context of recent advancements is challenging due to the wide variety of hardware platforms, algorithm configurations, and datasets available. This study aims to test SLAM algorithms on resource-constrained devices such as the NVIDIA Jetson AGX Orin 64GB. Experiments are conducted with various visual-based localization algorithms that either leverage deep learning models for specific tasks within the SLAM process or are learned end-to-end to estimate camera pose. The evaluation focuses on the following systems: RDS-SLAM and VDO-SLAM, which utilize semantic information to achieve precise motion estimation; TSformer-VO, an end-to-end Transformer-based model designed for monocular visual odometry; and DeepVO, which based on recurrent neural networks. The systems are evaluated using several metrics, including ATE and RPE to assess pose accuracy and rotational drift, respectively, alongside runtime, energy consumption, and resource usage to gauge their efficiency and practicality for real-world applications.

DOI: 10.1201/9788743808862-4

Keywords: SLAM, localization, benchmarking, computational efficiency, system efficiency, performance.

4.1 Introduction and Background

Simultaneous Localization and Mapping (SLAM) is a crucial technology that enables autonomous systems, such as robots, drones, and AR/VR devices, to navigate and understand their environments without relying on external reference systems. SLAM algorithms operate by simultaneously constructing a map of an unknown environment while tracking the system's position within it. They typically utilise a combination of sensors, including cameras, LiDAR, and inertial measurement units (IMUs), to gather data about the surrounding area. Integrating these sensors improves the accuracy and robustness of SLAM but also introduces a higher computational burden, posing a substantial challenge to achieving real-time performance. To maintain operational efficiency, optimizations such as simplifying algorithms, reducing the number of processed features, and leveraging parallel processing capabilities are essential. When deployed on embedded systems, SLAM encounters unique challenges due to resource constraints, including limited processing power, memory, and energy consumption. These limitations necessitate the development of highly efficient algorithms capable of performing complex tasks in real time, such as image processing, sensor fusion, loop closure detection, and optimization. Despite advances in the field, numerous challenges persist. Ensuring robustness against sensor noise, managing varying environmental conditions, and scaling algorithms to accommodate different map sizes and complexities are critical areas of ongoing research. Furthermore, the demand for lightweight implementations that do not compromise performance underscores the continuous evolution of SLAM technologies.

The primary aim of this paper is to benchmark specific SLAM methods on the NVIDIA Jetson AGX Orin 64GB, a robust embedded platform tailored for real-time processing in autonomous systems. Specifically, we perform a comprehensive performance analysis, comparing RDS-SLAM [1] and VDO-SLAM [2], two semantic SLAM techniques, against the Visual Odometry (VO) performance of CNN-based methods trained in an end-to-end manner. This benchmarking aims to assess the performance of these SLAM algorithms under constrained resource conditions, with a specific focus on two key aspects:

- **Performance Comparison:** Evaluating the ability of each algorithm to accurately localize in dynamic environments.
- **Resource Utilisation Assessment:** Analysing how each SLAM method leverages the resources of the Jetson AGX Orin, specifically regarding CPU and GPU performance, memory usage, and energy consumption

Finally, the study provides recommendations for optimizing SLAM algorithms on embedded platforms and identifies key areas for future research and development.

4.2 Related Work

SLAM has attracted considerable attention over the past few decades, leading to the development of numerous approaches [1-7]. A thorough review of the existing literature on SLAM methods reveals a diverse array of algorithms tailored for different applications and hardware platforms. For example, ORB-SLAM [3] has been widely recognized for its efficiency and robustness on conventional CPUs, demonstrating good performance across various environments. Similarly, VINS-Mono [4] and VINS-RGBD [5] are noted for their effectiveness in combining visual and inertial data. Recent studies have highlighted the potential of DNN-based SLAM systems, such as RDS-SLAM [1], VDO-SLAM [2], DF-SLAM [6], Dyna-SLAM [7], and DeepFactors [8], which leverage deep learning techniques to improve feature extraction, pose estimation, or environment understanding. End-to-end deep learning methods like DeepVO [9], TS-Former [10], and DROID-SLAM [11] also mark a significant shift in visual localization systems by using neural networks to directly learn the entire process from raw sensor data to pose estimation and map generation, bypassing traditional hand-crafted feature extraction and geometric modeling. However, these methods frequently encounter limitations compared to traditional SLAM techniques, such as lower accuracy and significant dependence on large training datasets. Neural Radiance Fields (NeRF)- based SLAM [12–16] offers a novel solution by incorporating NeRF models into SLAM systems to enhance the representation of 3D environments. In contrast to traditional SLAM methods that rely on discrete points or sparse features, NeRF-enhanced approaches produce continuous volumetric fields to create highly detailed and realistic 3D reconstructions. Through the use of neural networks, these methods can model entire scenes and facilitate photorealistic rendering by understanding the interactions of light with surfaces. However, NeRF-based SLAM methods face challenges on embedded systems due to their high computational requirements, energy consumption, complex integration, and limited generalisation, underscoring the need for more efficient solutions.

A growing number of embedded computing platforms now feature NPU/GPU units, enabling lightweight deep learning networks to function in real time. Many researchers have worked to modify SLAM algorithms for low-power embedded platforms, reengineering them to ensure compatibility with these devices. Despite these advancements in SLAM technology, implementing these algorithms on resource constrained platforms still faces considerable challenges, largely determined by the distinct characteristics of both the algorithms and the underlying embedded architectures. Although some researchers [17, 18] have explored VO systems resulting in a decreased accuracy, others [19–22] have successfully developed keyframe-based SLAM systems. Various approaches have been explored to optimize computation and power overhead in visual-inertial odometry (VIO), particularly through hardware acceleration using FPGAs [23–25]. Although advancements have been made, keyframe-based SLAM systems still face challenges in achieving an optimal balance between efficiency and accuracy for mobile robot applications. One of the most recent developments, Dynamic-VINS [26], an enhanced iteration of VINS-Mono and VINS-RGBD, showcases remarkable performance on resource-constrained platforms such as the HUAWEI Atlas200 DK and NVIDIA Jetson AGX Xavier. Finally, a hardware-software co-design approach is proposed to optimize latency, power consumption, and tracking speed in VIO systems by incorporating the Optical Flow (OF) estimation on the sensor [27]. In the VINS-Mono pipeline, feature tracking was substituted with an OF camera that employs an ASIC-based accelerator, while the other components of the VIO pipeline operate on the main processor of a Raspberry Pi Compute Module4.

Assessing the performance of SLAM algorithms is essential for both researchers and users of robotic systems. Comprehensive benchmarking enables in-depth evaluations, helping to identify the most effective SLAM algorithms, and laying the groundwork for future enhancements and innovations in the field. The wide variety of hardware configurations, algorithm settings, and datasets complicates thorough comparisons across the state-of-the-art. There is a notable scarcity of research focused on benchmarking SLAM algorithms on embedded systems, highlighting significant gaps in the existing literature that this study aims to address. For example, the research presented in [28] assesses power consumption, accuracy metrics, and processing frame rates for ORB-SLAM and OpenVSLAM [29] on NVIDIA Jetson embedded systems, particularly the Jetson Nano, Jetson TX2, and Jetson Xavier. The SLAM Hive Benchmarking Suite [30] has recently addressed this challenge by offering a scalable solution that leverages container technology and cloud deployment to analyze thousands of SLAM executions.

4.3 Methodology

For this comparison study, a diverse range of approaches have been selected, including semantic geometric SLAM methods as well as VO techniques derived from deep learning models trained in an end-to-end manner. This enables a comprehensive evaluation of various strategies to identify those most suitable for embedded systems, focusing on balancing performance and computational efficiency. The systems are assessed based on several metrics such as pose accuracy and rotational drift, along with an analysis of runtime, energy consumption, and GPU and CPU usage to determine their efficiency and suitability for real-world applications.

4.3.1 Selected systems

We examine the following systems: RDS-SLAM, which enhances the localization process with semantic segmentation; VDO-SLAM, which utilizes semantic information for accurate motion estimation and tracking of dynamic rigid objects; TSformer-VO, an end-to-end Transformer-based model for monocular visual odometry that learns motion estimation directly from raw images; and DeepVO, which leverages recurrent networks to capture temporal dependencies.

Unlike traditional SLAM algorithms, which assume a static scene, RDS-SLAM (see Figure 4.1) detects and excludes dynamic objects to enhance the robustness of tracking and mapping. The algorithm extends the base framework of ORB-SLAM3 by introducing two parallel threads: a semantic segmentation thread and a semantic-based optimization thread. These threads enable the segmentation of images into static and dynamic objects using methods such as Mask R-CNN [31] or SegNet [32], while optimizing the tracking data in real time without blocking the process. The semantic thread is executed selectively to keyframes, rather than every frame. The semantic information is then propagated across the global map, where each map

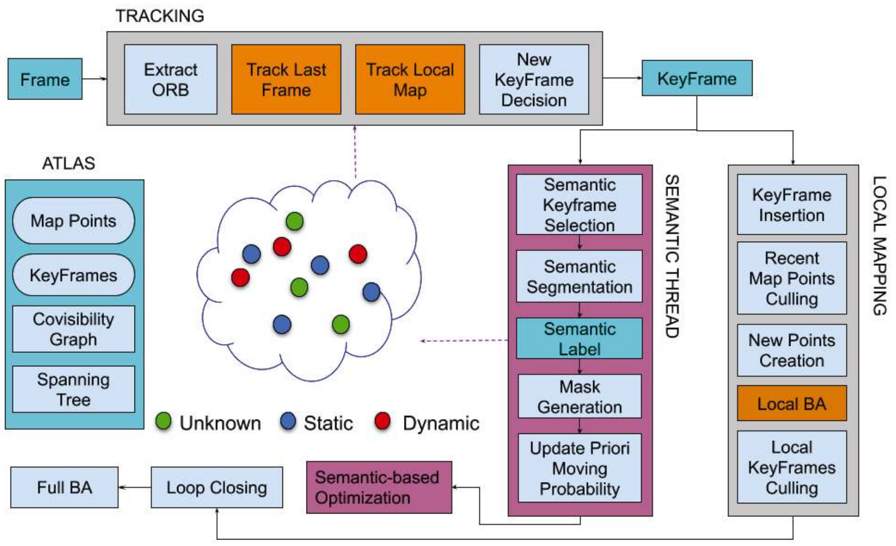


Figure 4.1 Overview of RDS-SLAM [1]. ⏎

point is assigned a moving probability. This probability is updated as new keyframes are processed and is used to classify map points as dynamic, static, or unknown. The algorithm identifies dynamic objects using segmentation masks from semantic models, assuming that classes such as people and vehicles are likely dynamic. Points classified as dynamic are excluded from the tracking process to avoid introducing errors in the camera pose estimation. In contrast, static points are used to improve the accuracy of the tracking. This approach allows RDS-SLAM to achieve precise tracking and robust mapping, even in environments with dynamic objects that typically pose challenges for traditional SLAM algorithms.

The VDO-SLAM system (Figure 4.2) is also designed to handle dynamic environments. Before the execution of the SLAM algorithm, two crucial pre-processing steps are applied. First, Mask R-CNN is used for instance-level semantic segmentation, which allows for the identification of both static and dynamic objects, such as vehicles and pedestrians, by generating object masks. Second, PWC-Net [33], a state-of-the-art optical flow network, is applied to estimate the dense pixel motion between consecutive frames. Using these data, VDO-SLAM can estimate the full SE(3) motion of dynamic objects, including both their linear velocity and rotational movement, while also refining its own camera pose. This integration allows

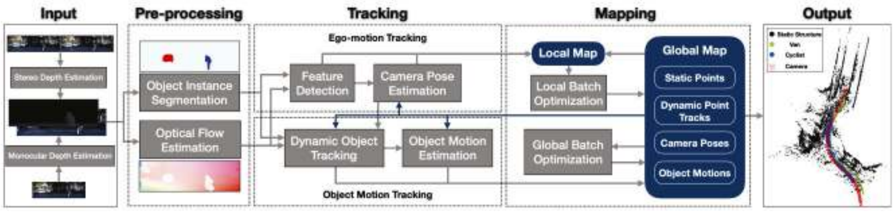


Figure 4.2 Overview of VDO-SLAM [2]. ⏎

for robust navigation in dynamic environments where traditional SLAM methods, which assume static surroundings, would fail.

Visual odometry, a key component of the tracking phase in SLAM, can also be accomplished using methods based on Convolutional Neural Networks (CNNs). The goal is to enable the system to directly learn motion estimation from raw image inputs, eliminating the need for traditional feature extraction and matching techniques. DeepVO combines Convolutional Neural Networks (CNNs) to extract visual features from images with Long Short-Term Memory networks (LSTMs) to capture and model temporal relationships between consecutive images, enabling the prediction of camera movements.

On the other hand, TSformer-VO takes a distinct approach by utilizing transformers to extract spatio-temporal features from video sequences. Unlike DeepVO, which relies on recurrent mechanisms, TSformer-VO utilizes spatio-temporal attention to capture interactions between images across both spatial and temporal dimensions. This allows for a more holistic understanding of the scene, leading to enhanced precision in estimating the camera's 6-DoF poses. By processing long-range dependencies within the video data, TSformer-VO reduces pose drift and improves robustness in dynamic environments. Additionally, the model's end-to-end learning framework enables it to adaptively optimize feature representations, making it a powerful alternative to traditional visual odometry methods.

4.3.2 Selected systems

For the benchmarking setup, we used the NVIDIA Jetson AGX Orin 64GB, a high-performance platform specifically designed for real-time AI and embedded applications. The Jetson AGX Orin features a 2048-core NVIDIA Ampere architecture GPU with 64 Tensor Cores and a 12-core ARM Cortex-A78AE CPU, running at 2.2 GHz. The system is equipped with 64GB of LPDDR5 memory and provides 275 TOPS (INT8) of AI performance.

The Jetson platform allows for efficient execution of SLAM algorithms, balancing high computational power and energy efficiency, making it ideal for real-time applications in resource-constrained embedded environments. On the software side, all algorithms were deployed using Docker to guarantee consistent and efficient testing across various approaches and datasets. This containerization enables performance and results to be compared under the same conditions, thus streamlining the testing process.

4.3.3 Evaluation metrics

The evaluation of SLAM algorithms is based on a set of well-defined metrics designed to assess both accuracy and computational efficiency. These metrics provide quantitative insight into the global alignment and local consistency of the estimated trajectories. Specifically, we utilize Absolute Trajectory Error (ATE) and Relative Pose Error (RPE) to measure the discrepancy between the estimated and ground truth trajectories, capturing both overall accuracy and the drift over time. Additionally, Frames Per Second (FPS) is employed to evaluate the real-time performance of the system. To further assess computational efficiency, we monitor system resource usage, including CPU and GPU utilization, memory footprint, and power consumption. Detailed definitions and the methodology for computing these metrics can be found in the Appendix.

4.3.4 Dataset

For evaluation, we used the TUM RGB-D dataset [34], a widely recognized benchmark for SLAM systems. This dataset provides various indoor sequences captured using RGB-D cameras, which include both static and dynamic scenes. These sequences are ideal for testing the performance of SLAM algorithms in diverse and challenging real-world scenarios. To ensure diversity in our evaluation, we selected a specific subset of sequences that represent a wide range of environments, motions, and dynamics. The chosen sequences are as follows:

- **freiburg1_desk:** This sequence captures normal movements within a static office environment, making it suitable for evaluating the performance of SLAM algorithms in controlled, steady indoor settings.
- **freiburg1_xyz:** In this sequence, the camera undergoes translations along all three axes (X, Y, Z) while the environment remains static.

It evaluates the algorithm's ability to handle structured translational movements.

- **freiburg2_xyz:** This sequence involves rapid translations in a static indoor setting, challenging SLAM systems with faster motions, and requiring precise tracking in environments with minimal changes.
- **freiburg2_rpi:** The camera performs quick rotations around the roll, pitch, and yaw axes within an indoor space. This sequence stresses the system's ability to manage sudden rotational movements.
- **freiburg3_long_office_household:** A longer sequence set in a domestic environment with various objects and changing lighting conditions. This provides a complex, real-world scenario with longer-term tracking requirements and varying conditions.
- **freiburg3_walking_xyz:** This sequence captures rapid movements with significant translations and human dynamics, simulating more unpredictable and dynamic real-world conditions.
- **freiburg3_walking_static:** Featuring fast movements within a static environment, this sequence tests the robustness of SLAM algorithms when confronted with high-speed camera motion while the scene remains unchanged.

4.4 Experimentation

The experiments were conducted to analyze the performance of the selected SLAM algorithms under realistic conditions, with a focus on their accuracy, efficiency, and suitability for embedded platforms.

4.4.1 Performance evaluation

Our study begins with a visual comparison of the accuracy in trajectory estimation across the selected localization systems, using sequences from the TUM RGB-D dataset. Figure 4.3 shows the trajectories for the *freiburg3_structure_texture_far*. For TSformer, three configurations are used, considering 1, 2, or 3 images to predict the camera motion.

Table 4.1 & Table 4.2 summarize the average performance on selected sequences. Inference for models, whether trained end-to-end or integrated as a semantic thread, is performed using PyTorch (FP32), without utilizing TensorRT for performance optimization on NVIDIA hardware. VDO-SLAM demonstrates a balanced approach, achieving the best performance among

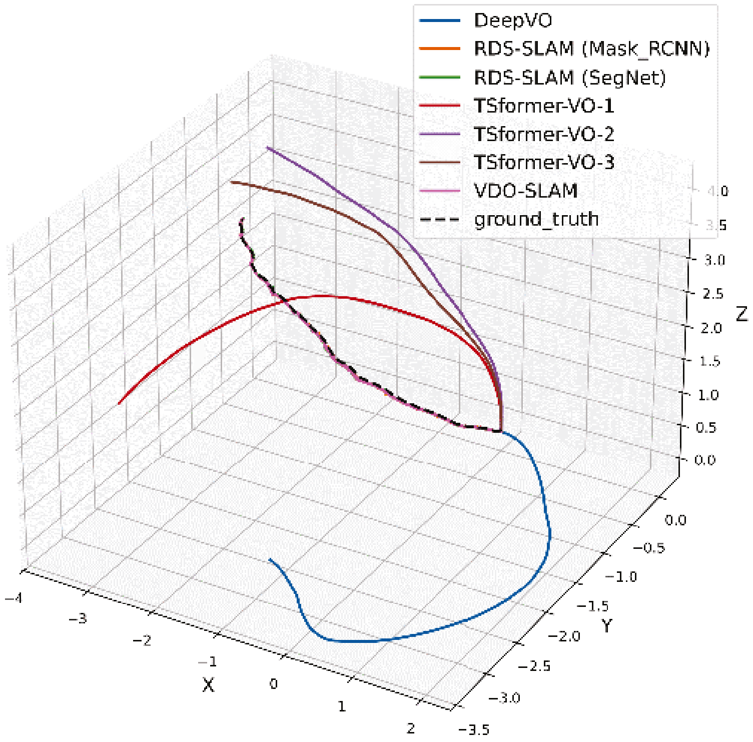


Figure 4.3 Trajectory predictions: each color denotes a different tested system. ⏎

Table 4.1 Performance Metrics: overall localization accuracy (ATE), Error between successive poses (RPE) and Inference Time ⏎

Methods	ATE (cm)	RPE (cm)	FPS
DeepVO	402.5	1.42	9.6
TSformer-VO-1	135.2	0.95	7.1
TSformer-VO-2	172.9	0.91	5.0
TSformer-VO-3	152.2	0.92	3.8
RDS SLAM (Mask RCNN)	3.4	0.96	3.2
RDS SLAM (SegNet)	3.3	1.00	7.5
VDO SLAM	3.4	1.00	8.1

the two SLAM systems tested, with an FPS of 8.1, reasonable energy consumption (∼10.85W), and a competitive ATE of 3.4 cm. However, a key factor driving VDO-SLAM's efficiency is that semantic segmentation is handled in a pre-processing step (0% in the SLAM pipeline itself). This approach allows the system to focus the bulk of its computational

Table 4.2 Resource Usage ⏎

Methods	CPU (%)	GPU (%)	RAM (%)	Power (mW)
DeepVO	20.99	46.45	15.93	14017.10
TSformer-VO-1	12.11	95.19	17.90	16881.53
TSformer-VO-2	10.65	96.77	17.98	16767.64
TSformer-VO-3	11.07	96.96	18.04	16637.78
RDS SLAM (Mask RCNN)	18.08	70.51	14.78	16000.26
RDS SLAM (SegNet)	16.26	60.20	13.49	14535.83
VDO SLAM	21.20	30.96	10.34	10852.83

resources on tracking (62.06%), while tasks such as mask generation (13. 47%) and map optimization (12.79%) consume significantly fewer resources (see Figure 4.4). Although this preprocessing strategy improves real-time performance during the execution of SLAM, it presents certain limitations. From a design perspective, offloading semantic segmentation and motion estimation to an earlier stage outside the main pipeline is not an ideal long-term solution. Ideally, these tasks should be integrated directly into the SLAM architecture to enable a more adaptive and responsive system that can adjust to new scenes in real time. By separating these processes from the SLAM pipeline, VDO-SLAM sacrifices flexibility, which could be a drawback in environments where on-the-fly processing of dynamic objects and scene changes is essential.

In contrast, the performance of RDS-SLAM is highly dependent on the choice of semantic segmentation model. When using Mask R-CNN, RDS-SLAM dedicates 51.64% of its computational resources to semantic segmentation, achieving high accuracy (ATE of 3.4 cm) but with a significant trade-off in real-time performance (3.2 FPS) and resource consumption (16W). This makes it less practical for power-sensitive applications. However, when SegNet is used, the performance of RDS-SLAM improves substantially, with an FPS of 7.5 and lower energy consumption (14.5W), making it comparable to VDO-SLAM. SegNet offers a lighter alternative that better balances the trade-off between accuracy and efficiency. Despite these improvements, both methods still lag behind true real-time performance, underscoring the challenge of optimizing SLAM systems for dynamic environments without excessive resource demands.

End-to-end deep learning-based approaches, such as DeepVO and TSformer-VO, exhibit significant limitations. Although this architecture promises to simplify the visual odometry pipeline, the results show that these approaches struggle in terms of accuracy and resource efficiency. For example, DeepVO achieves a poor ATE of 402.5 cm, reflecting significant drift

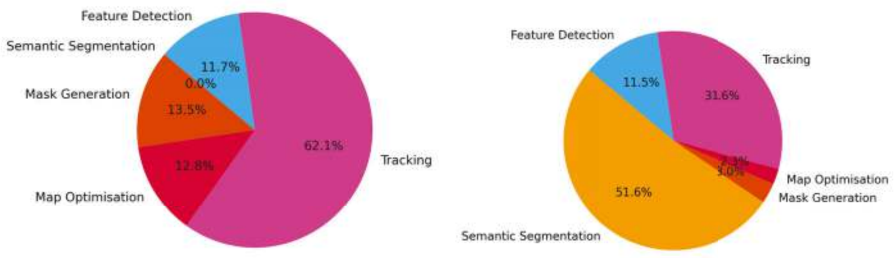


Figure 4.4 SLAM Block Execution Time Breakdown. Left: VDO-SLAM; Right: RDS-SLAM. ⏎

over time, particularly in environments that differ from the training datasets. Similarly, TSformer-VO, although slightly better in terms of accuracy, still performs poorly compared to geometric SLAM methods. More importantly, these end-to-end models suffer from extremely high GPU usage, with TSformer-VO consuming more than 95% of the available GPU resources. This excessive consumption of computational resources, coupled with limited accuracy, makes end-to-end approaches inefficient for real-world applications on embedded systems. Although they deliver higher FPS, they remain inadequate for real-time performance. Furthermore, their imprecision and inefficient resource utilization make them unsuitable for tasks that demand reliable localization.

4.4.2 TensorRT optimization for SLAM algorithms

Exporting the model to TensorRT is essential for running it on NVIDIA Jetson devices, as TensorRT is specifically designed to maximize the performance of deep learning models on NVIDIA GPUs. It achieves this by implementing optimizations such as precision calibration, kernel fusion, and layer fusion, which collectively improve inference speed and reduce memory usage. In our investigation into improving the efficiency of SLAM algorithms, we explored the possibility of optimizing the DeepVO PyTorch model using TensorRT. To accomplish this, the model was initially exported to the ONNX format, which serves as an intermediary representation that facilitates compatibility with TensorRT. Then, we evaluated the performance of the model using various precision formats, including FP32, FP16, and INT8, along with the quantization technique utilizing PQT (Post-Training Quantization). This allowed us to assess the trade-offs between model accuracy and computational efficiency across different precision levels. The results of these optimizations, in terms of speedup compared to the original PyTorch implementation, are presented in Table 4.3. These findings indicate that the application of TensorRT to a deep learning model for VO can result in significant speed enhancements while maintaining minimal variations in prediction accuracy. Depending on the chosen precision level, it is possible to effectively balance speed and accuracy, allowing tailored performance based on specific application needs.

Table 4.3 Performance Comparison of DeepVO-pytorch Optimized with TensorRT ⏎

Precision Format	Speedup over PyTorch	ATE (cm)
FP32	4.36x	405.5
FP16	3.80x	411.9
INT8	4.23x	468.5

4.5 Conclusion

This work evaluated the performance of various localization systems, such as SLAM and VO, on resource-constrained devices, specifically the NVIDIA Jetson AGX Orin 64GB. By benchmarking systems such as RDS-SLAM, VDO-SLAM, DeepVO, and TSformer-VO, the study provided valuable insights into the efficiency and accuracy of these methods under constraints of computational power, memory, and energy resources. The results highlighted the trade-offs between real-time performance and accuracy. Although RDS-SLAM achieved higher accuracy, it incurred greater computational and energy demands. VDO-SLAM performs at a similar level in terms of performance; however, it is important to note that the CNN-based component, semantic extraction, and optical flow processing are conducted upstream on pre-known datasets. Despite this pre-processing step, which limits its adaptability to dynamic environments or new scenarios where the models have not been previously trained, VDO-SLAM still fails to achieve efficient real-time performance. On the other hand, deep learning-based models like DeepVO and TSformer-VO showed limitations in accuracy and high resource consumption, making them unsuitable for real-time applications.

The study emphasized the need for optimization, particularly in integrating semantic segmentation directly into the SLAM pipeline for greater flexibility. The findings suggest that there is significant potential for further optimizing deep learning-based approaches to improve their viability on resource-constrained platforms. Future research could focus on co-designing, refining these algorithms, and exploring ways to balance accuracy and efficiency, while further tailoring them to the constraints of embedded hardware.

Acknowledgements

This work was conducted as part of a PhD thesis within the AI optimization roadmap at the LIAE laboratory of the DSCIN department at CEA-LIST.

References

[1] Y. Liu and J. Miura, "RDS-SLAM: Real-Time Dynamic SLAM Using Semantic Segmentation Methods," IEEE Access, vol. 9, pp. 23772–23785, 2021, doi: https://doi.org/10.1109/access.2021.3050617.

[2] J. Zhang, M. Henein, R. Mahony, and V. Ila, "VDO-SLAM: A Visual Dynamic Object-aware SLAM System," arXiv.org, 2020. https://arxiv.org/abs/2005.11052 (accessed Jan. 24, 2025).

[3] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015, doi: https://doi.org/10.1109/tro.2015.2463671.

[4] T. Qin, P. Li, and S. Shen, "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, Aug. 2018, doi: https://doi.org/10.1109/tro.2018.2853729.

[5] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang, "OpenVINS: A Research Platform for Visual-Inertial Estimation," May 2020, doi: https://doi.org/10.1109/icra40945.2020.9196524.

[6] R. Kang, J. Shi, X. Li, Y. Liu, and X. Liu, "DF-SLAM: A Deep-Learning Enhanced Visual SLAM System based on Deep Local Features," *arXiv.org*, 2019. https://www.semanticscholar.org/paper/DF-SLAM%3A-A-Deep-Learning-Enhanced-Visual-SLAM-based-Kang-Shi/6dd357e65acc5faae4e36c506553f51ec71777d2 (accessed Jan. 24, 2025).

[7] B. Bescos, C. Campos, J. D. Tardós, and J. Neira, "DynaSLAM II: Tightly-Coupled Multi-Object Tracking and SLAM," *arXiv.org*, 2020. https://arxiv.org/abs/2010.07820 (accessed Jan. 24, 2025).

[8] J. Czarnowski, T. Laidlow, R. Clark, and A. J. Davison, "DeepFactors: Real-Time Probabilistic Dense Monocular SLAM," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 721–728, Apr. 2020, doi: https://doi.org/10.1109/lra.2020.2965415.

[9] S. Wang, R. Clark, H. Wen, and N. Trigoni, "DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks," *IEEE Xplore*, May 01, 2017. https://ieeexplore.ieee.org/document/7989236

[10] Françani, André O and M. Marcos, "Transformer-based model for monocular visual odometry: a video understanding approach," *arXiv.org*, 2023. https://arxiv.org/abs/2305.06121

[11] Z. Teed and J. Deng, "DROID-SLAM: Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras," *arXiv:2108.10869 [cs]*, Feb. 2022, Available: https://arxiv.org/abs/2108.10869

[12] E. Sucar, S. Liu, J. Ortiz, and A. J. Davison, "iMAP: Implicit Mapping and Positioning in Real-Time," *arXiv.org*, 2021. https://arxiv.org/abs/2103.12352 (accessed Sep. 14, 2024).

[13] Z. Zhu *et al.*, "NICE-SLAM: Neural Implicit Scalable Encoding for SLAM," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2022, doi: https://doi.org/10.1109/cvpr52688.2022.01245.

[14] A. Rosinol, J. J. Leonard, and L. Carlone, "NeRF-SLAM: Real-Time Dense Monocular SLAM with Neural Radiance Fields," *arXiv.org*, Oct. 24, 2022. https://arxiv.org/abs/2210.13641

[15] D. Liao and W. Ai, "VI-NeRF-SLAM: a real-time visual–inertial SLAM with NeRF mapping," *Journal of Real-Time Image Processing*, vol. 21, no. 2, Feb. 2024, doi: https://doi.org/10.1007/s11554-023-01412-6.

[16] G. Li, Q. Chen, Y. Yan, and J. Pu, "EC-SLAM: Effectively Constrained Neural RGB-D SLAM with Sparse TSDF Encoding and Global Bundle Adjustment," *arXiv.org*, 2024. https://arxiv.org/abs/2404.13346 (accessed Jan. 24, 2025).

[17] Z. Zakaryaie Nejad and A. Hosseininaveh Ahmadabadian, "ARM-VO: an efficient monocular visual odometry for ground vehicles on ARM CPUs," *Machine Vision and Applications*, vol. 30, no. 6, pp. 1061–1070, May 2019, doi: https://doi.org/10.1007/s00138-019-01037-5.

[18] S. Bahnam, S. Pfeiffer, and G. C. H. E. de Croon, "Stereo Visual Inertial Odometry for Robots with Limited Computational Resources," *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*

(IROS), pp. 9154–9159, Sep. 2021, doi: https://doi.org/10.1109/iros51168.2021.9636807.

[19] L. Xiao, J. Wang, X. Qiu, Z. Rong, and X. Zou, "Dynamic-SLAM: Semantic monocular visual localization and mapping based on deep learning in dynamic environment," *Robotics and Autonomous Systems*, vol. 117, pp. 1–16, Jul. 2019, doi: https://doi.org/10.1016/j.robot.2019.03.012.

[20] T. Ji, C. Wang, and L. Xie, "Towards Real-time Semantic RGB-D SLAM in Dynamic Environments," *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11175–11181, May 2021, doi: https://doi.org/10.1109/ICRA48506.2021.9561743.

[21] HiIAmTzeKean, "GitHub - HiIAmTzeKean/Jetson-Nano-SLAM: A Nanyang Technological University research project. Multi-USB-Cam implementation on Jetson Nano.," *GitHub*, 2022. https://github.com/HiIAmTzeKean/Jetson-Nano-SLAM

[22] M. Ali, M. Erfani, and J. Mesbah, "Same App, Different App Stores: A Comparative Study," doi: https://doi.org/10.1145/1235.

[23] G. Lentaris, Ioannis Stamoulias, Dimitrios Soudris, and Manolis, "HW/SW Codesign and FPGA Acceleration of Visual Odometry Algorithms for Rover Navigation on Mars," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 8, pp. 1563–1577, Aug. 2016, doi: https://doi.org/10.1109/tcsvt.2015.2452781.

[24] D. Törtei Tertei, J. Piat, and M. Devy, "FPGA design of EKF block accelerator for 3D visual SLAM," *Computers & Electrical Engineering*, vol. 55, pp. 123–137, Oct. 2016, doi: https://doi.org/10.1016/j.compeleceng.2016.05.003.

[25] Z. Zhang, A. Suleiman, L. Carlone, V. Sze, and S. Karaman, "Visual-Inertial Odometry on Chip: An Algorithm-and-Hardware Co-design Approach," *DSpace@MIT (Massachusetts Institute of Technology)*, Jul. 2017, doi: https://doi.org/10.15607/rss.2017.xiii.028.

[26] J. Liu, X. Li, Y. Liu, and H. Chen, "RGB-D Inertial Odometry for a Resource-Restricted Robot in Dynamic Environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 9573–9580, Oct. 2022, doi: https://doi.org/10.1109/lra.2022.3191193.

[27] J. Kühne, M. Magno, and L. Benini, "Low Latency Visual Inertial Odometry with On-Sensor Accelerated Optical Flow for Resource-Constrained UAVs," *IEEE Sensors Journal*, pp. 1–1, 2024, doi: https://doi.org/10.1109/jsen.2024.3406948.

[28] T. Peng, D. Zhang, D. L. N. Hettiarachchi, and J. Loomis, "An Evaluation of Embedded GPU Systems for Visual SLAM Algorithms," *Electronic Imaging*, vol. 2020, no. 6, pp. 325–1325–6, Jan. 2020, doi: https://doi.org/10.2352/issn.2470-1173.2020.6.iriacv-074.

[29] S. Sumikura, M. Shibuya, and K. Sakurada, "OpenVSLAM: A Versatile Visual SLAM Framework," doi: https://doi.org/10.1145/3343031.3350539.

[30] X. Liu, Y. Yang, B. Xu, and S. Schwertfeger, "Benchmarking SLAM Algorithms in the Cloud: The SLAM Hive System," *arXiv.org*, 2024. https://arxiv.org/abs/2406.17586.

[31] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *arXiv.org*, 2017. https://arxiv.org/abs/1703.06870.

[32] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *arXiv:1511.00561 [cs]*, Oct. 2016, Available: https://arxiv.org/abs/1511.00561

[33] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, "PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, doi: https://doi.org/10.1109/cvpr.2018.00931.

[34] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2012, doi: https://doi.org/10.1109/iros.2012.6385773.

[35] Z. Shan, R. Li, and S. Schwertfeger, "RGBD-Inertial Trajectory Estimation and Mapping for Ground Robots," *Sensors*, vol. 19, no. 10, p. 2251, May 2019, doi: https://doi.org/10.3390/s19102251.

4.6 Appendix

4.6.1 Calculation of the metrics used in evaluation

In order to evaluate the accuracy and performance of SLAM algorithms, we employ several key metrics that assess both the global alignment and local consistency of the estimated trajectories. The following metrics provide a comprehensive understanding of the system's behavior, including Absolute Trajectory Error (ATE), Relative Pose Error (RPE), and Frames Per Second (FPS). In addition, we monitor system resource usage using hardware statistics on CPU, GPU, memory, and power consumption to evaluate the efficiency on embedded platforms.

4.6.1.1 Absolute Trajectory Error (ATE)

The Absolute Trajectory Error (ATE) is used to measure the global accuracy of the SLAM system by evaluating the difference between the predicted trajectory and the ground-truth trajectory after aligning them. Let the ground-truth poses be represented as a series of homogeneous transformation matrices $\mathbf{P}_i^{gt} \in SE(3)$ where *i* indexes the sequence frames, and let the estimated poses be $\mathbf{P}_i^{est}$. The ATE is computed as the root mean square error (RMSE) between the positions of the estimated trajectory and the ground truth after aligning them through a rigid body transformation.

The ATE is mathematically defined as:

$$\text{ATE}_{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \| \mathbf{t}_i^{gt} - \mathbf{t}_i^{est} \|^2}, \tag{4.1}$$

where $\mathbf{t}_i^{gt}$ and $\mathbf{t}_i^{est}$ represent the translation vectors (positions) extracted from the ground-truth and estimated poses $\mathbf{P}_i^{gt}$ and $\mathbf{P}_i^{est}$, respectively. The RMSE measures the Euclidean distance between corresponding positions in both trajectories, giving a global measure of accuracy.

4.6.1.2 Relative Pose Error (RPE)

The Relative Pose Error (RPE) evaluates the local consistency of the trajectory by comparing the relative motion between consecutive poses in the estimated trajectory to that in the ground-truth trajectory. This metric is essential for assessing the short-term accuracy of the system in tracking small movements, which is critical in dynamic environments.

Given two consecutive ground-truth poses $\mathbf{P}_i^{gt}$ and $\mathbf{P}_{i+1}^{gt}$, the relative transformation between these poses is:

$$\mathbf{T}_i^{gt} = \left(\mathbf{P}_i^{gt}\right)^{-1}\mathbf{P}_{i+1}^{gt}. \tag{4.2}$$

Similarly, the relative transformation between consecutive estimated poses is:

$$\mathbf{T}_i^{est} = \left(\mathbf{P}_i^{est}\right)^{-1}\mathbf{P}_{i+1}^{est}. \tag{4.3}$$

The RPE is then computed as the difference between the relative transformations of the ground-truth and estimated poses. The translational RPE is defined as the root mean square error (RMSE) of the translation vectors from the relative transformations:

$$\mathrm{RPE}_{trans} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left\| \left(\mathbf{t}_i^{gt} - \mathbf{t}_{i+\triangle}^{gt}\right) - \left(\mathbf{t}_i^{est} - \mathbf{t}_{i+\triangle}^{est}\right) \right\|^2}, \tag{4.4}$$

where $\mathbf{t}_i^{gt}$ and $\mathbf{t}_i^{est}$ are the translation components of the relative transformations $\mathbf{T}_i^{gt}$ and $\mathbf{T}_i^{est}$ and $\triangle$ is the time interval.

For the rotational RPE, we measure the angular difference between the relative rotation matrices $\mathbf{R}_i^{gt}$ and $\mathbf{R}_i^{est}$, which can be quantified using the angle-axis representation. The rotational error is defined as:

$$\mathrm{RPE}_{rot} = \frac{1}{n}\sum_{i=1}^{n} \arccos\left(\frac{\mathrm{tr}\left(\left(\mathbf{R}_i^{gt} - \mathbf{R}_{i+\triangle}^{gt\,\mathrm{T}}\right)^{\mathrm{T}}\left(\mathbf{R}_i^{est} - \mathbf{R}_{i+\triangle}^{est\,\mathrm{T}}\right)\right) - 1}{2}\right), \tag{4.5}$$

where $\mathbf{R}_i^{gt}$ and $\mathbf{R}_i^{est}$ represent the rotation matrices of the ground-truth and estimated poses, tr denotes the trace of a matrix, and $\triangle$ is the time interval. This metric provides the average rotational error over the trajectory.

4.6.1.3 Frames Per Second (FPS)

In real-time applications, the Frames Per Second (FPS) is a critical metric that measures the number of frames processed by the SLAM system per second. High FPS is essential for ensuring that the SLAM algorithm operates efficiently and in real time, especially in embedded or constrained environments where computational resources are limited. FPS can be computed as:

$$\mathrm{FPS} = \frac{\text{number of frames}}{\text{total processing time}}. \tag{4.6}$$

This metric helps assess the speed and responsiveness of the SLAM system.

4.6.2 Alignment methods

In this section, we describe four different trajectory alignment methods commonly used to compare predicted poses against ground-truth data in odometry and SLAM systems: scale, 6DOF, 7DOF, and 7DOF with scale. Each method focuses on optimizing different parameters (scale, rotation, and translation) to minimize the alignment error.

4.6.2.1 Scale alignment (scale)

The **scale** alignment adjusts only the global scale factor c between the predicted trajectory *X* and the ground-truth trajectory *Y*, without modifying the rotations or translations.

Method:

- The objective is to minimize the distance between *X* and *Y* by adjusting only the scale.
- The optimal scale factor c is computed as:

$$c = \frac{\sum (X \,.\, Y)}{\sum X^2}. \tag{4.7}$$

- The aligned trajectory is then scaled as:

$$X_{aligned} = \ c \,.\, X. \tag{4.8}$$

This method is particularly useful for monocular systems, where the scale is typically unknown and must be estimated post-facto.

4.6.2.2 6 Degress of Freedom (6DOF)

The **6DOF** alignment adjusts both the rotation and the translation between the predicted trajectory *X* and the ground-truth *Y*, but keeps the scale fixed. This allows the correction of orientation and position errors.

Method:

- The six degrees of freedom include three for rotation and three for translation.
- The alignment is based on the Singular Value Decomposition (*SVD*) of the covariance matrix between *X* and.
- The covariance matrix cov_{xy} is computed as:

$$\text{cov}_{xy} = \frac{1}{n} \overset{n}{\underset{i=1}{}} (y_i - \text{mean}_y)(x_i - \text{mean}_x)^{\mathrm{T}}. \tag{4.9}$$

- The SVD of cov_{xy} gives:

$$\text{cov}_{xy} = \ UDV^T. \tag{4.10}$$

where *U* and *V* are orthogonal matrices representing the principal directions of *Y* and *X*, respectively, and *D* is a diagonal matrix of singular values.

- The rotation matrix *R* is computed as:

$$\mathrm{R} = U \qquad V^T. \tag{4.11}$$

where Σ is a diagonal matrix ensuring a proper right-handed coordinate system.
- The translation vector *t* is calculated as:

$$\mathrm{t} \;= \mathrm{mean}_y - R.\mathrm{mean}_x. \tag{4.12}$$

- The final aligned trajectory is:

$$X_{\text{aligned}} = \; \mathrm{R} \,.\, \mathrm{X} \,+\, \mathrm{t}. \tag{4.13}$$

This method is ideal for LiDAR or stereo systems where scale is fixed but orientation and position errors need correction.

4.6.2.3 7 Degress of Freedom (7DOF)

The **7DOF** alignment extends the 6DOF method by also adjusting the scale factor *c*, in addition to the rotation and translation. This allows for simultaneous correction of scale, rotation, and translation errors.

Method:

- In addition to the rotation *R* and translation *t*, the scale factor *c* is computed to minimize the alignment error.
- The scale factor *c* is given by:

$$\mathrm{c} \;= \frac{1}{\sigma_x} \,.\, tr(D\Sigma), \tag{4.14}$$

where σ_x is the variance of the points in *X*, and $tr(D\Sigma)$ is the trace of the product of the singular values and the diagonal matrix Σ.
- The transformed trajectory becomes:

$$X_{\text{aligned}} \;= \; \mathrm{c} \,.\, \mathrm{R} \,.\, \mathrm{X} \,+\, \mathrm{t}. \tag{4.15}$$

This approach is beneficial in systems where the scale might not be exactly known, such as stereo or some lidar systems.

4.6.2.4 Scale + 7 Degrees of Freedom

The **scale_7DOF** alignment combines the benefits of the 7DOF alignment with an additional scale optimization step. After applying the rotation and translation, the scale factor is optimized separately to further minimize the final trajectory error.

Method:

- First, perform a 6DOF alignment to adjust the rotation *R* and translation *t*.
- Then, optimize the scale factor *c* independently using the formula:

$$\mathrm{c} = \frac{\sum (X_{aligned} \cdot Y)}{\sum (X_{aligned}{}^2)}. \tag{4.16}$$

- The final trajectory becomes:

$$X_{final} \;=\; c \cdot X_{aligned}. \tag{4.17}$$

This method provides a finer correction of scale after the rotational and translational alignment, making it useful when significant scale variations exist between predicted and ground-truth trajectories.

5

Designing Accelerated Edge AI Systems with Model Based Methodology

Petri Solanti[1] and Russell Klein[2]

[1]Siemens EDA, Germany
[2]Siemens EDA, USA

Abstract

Deploying AI at the edge can be challenging. AI algorithms are very compute intensive. In the data centre, multiple large, power-hungry GPUs are often employed. However, edge systems typically have constrained compute capabilities and limited power. Further, many systems need to deal with size, weight, cost, thermal, and other limitations. Successfully deploying AI while meeting these limitations requires a holistic analysis of the system. A Model-Based Cybertronic System Engineering (MBCSE) methodology enables modelling and analysis of complex systems at a high abstraction level. It can be used to analytically find an optimal system architecture and hardware/software partitioning. Meeting the computational requirements may call for the development of bespoke machine learning accelerators. These are complex dedicated compute resources that deliver parallel computation, local data buffers, and some level of programmability. Designing an optimal accelerator architecture can be accomplished with an AI assisted High-Level Synthesis (HLS) process to efficiently explore the design space.

This paper describes a Model Based Cybertronic System Engineering (MBCSE) methodology that can be used to craft a combined hardware/software implementation of an inferencing algorithm, balancing performance, power, cost, and other key design metrics. It begins with an algorithmic analysis, determining areas of significant complexity. This is followed by allocation of functions to physical computation elements, targeting

DOI: 10.1201/9788743808862-5

both board-level and chip-level placement. During the allocation phase, complex algorithms may be mapped to bespoke accelerators that will be synthesized from the algorithmic description using high-level synthesis. Finally, an analysis of design is performed to ensure that all design metrics are met.

Keywords: edge AI, system design, system optimization, high-level synthesis.

5.1 Introduction and Background

Edge systems are often limited in several dimensions. Power consumption and compute capabilities are often limited in support of form-factor, weight, cost, mobility, and other requirements. This makes deploying AI on these systems challenging, as AI algorithms typically consume significant compute resource and power. One way to mitigate this is to architect the system with the compute resources that meet, but do not materially exceed, the requirements for the AI processing needed.

AI processing can be performed on different types of compute resources. For example, inferences can be run on general-purpose processors, graphics processing units (GPUs), arrays of multiply/accumulate processing elements, FPGA fabric, or bespoke hardware accelerators. Each of these represents compromises for the system designer. For example, deploying AI on a general-purpose processor typically will require around 100 watts of power or more. This results in a heavy battery, limited battery life, and potential cooling issues. But it significantly eases the development efforts, as the same software and machine learning frameworks can be used in the edge design as was used by the data scientists in the data centre. A bespoke hardware accelerator will be orders of magnitude more efficient but requires a custom IC development effort.

Developers need a way to understand the impact of their design tradeoffs early in the design cycle to find the optimal architecture for the system that addresses the myriads of design constraints imposed by the business, regulatory, competitive, and other forces influencing the design of the system. Yet, finding a suitable methodology that can address all the design needs is tricky.

Model-based Systems Engineering has been used for software development for 25 years. Unified Modelling Language (UML) [1] was the first standardized graphical modelling language for specification, construction, documentation and visualization of software intensive systems. Because

UML was originally developed for modelling SW systems, it was not suitable for general system modelling. System Modelling Language (SysML) was based on UML but targeted more to the needs of general systems engineering. It is widely used for modelling system functionality but has severe capacity limitations.

Systems with functionality implemented in both SW and bespoke HW blocks are called Cybertronics Systems [2]. These are difficult to model with UML or SysML, which are targeted to single-domain modelling. Finding the optimal HW/SW partitioning needs an extensive design space exploration and capability to analyse different metrics like processor and bus utilization, task latency, power consumption or network load. The modelling methodology must support clear separation of function from structure, function allocation to structural elements and mapping of structural elements to any target technology. One of the methodologies developed for this purpose is Architecture Analysis and Design Language (AADL) [3]. It is used to model the SW and HW architectures of embedded real-time systems. AADL is a useful methodology for HW/SW system modelling and analysis, but it is lacking capabilities to the operational and functional analysis and modelling of cyber-physical systems.

5.2 Model Based Cybertronic Systems Engineering

Challenges of the cybertronics systems engineering are diverse. It begins with the system context that can be a network, a computing enclosure with multiple Printed Circuit Boards (PCB), a single PCB, a System-on-Chip (SoC), a Field Programmable Gate Array (FPGA) or embedded SW. The physical system sets the functional constraints that define the requirements for the cybertronics subsystems. Furthermore, the individual algorithms communicating with each other and consuming computing resources, can be allocated to different processing elements. The design space is huge and finding the optimal architecture is difficult.

Another challenge is the variety of the design domains that are usually tightly coupled. An architectural component like PCB contains other architectural components that are cybertronics subsystems themselves such as 3DIC or SoC. SW functionality can be allocated onto multiple processors that are potentially in different subsystems. In such cases the network of data exchanges between the functions must be allocated to physical interconnect that can be a network segment, platform bus like PCIe, network-on-chip, or something similar, depending on the implementation technology.

Third challenge is the fragmented organization of the domain specific development processes. Communication between the teams is negligible because the different terminologies and specifications are interpreted differently. This makes maintaining the integrity of the system difficult.

Model Based Cybertronic Systems Engineering is a new model-based methodology developed for modelling and analysis of cybertronics systems. It borrows concepts from multiple modern systems engineering methodologies. The main target of this new methodology is to unify the modelling methodologies on different system levels and enable seamless communication between the architects in the different implementation domains and the design teams.

The first methodology is called HW/SW co-architecting [4]. It is based on C++ function tree breakdown, where the functions are allocated to SW that is executed on one or more processors or HW accelerators that are implemented by using HLS. Because the C++ source code can be used both as SW code and input for the HLS, the mapping decisions can be done late in the design cycle. This approach works nicely for algorithms that are available, or can be translated into, C++, like streaming video or AI algorithms.

The second methodology used in MBCSE is ARChitecture Analysis and Design Integrated Approach, ARCADIA [5]. It is a very simple static information model methodology for system modelling according to the INCOSE SE handbook. Arcadia can be used to model the functional, logical, and physical architecture of the system using standardized artifacts. The functions and structures are kept separated, but the allocation of functions, exchanges, and components to structures is supported. Arcadia methodology has no technology binding nor simulation capabilities itself, so the designer can specify any target technology or simulation by using properties. It also has capability to transition a component to a subsystem in a separate project. These three capabilities make Arcadia a perfect methodology for cybertronics system modelling.

Yet, a comprehensive system modelling methodology must enable design space exploration with different types of simulations. Property Model Methodology [6] is a dynamic, simulation-based methodology for requirements driven system modelling and analysis. Simulations are needed to validate the functional correctness of the system model, to analyse the system performance of different architecture options and verify the implementation. PMM and Arcadia complement each other and form a basis for a cybertronics system modelling and validation methodology.

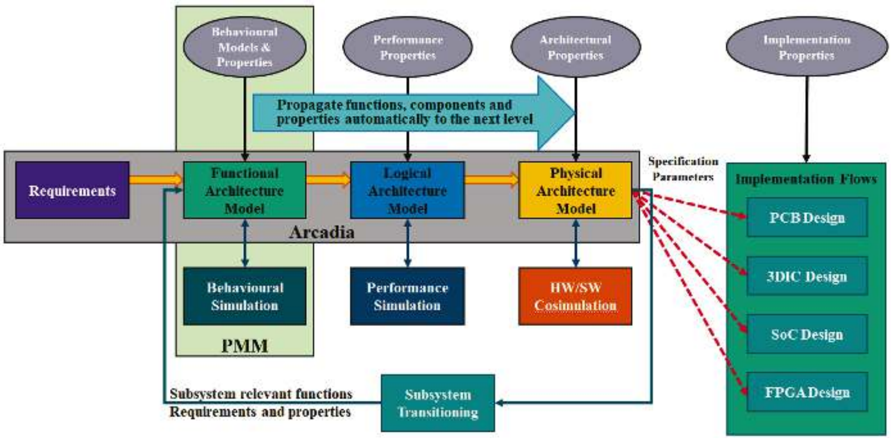

Behavioural Models & Properties
Performance Properties
Architectural Properties
Implementation Properties
Propagate functions, components and properties automatically to the next level
Requirements
Functional Architecture Model
Logical Architecture Model
Physical Architecture Model
Arcadia
Specification Parameters
Implementation Flows
PCB Design
3DIC Design
SoC Design
FPGA Design
Behavioural Simulation
Performance Simulation
HW/SW Cosimulation
PMM
Subsystem relevant functions Requirements and properties
Subsystem Transitioning

Figure 5.1 Model-Based Cybertronics Systems Engineering Methodology ⏎

MBCSE [7] workflow is described in the Figure 5.1. The horizontal Requirements-Functional-Logical-Physical (RFLP) architecture process is based on Arcadia. The transition from one design layer to another one is automated to maintain the integrity between the layers.

The vertical pillars for each Arcadia layer represent the PMM concepts used to import behavioural, performance, and implementation information to the Arcadia model and interfacing to the different simulations. Complexity management with subsystem transition is part of the Arcadia methodology.

The third integral part of the MBCSE methodology is interface to all cybertronics implementation domains. To keep the number of details to a manageable level in the system model the transition to a domain specific design flow is a design step, not an automatic model generation.

The MBCSE methodology allows modelling any type of system. Therefore, it can be used for modelling of cyber-physical systems with, for example, mechanical, mechatronics, electrical, and physical subsystems as well. Each model can be specified to be an abstract model without any technology binding, or it can be bound to a target technology like a platform network or PCB, where the components in the model are physical components at this model level like a processor, SoC, or FPGA. But the component is a top-level functional model of a separate subsystem that is decomposed individually. Yet, all functions, requirements, and properties related to the top-level are maintained there and propagated to the subsystem automatically.

5.3 Designing Edge AI Systems with MBCSE Methodology

Edge AI systems have a special character. They are power and computing resource limited and usually have hard real-time requirements. At the same time, they apply multiple cascaded, or nested, AI algorithms. Therefore, they are perfect candidates for model-based design methodology described in Figure 5.2.

The design process begins in the traditional AI algorithm development environment like TensorFlow, Caffe, or PyTorch. Once the algorithm is optimized for accuracy and mathematical complexity, the network is translated to C++ using one of the Python-to-C converters. The generated C++ code is validated against the original Python model to ensure its correctness.

The same floating-point C++ model is a functional description of the AI system functionality and is used as a starting point for the architecture analysis and development. In the functional analysis the AI algorithm can be treated as one entity, but for a more detailed analysis it must be broken down to smaller entities that usually are the individual layers of the neural network. Arcadia methodology enables function breakdown into any granularity. For example, in the top-level system context the AI algorithm can be one function, which is broken down into the individual layers in the lower subsystem level. Additional information like kernel size or number of channels can be added to the model as properties.

In the logical architecture phase, the functions are allocated to logical components and the data exchanges between them through logical channels. This is the first attempt to allocate the functionality to an implementation

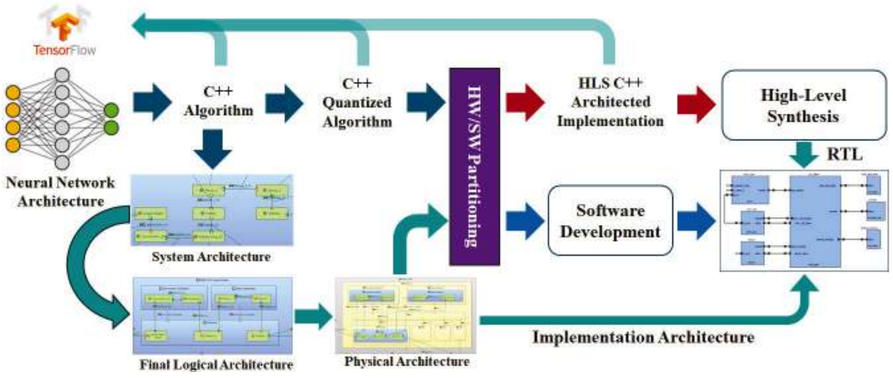


Figure 5.2 MBCSE Process for AI system design ⏎

architecture. The amount of information at this point is still very small and it is easy to change the logical allocation by just assigning a function to another logical component. This architecture exploration phase enables a performance simulation to analyse the latencies, interconnect traffic, processor utilization, and many other metrics. The performance model can be parameterized to test what-if exploration of design alternatives as shown in Figure 5.3. Here, the convolution function is changed from a SW implementation to a HW accelerator resulting in a 96% reduction in the latency of the computation without changing the performance model.

The purpose of the performance analysis is to find the optimal HW/SW allocation and simultaneously design constraints for the HW designers. When the optimal architecture is found, a physical architecture model is created. It contains more detailed implementation information for creating a virtual model for more detailed analysis and HW implementation.

Parallel to the architectural exploration, the C++ functions targeted to HW acceleration are quantized by using fixed-point data types to model the HW implementation effects to the AI algorithm. This quantized C++ code is validated against the original Python model using the same testbench as earlier for functional validation.

In the next step the quantized C++ code is partitioned into HW models that are further optimized for HLS, while functions that remain as SW will go through the standard SW development process.

The HW platform integration is based on the physical architecture model. The components, ports, and interconnects can be extracted from the system

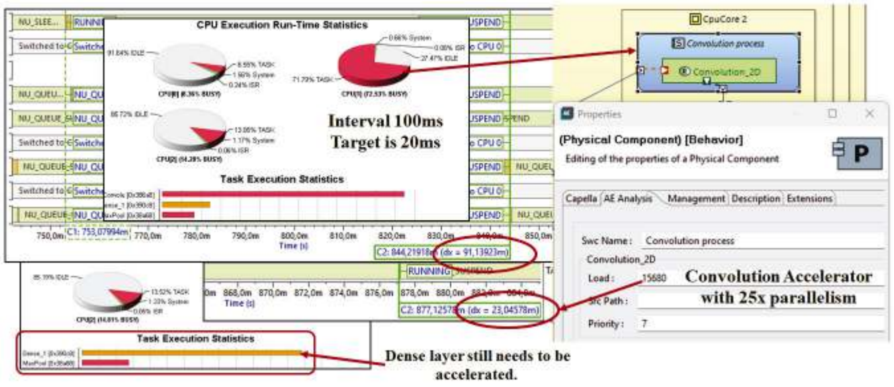

CPU Execution Run-Time Statistics
Interval 100ms
Target is 20ms
Task Execution Statistics
C2: 844,21918m (dx = 91,13923m)
C2: 877,12578m (dx = 23,04578m)
CpuCore 2
Convolution process
Convolution_2D
Properties
(Physical Component) [Behavior]
Editing of the properties of a Physical Component
Capella
AE Analysis
Management
Description
Extensions
Swc Name: Convolution process
Convolution_2D
Load: 15680
Src Path:
Priority: 7
Convolution Accelerator with 25x parallelism
Dense layer still needs to be accelerated.

Figure 5.3 Performance exploration ⏎

model and refined in the domain specific design environment. Standard components like processors, memories, and peripherals can be taken from the library, while custom models are synthesized to RTL using HLS.

When the platform is integrated, it goes through the domain specific verification flow.

5.4 Creation of Bespoke AI Accelerator

Once the MBCSE analysis shows that a bespoke AI accelerator is needed, whether deployed as an ASIC or FPGA, a hardware design project must commence to design, verify, and integrate that hardware accelerator. MBCSE provides a set of requirements and constraints for the AI accelerator. It also defines the data flows, both the inputs and outputs, to the larger system. This constrains the interfaces used for the implementation of the AI accelerator.

While the deployment of the accelerator must meet the constraints derived from the system analysis, there is still a great deal of latitude in the implementation. The nature of hardware design is such that aspects of the design, such as power consumption or throughput, could vary by an order of magnitude or two, while meeting the remaining constraints on the system. This means that it is critical to understand both the constraints as well as the limitations of the constraints. For example, there may be a requirement that an inference must be completed in 2 milliseconds. This is a clear performance requirement. But this computation may be performed in parallel with another calculation that takes, say, 1.5 milliseconds, where the slower of the two limits the overall throughput. Or the collection of the feature data from sensors may take 1.5 milliseconds. In these cases, there will be no benefit to making the AI accelerator go any faster than 1.5 milliseconds. Implementing a faster accelerator will typically degrade other system characteristics, such as power consumption or silicon area. However, for some systems, achieving higher performance or lower power consumption is better, with no limit or point of diminishing returns. Thus, itis critical for the designer to know both the constraints and the limitations of the constraints to architect an optimal implementation. The MBCSE flow delivers this data, and it should be used to inform the design process.

With the constraint data, the hardware development effort can commence. However, a traditional hardware development cycle for either ASIC or FPGA can take months and requires a level of manual effort such that it is not practical to create multiple implementations that are meaningfully different.

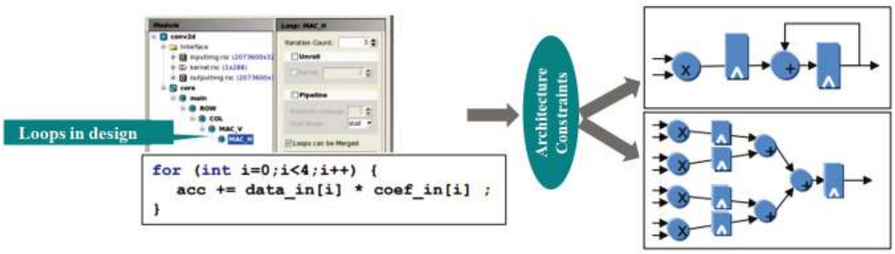
In the traditional hardware design flow, an initial architecture is decided upon, and minor course corrections can be made through the design cycle. But few design teams have the resources to do true design space exploration. If the design marginally meets the requirements, it will be accepted, even if it is sub-optimal. Traditional hardware design relies on the experience and intuition to select an architecture. However, if the design team has not built a bespoke AI accelerator before, there is likely little or no experience base to draw from.

5.4.1 High-Level Synthesis

High-Level Synthesis is a technology that takes an algorithm as input and produces a target technology specific implementation in the form of a synthesizable RTL design. The input algorithm is generally in the form of C++ or SystemC, but new approaches are supporting SystemVerilog and AI framework CNNs as input [8], [9].

High Level Synthesis is a well-established technology [10], available as open-source projects and from commercial EDA and FPGA suppliers. HLS works by parsing the input algorithm and creating a global control data-flow graph (CDFG) of the algorithm. This graph is analysed to determine data dependencies, parallelism in the algorithm, and potential resource sharing opportunities. The graph is then transformed into a set of state machines and data flow constructs that exactly implement the original algorithm. From this the RTL representation is constructed. The RTL implementation requires the target clock frequency, and information about the target silicon technology. This data would be in the form of an ASIC technology data for a targeted ASIC, or for FPGAs, detailed information on the characteristics of the FPGA design elements available.

The HLS tool can create a variety of implementations from a single input algorithm. For example, a multiply/accumulate loop could be fully unrolled, with a physical multiplier for every multiplication operation. Or it could share a single multiplier instance for each loop iteration. The former implementation would be fast but would be large and inflexible. The latter would be slower but would be smaller and could more easily accommodate a varying number of loop iterations. Or, a partial unrolling could be done, giving a balance between the two. Typically, the hardware engineer will provide guidance to the tool in the form of programming "pragmas" or tool settings. This gives the developer control over the level of parallelism and pipelining in the resulting implementation, See Figure 5.4.



```
for (int i=0;i<4;i++) {
   acc += data_in[i] * coef_in[i] ;
}
```

Figure 5.4 Alternative micro-architectures from a single source, based on tools settings and constraints ⏎

Since the creation of each RTL implementation is automated, the creation of several meaningfully different design options can be done quite quickly, in hours instead of months. In a traditional design flow, this would be prohibitively expensive. Each implementation can then be evaluated for power, performance, and area (PPA). From the scheduled state machine construction, the HLS tool can accurately calculate the latency for the algorithm for a given implementation. With knowledge of the target ASIC technology or FPGA device the HLS tool can ascertain the specific design primitives used to construct each design alternative which provides a good basis for determining the area. When combined with design activity (switching data), this can be used to produce an estimate of the power consumption. This power estimate will include the static power for each gate, as well as the dynamic switching power. But it will not include the parasitics, which are only available after RTL synthesis and place and route. While the power estimates will have some degree of uncertainty, they are still useful for performing comparisons between different implementations.

5.4.2 Implementation and optimization with HLS

With an HLS flow in place, deploying a bespoke accelerator is more than just running the HLS tool. There are several modifications that can be done that will improve the operating characteristics of the accelerator. These will result in an implementation optimized for the specific application. The first is to reduce the size of the network as much as possible. This involves well understood pruning techniques. Often a trained network can be reduced by 90% or more with little to no reduction in accuracy [11].

Next would be select an optimal numeric format, or "quantizing" the network. Using a more compact numeric representation will reduce the size of memory needed for weights, biases, and intermediate results. It will also reduce the size of the operators that process that data. Multipliers will take up the most area of the computational logic of the accelerator. And memory, used for storing weight, biases, features, and intermediate results will take up the most area and consume the most power. In terms of performance, movement of the data to and from the processing elements will be the biggest performance bottleneck.

In machine learning frameworks running on general purpose computers or GPUs, numbers are stored and operated on as 32-bit IEEE format floating-point numbers. But, for most AI processing, values usually small in magnitude, with a significant portion being between -1.0 and 1.0. Which means that a 32-bit floating point representation is excessive, resulting in significant unused range, which is wasteful in the design. Eliminating these makes the design smaller and more efficient. AI algorithms usually do not need to dynamic range supported by floating point numbers. Converting to a smaller numeric format such as fixed point, reduced precision floating point representations like bfloat16 [12] or posits [13] will result in a reduction in memory storage requirements, faster movement of the data, and smaller operators in hardware. Using quantized aware training, the size of representation can often be reduced by a factor of 4 or more [14]. By moving from a 32-bit floating point number to an 8-bit fixed point number will reduce the size of the multipliers by $\sim$97%, with a roughly proportional reduction in the power consumption. In an HLS flow the developer can choose to create a much smaller and more efficient design, or the designer could use the silicon area to add more multipliers and execute them in parallel, delivering increased performance.

By changing both the neural network architecture, through pruning, and the underlying math, through quantization, the accuracy of the network will change. Thus, it is critical that throughout the pruning and quantization steps that the accuracy is verified. Quantization and pruning actions will impact each other and can be traded-off against each other. For example, reducing the size of features or weights will reduce the accuracy of the neural network. Increasing the number of layers or channels within each layer will increase the accuracy of the neural network. Careful balancing of the pruning and quantization will result in an optimal implementation. There are AI approaches being researched for finding an optimal quantization and neural network architecture, as this is an unbounded search space it is an ideal application for neural networks [15].

While optimizing the neural network, it is important to consider the impact of these changes on PPA. HLS tools that provide PPA estimates, as described earlier, allow the developer to make informed trade-offs between micro-architectural alternatives in a way that is simply not possible in a traditional RTL design flow.

5.4.3 Verification and integration

Once the accelerator is created, the next step is verification of the accelerator and integration into the larger system. Verification can be done with both formal methods and dynamic verification. One approach is to formally prove equivalence between the algorithmic description and the synthesized RTL. Some HLS tools provide this capability. However, it is unusual for formal equivalence to fully verify the accelerator. To complete the verification, the algorithmic model can be instrumented to collect stimulus and expected responses that can be used in dynamic simulations. The combination of static and dynamic approaches will often reach full verification coverage much faster than manual methods.

Integrating the accelerator into the larger system means creating bus and memory interfaces to the external circuitry, in addition to wired connections. Most HLS tools have interface synthesis, which will create commonly used bus protocols and supports memory interfaces. These will be supported for both pre- and post-synthesis models, enabling common testbenches and verifications strategies across abstraction levels.

5.5 Exemplary Results

To illustrate these concepts, we describe an exemplary system where this design flow is applied. The system is a handheld address scanner that reads postal codes off address labels on letters and packages. There are multiple design domains in this project: the physical design of the package/enclosure, the design of the printed circuit board (PCB) electronic sub-system, and a system-on-chip (SoC). All these design domains need to support multiple requirements: weight, battery life, performance, accuracy, communications, security, and more. Balancing these requirements and demands require a holistic approach that embodies the multiple design domains and facilitates communication between divergent design teams.

For the electronics subsystem implementation there are three options: using a standard processor on a PCB, using a standard microcontroller and FPGA for accelerating some functions, or a custom SoC. Using the performance simulator in the cybertronics module level, which contains all electronics and SW functions, the design space was narrowed down to SoC. A standard processor solution can't meet the timing and power consumption requirements. Also, the thermal issues preclude this option. Using a microcontroller and FPGA would meet the timing requirements, but the power consumption of the FPGA exceeds the required limits, therefore the only implementation option is an SoC.

The SoC implementation was thoroughly analysed using the performance simulation and the optimal architecture contains a microcontroller and two bespoke accelerators: one for convolution and max pooling and another for the two dense layers. Based on the performance analysis, the convolution accelerator can use 4 milliseconds for one feature map and the dense accelerator 3 milliseconds including data transfer times. The HLS design space exploration was used to optimize the latency, area and power consumption. Because the data and weights are stored locally in the accelerators, the processing time is dominant in the latency and the target latency can be reached with a minimum resource implementation.

5.6 Conclusion

Complexity of the AI algorithms in the Edge systems is growing rapidly. Using general purpose processors or GPUs is no more practical because of the high computational load, power consumption limitations, and timing constraints. Most of the current design tools and methodologies focused on automating the flow from AI algorithm development to implementation, but they do not take in account the system-level aspects. The edge systems require a careful analysis of the different implementation options, which needs a new holistic approach that can handle the complexity of the system level architecture.

Model-based cybertronics systems engineering methodology enables the analysis and partitioning of the algorithms on multiple hierarchy levels enabling distribution of algorithms across a multi-device system and optimizing the device architecture to provide optimal HW resources for the execution of the given algorithm. The C++-based modelling approach allows free function allocation to different processing elements and creating bespoke accelerators using HLS. This capability streamlines the development process and enables late changes of function allocation between HW and SW.

Acknowledgements

Special thanks to Ahmed Hamza, Brendan Hall and Susanna Solanti-Iltanen for their contribution to the methodology development.

References

[1] "Unified Modeling Language Specification Version 2.0", https://www.omg.org/spec/UML/2.0/, July 2005

[2] M. Eigner, C. Muggeo, T. Dickopf, K.-G. Faißt, "An Approach for a Model Based Development Process of Cybertronic Systems", *58th Ilmenau Scientific Colloqium,* Ilmenau, Germany, Sept. 2014

[3] P. Feiler, B. Lewis, S. Vestal, E. Colbert, "An Overview of the SAE Architecture Analysis & Design Language (AADL) Standard: A Basis for Model-Based Architecture-Driven Embedded Systems Engineering", *IFIP World Computer Congress*, 2004

[4] P. Solanti, R. Klein, "Model-Based Approach for Developing Optimal HW/SW Architectures for AI systems", *DVCon Europe 2023*, Munich, Germany, Oct. 2023

[5] J.-L.Voirin, "Model-based System and Architecture Engineering with the Arcadia Method", *ISTE Press,* London & Elsevier, Oxford, 2017

[6] P. Micouin, "Property-Model Methodology: A Model-Based Systems Engineering Approach Using VHDL-AMS", *Systems Engineering,* Vol. 17-3, pp. 249–263, 2014

[7] S. Solanti-Iltanen, B. Hall, P. Solanti, "Model-Based Cybertronics Systems Engineering (MBCSE)", *INCOSE International Symposium,* Dublin, Ireland, July 2024

[8] D. Chen, "Lightning Talk: The Next Wave of High-level Synthesis," *2023 60th ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, 2023, pp. 1-3, doi: 10.1109/DAC56929.2023.10247880.

[9] L. E. Pozzoni, F. Ferrandi, L. Mendola, A. A. Palazzo and F. Pappalardo, "Using High-Level Synthesis to model System Verilog procedural timing controls," *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Antwerp, Belgium, 2023, pp. 1-6, doi: 10.23919/DATE56975.2023.10136907.

[10] A. Takach, "High-Level Synthesis: Status, Trends, and Future Directions," *IEEE Design and Test*, April 1^{st}, 2016, Volume 33, pp. 116-124

[11] S. Han, H. Mao, W. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization, and Huffman Coding," May 2^{nd}, 2016, *International Conference on Learning Representations (ICLR 2016)*

[12] N. Burgess, J. Milanovic, N. Stephens, K. Monachopoulos, andD. Mansell, "Bfloat16 processing for neural networks," in *IEEE 26thSymposium on Computer Arithmetic (ARITH*), 2019

[13] G. Genkina "Posits, a new Kind of Number Improves the Math of AI", *IEEE Spectrum*, September 2022

[14] C. Coelho, et al. "Automatic deep heterogeneous quantization of Deep Neural Networks for ultra-low area, low-latency inference on the edge at particle colliders," *arVix:2006.10159v2*, 11/2020

[15] H. Cai, C. Gan, T Wang, Z. Zhang, S. Han, "Once-for-All: Train One Network and Specialize it for Efficient Deployment," *arVix:1908.09791v5*, 8/2019

6

Edge AI Acceleration for Critical Systems: from FPGA Hardware to CGRA Technology

Pietro Nannipieri, Luca Zulberti, Tommaso Pacini, Matteo Monopoli, Tommaso Bocchi, and Luca Fanucci

University of Pisa, Italy

Abstract

This work discusses advancements in edge AI processing for critical systems, focusing on satellites. We explore hardware solutions for real-time and autonomous data processing using Field Programmable Gate Arrays (FPGAs) and Coarse-Grained Reconfigurable Arrays (CGRAs). The research outlines critical systems' challenges, such as power constraints, radiation tolerance, and the need for reliable AI processing. The paper details the design and implementation of an HDL-based GPU system on an FPGA and the FPGA-AI framework that automates the design of AI accelerators on FPGA. Future research focuses on extending neural network support and exploring collaborations for tape-out opportunities of CGRA prototypes.

Keywords: edge AI, acceleration, space, satellite, critical systems, FPGA, CGRA.

6.1 Introduction

In the rapidly advancing landscape of technological innovation, Artificial Intelligence (AI) is proving to be a pivotal driver for enhancing system autonomy and performance across a wide range of critical applications. These include not only space-based systems, such as satellites used for Earth

DOI: 10.1201/9788743808862-6

observation and scientific research [1], but also autonomous vehicles, defence systems, healthcare technologies, and industrial automation. In these diverse fields, large volumes of complex data must be processed efficiently and in real-time to ensure timely decision-making, reliability, and operational success.

For instance, satellites carry various advanced sensors like cameras, radars, altimeters, scatterometers, and lidars, which produce huge volumes of data to be processed and analysed. Conventionally, these systems rely on limited onboard processing, where raw data is gathered in mass storage for subsequent downlink to ground stations for further analysis. In any event, with the increasing need for timely information related to Earth observation, scientific missions, and space exploration, there is an ever-growing requirement for real-time data processing [2]. This paradigm shift places significant pressure on the avionics subsystems to process and transmit larger quantities of data with minimal delay, necessitating a new approach to onboard computing.

Similarly, in other critical systems such as autonomous vehicles, military defence platforms, and medical devices, AI must function with extreme reliability, power efficiency, and real-time responsiveness [3]. Current solutions, such as data compression techniques or basic on-the-edge pre-processing, are far from giving an adequate answer to these complex demands. Besides, real-time AI processing in critical systems should be resistant to environmental challenges: whether it is radiation tolerance for space systems, power constraints for remote sensing equipment, or safety-critical computation in autonomous driving.

The need for hardware accelerators capable of executing AI algorithms with high efficiency, reliability, and adaptability is increasingly clear. Systems like Field Programmable Gate Arrays (FPGAs) [4], [5] and Coarse-Grained Reconfigurable Arrays (CGRAs) are emerging as leading candidates for such tasks. Because they can support parallel processing, adapt to changing workloads, and provide low-power yet high-performance computation, they are particularly suited to an environment where real-time responses and resilience to failure are paramount.

As industry agendas such as ESA's 'Space for a Green Future' vision for 2025 [6] look toward harnessing AI for environmental monitoring, disaster response, and real-time analytics, these hardware solutions will play a key role in advancing next-generation capabilities. Beyond space, applications such as autonomous navigation, object detection, and emergency response in smart cities will require similarly robust, power-efficient AI systems that can operate seamlessly in challenging environments.

Edge AI represents a promising solution for developing more autonomous and real-time systems, paving the way for innovative applications that can address the complex challenges of modern critical environments. As we look towards a future with more autonomous and capable critical systems, AI will play a pivotal role in transforming how we observe, interact with, and understand them. However, it is well known that AI algorithms are computationally intensive. Therefore, it is necessary to identify reliable hardware solutions capable of supporting the execution of those algorithms in critical application scenarios.

In the following sections, we will present three ongoing projects to meet this technological need. In the short term, we propose an FPGA-based approach with two possible solutions, the GPU@SAT project and the FPG-AI project, a tool flow to automatically generate Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) hardware accelerators for FPGAs. In the long term, we propose the CGR-AI project to develop a radiation-hardened AI heterogeneous processing platform relying on a reconfigurable CGRA core.

6.2 State of the Art

Edge AI has garnered considerable interest among researchers in the field of space applications, particularly for scenarios where algorithms requiring acceleration must balance high-performance demands with stringent resource constraints. The choice of technology is inherently application-specific and driven by unique requirements, leading to a diverse array of proposed solutions and hardware implementations in the literature.

FPGAs have emerged as energy-efficient platforms for executing artificial intelligence algorithms [4]. Their inherent hardware parallelism aligns well with the computational demands of AI workloads. However, leveraging FPGAs for neural network acceleration poses challenges, primarily due to the significant engineering effort required to design custom, optimized hardware accelerators. To streamline the development and experimentation of Deep Neural Networks (DNNs) on FPGAs, several automation toolflows have been introduced. Tools such as Microchip VectorBlox AI [7], AMD Xilinx Vitis AI [8], and the MATLAB Deep Learning HDL Toolbox [9] support a wide range of models, enabling users, even those without extensive FPGA expertise, to efficiently deploy and accelerate DNNs on these platforms.

Conversely, hardware platforms like Graphics Processing Units (GPUs), which dominate commercial machine learning (ML) applications, are rarely utilized in space missions. This is primarily due to their higher power consumption and vulnerability to Single Event Latchups (SELs). Despite the significant potential of deploying GPUs in space, particularly for Deep Learning (DL) algorithms, to our knowledge, there have been no suitable solutions specifically designed for satellites. Previous research [10], [11] has primarily focused on evaluating the feasibility of using Commercial Off-The-Shelf (COTS) GPUs, despite their inherent limitations for space applications. Similarly, neuromorphic processors are gaining traction as a promising alternative for space-based applications [12].

CGRAs offer a compelling compromise between flexibility and efficiency. By combining spatial and temporal computation, CGRAs optimize performance for specific AI workloads without necessitating task-specific hardware redesign, achieving near-ASIC-level performance with software-like reconfigurability. For instance, the work presented in [13] integrates ML with CGRA compilation, significantly enhancing their accessibility and efficiency for practical applications. Another noteworthy advancement is Eyeriss v2 [14], a state-of-the-art DNN accelerator designed to enhance throughput and energy efficiency for both large-scale models like AlexNet and compact models such as MobileNet.

6.3 FPG-AI: Automation Tool Flow for Efficient Deployment of Pre-trained CNN/RNN Models on FPGA Technology

FPGAs are currently the most suitable solution for bringing efficient and performant AI capabilities onboard space missions. The high degree of parallelism offered by FPGAs is compatible with AI algorithms, allowing for spatial pipelining and achieving high efficiency and performance [4]. Besides this, there are several commercial Radiation-Hardened by Design (RHBD) FPGA solutions, which also make them a good option for space applications. However, the development time for an AI algorithm on FPGAs is long since these platforms involve high design effort with required intensive optimization due to their narrow resource budgets.

This challenge has been partially addressed by developing automatic AI accelerator generation frameworks [15]. These tools facilitate accelerating DNNs on FPGAs by a wide variety of users without any expertise. That is important because it reduces the development time while maintaining high performance, hence making the use of FPGAs for AI in space missions possible.

FPG-AI [16] is a technology-independent toolflow for automating the acceleration of DNNs onboard FPGAs. It takes as input a pre-trained DNN model along with its application dataset. First, FPG-AI prepares the target network for hardware acceleration by performing optimizations on model topology and shifting arithmetic from Floating-point (FP) to Fixed-point (FXP). Then, a Design Space Exploration (DSE) process selects in the parameters space of the underlying architecture the point that meets the additional constraints provided by the user on application metrics, resource consumption, and performance. In this way, the generation of the target accelerator will be automated, and FPG-AI will be an end-to-end, ready-for-use tool while still keeping a high degree of customization concerning the user directives. Once the set of parameters is identified, DSE produces a configuration file for tuning the hardware architecture. The accelerator in the case of FPG-AI is the MDE, which can be a fully handcrafted HDL-based design that hosts no third-party IPs and can be customized according to the available resource primitives of FPGAs. Thanks to these properties, the MDE imposes no limit on device portability and allows, therefore, the implementation of components from different vendors with disparate resource budgets. Presently, this tool supports AMD XILINX, Microchip, Intel ALTERA, and NanoXplore FPGAs.

FPG-AI generates as output the HDL files describing an accelerator customized for the given model and device that meets the additional directives received by the user. Unlike other solutions, the toolflow provides the HDL sources of the accelerator and not the final bitstream, allowing the user to exploit the unused portion of the FPGA for complementary tasks. Figure 6.1 reports the block diagram of FPG-AI.

The FPG-AI framework is characterized by several key features that make it highly effective for deploying AI in space missions. One of the standout characteristics is technology independence, achieved through the MDE architecture. This solution's handmade HDL code without third-party IPs allows it to target multiple FPGA vendors, making the tool particularly suited for quickly supporting new devices.

Another significant feature is the high degree of scalability and customization of the hardware accelerator core, which allows users to set constraints on resource utilization (DSPs and on-chip-memories), inference time, and application metrics, tailoring the solution to the specific requirements of the space mission.

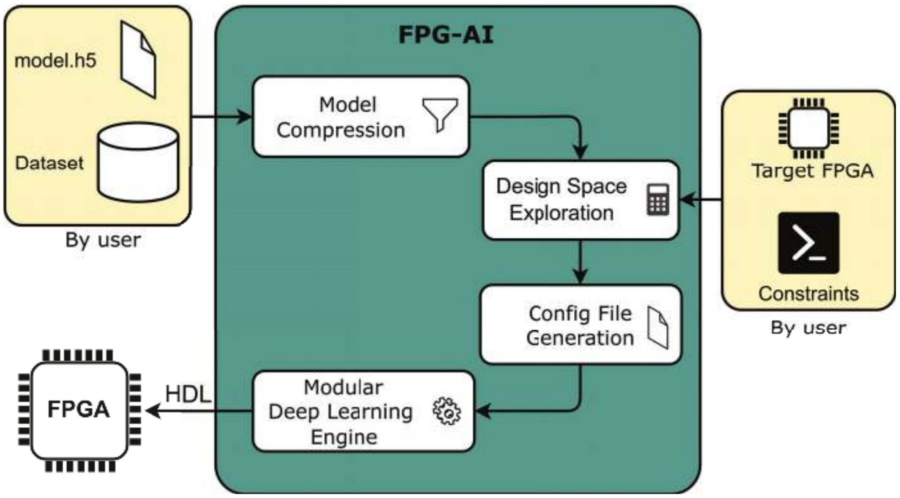


Figure 6.1 FPG-AI block diagram. ⏎

The ease of System-on-Chip (SoC) integration is guaranteed by the generation of an AXI Memory-Mapped accelerator which facilitates straightforward embedding into existing systems. In addition, FPG-AI's accelerators are completely autonomous during the computation of a neural network, eliminating the need to split the workload with the host CPU, improving efficiency and performance.

Currently, the FPG-AI solution supports Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), [17] broadening its applicability across different types of AI models. Combined with the aforementioned features, this versatility positions FPG-AI as a robust and adaptable choice for implementing AI in space applications, ensuring high performance and reliability in challenging environments.

6.3.1 Network-in-Network (NiN) case study

In the following, implementation results are reported for an image processing use case to exemplify the workflow of FPG-AI. The model selected for this study is Network-in-Network, a commonly used CNN model for image classification on the CIFAR10 dataset. Each image in CIFAR10 is a 32 × 32-pixel colour image, which means it has three colour channels (red, green, and blue) for each pixel. These images are divided into ten classes, with each class representing a different object or category. The NiN model involves three triples of Convolutional layers with padding set to "same". The

Convolutional layers are followed by a Global Average Pooling layer which provides predictions. Table 6.1 summarizes the features of the NiN.

As a first step, the pre-trained model undergoes the Model Compression process of FPG-AI to quantize the network and shift from FP to FXP arithmetic. Initially, pre-analyses are carried out to study the effects on model accuracy caused by different bit-width settings of each quantity within the network. In particular, the input dataset quantization pre-analysis evaluates the effects on the application metric depending on the input dataset bit-width and identifies the input dynamic range.

- Weights quantization pre-analysis evaluates the effects on application metrics depending on the weights bit-width and identifies the weights and biases dynamic ranges.
- Activations quantization pre-analysis: studies the effects on application metrics of different activation bit-widths by evaluating one layer at a time and identifies the activation dynamic range.

Table 6.1 Model summary of Network-in-Network. ⏎

Model Summary	
Model	NiN
Type	CNN
Dataset	CIFAR10
Input image dimension	$32 \times 32 \times 3$
# Convolutional layers	9
# Convolutional filters for each layer	192, 160, 96, 192, 192, 192, 192, 192, 10
Convolutional filters dimensions	5×5, 1×1, 3×3
# Pooling layers	3
Pooling filters dimensions	2×2, 8×8
Type of pooling	Max Pooling, Average Pooling
# Fully Connected layers	0
# Neurons for each FC layer	-
Total Parameters	969K
Memory for Parameters [Mbit]	3.69 Mb

After the pre-analysis, additional bit-true simulations are carried out to study the combined effects of quantized inputs, weights, and activations. The accuracy trend is analysed in a range across the optimal bit-widths provided by the pre-analyses. In this phase, different truncation settings are applied to modify activation bit-widths. The whole collection of simulation results is finally saved into a table that will be used during the DSE tool phase.The DSE then selects a quantized configuration among the ones generated by the Model Compression step. For this case study, we decided to extract the configuration with the best application metric, obtaining a final accuracy of 88.88%.Thereafter, the DSE exploits an iterative algorithm to best configure the parameters of the MDE architecture. Once the DSE algorithm converges to a solution, the toolflow runs an automation script that generates all the necessary files for the hardware deployment of the network. As a first step, the chosen quantization parameters and the obtained architectural parameters are written on the MDE configuration file. The latter completes the HDL description of the MDE architecture configured to optimally accelerate the given quantized model on the selected FPGA device. At the end of this automatic process, the user can finally synthesize and implement the accelerator using the vendor-specific tools associated with the chosen FPGA component. Table 6.2 reports the configuration at maximum parallelism for the NiN model on FPGAs from diverse vendors. The results have been obtained by using the following tools with default synthesis and place-and-route strategies: Vivado Design Suite for AMD Xilinx, Synplify Pro for Microsemi, and Quartus Prime for Intel.

Table 6.2 NiN implementation results for maximum parallelism configuration. ⏎

Device	**ZU7EV**	**RTPF500T**	**XQRKU060**	**10AX048**
LUT	56375 (24.5%)	63028 (13.1%)	105950 (31.9%)	26115 (14.2%)
FF	14432 (3.1%)	28386 (5.9%)	20012 (3.02%)	12819 (1.7%)
LUTRAM/μSRAM/MLAB	1104 (3.1%)	0(0%)	552 (0.38%)	96 (1.4%)
BRAM/LSRAM/M20K	245.5 (78.7%)	512(33.7%)	277.5 (25.7%)	525(36.7%)
URAM	5 (5.2%)	-	-	-
DSP	1210 (70%)	1214(82%)	2410 (87.3%)	1229 (89.8%)
MDE Frequency [MHz]	126.6	50.8	72.5	83.33
Inference Time [ms]	4.54	11.33	5.39	218.88
Power [W]	2.88	-	3.07	3.07
Accuracy [%]	88.88	88.88	88.88	88.88

These experiments demonstrate the adaptability of the proposed quantization methods and the scalability of the MDE architecture. The accelerated models have been deployed on FPGA devices from different vendors (Intel, Xilinx, Microsemi) and various FPGA families, resulting in diverse characteristics in terms of technology, available resources, and radiation tolerance. This level of adaptability sets FPG-AI apart from other automation tools in the existing literature.

6.4 GPU@SAT: RISC-V Based SoC Featuring a Soft-GPU Hardware Accelerator for Artificial Intelligence On-board

In recent years, General-Purpose Computing on Graphic Processing Units (GPGPU) [18] has been gaining popularity due to its high energy efficiency and computational capabilities. Although Graphic Processing Units were initially utilized only for graphic applications, GPGPU has since bridged the gap into the use of GPUs in a wide range of tasks, including image processing, cryptography, and Machine Learning [19]. GPGPU exploits frameworks such as OpenCL [20] and CUDA [21], which provide high-level programming interfaces, allowing for a significant reduction in development time. Compatibility with open-source tools, including TensorFlow, PyTorch, and Caffe, also supports AI applications. Yet, its performance could be on a par with those of custom hardware accelerators, while - because of the lack of flexibility and customizability - GPGPU is not capable of efficiently executing distinct families of algorithms. As such, combining the advantages of GPGPU with the reconfigurability features of FPGAs could represent a viable solution for the acceleration of a wide range of ML algorithms in applications with strict constraints on cost, size, and power consumption (e.g., in space systems). This approach could offer a cost-effective and scalable solution for space missions that require high-performance computing capabilities. In this sense, soft GPGPU holds the potential to revolutionize the field of space exploration by enabling more complex and data-intensive tasks. In space applications, special attention must be given to power consumption, resource efficiency, and fault-tolerant design. To reduce costs and enable the use of commercial FPGAs for Low Earth Orbit (LEO) missions, it is crucial to prioritize fault-tolerant solutions.

An AI acceleration system compliant with the strict requirements of both low and high-level space missions (e.g., fault-tolerance, resource utilization, power consumption), exploiting the existing GPGPU framework, as well as FPGA reconfigurability, is still missing. As such, we propose the implementation of an SoC featuring the GPU@SAT IP from IngeniArs S.r.l. [22], [23], [24], [25], which can execute OpenCL 1.2 kernels. This could prove to be an efficient solution not only for the acceleration of ML algorithms but also for the execution of any OpenCL kernel in various fields of application (e.g., image processing, consumer applications, cryptography), ensuring a higher degree of flexibility. Figure 6.2 illustrates the proposed base architecture for the SoC. The primary objectives of the project are to develop an AXI-compatible system incorporating the GPU@SAT IP and the open-source RISC-V architecture, with a strong emphasis on reliability—critical for the successful execution of high-level space missions. Additionally, we aim to characterize the system on various FPGA platforms, assessing performance-to-power efficiency metrics.

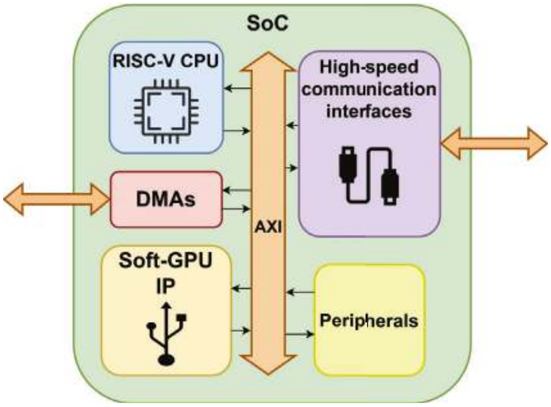


Figure 6.2 Overview of the System-on-Chip based on GPU@SAT. ⏎

6.4.1 Enhancing a soft GPU IP reliability against SEUs in space: Modelling approach and criticality analysis on a Radiation-Tolerant FPGA

To enhance the robustness of the GPU@SAT IP against Single Event Upsets (SEUs), we propose a detailed methodology revolving around modelling, fault criticality analysis, and a systematic application of fault mitigation techniques. Figure 6.3 shows a schematic overview of the GPU@SAT architecture and its main design modules.

We employ the Möbius [26] high-level modelling tool to estimate the reliability of the system. The tool supports both Fault Tree Models and Stochastic Activity Networks (SANs), but Fault Trees are used here due to their simplicity and alignment to assess fault impacts without accounting for repair mechanisms. The fault tree model allows the combination of basic events like failures of sub-modules using logical gates (AND, OR, XOR, etc.), making it possible to model how SEUs affect different components of the GPU architecture. Specifically, to quantify the SEU rates, we use data provided by Xilinx on the failure rates at Geosynchronous Earth Orbit (GEO) of the CRAM and Block RAMs in the target FPGA device, in our case the

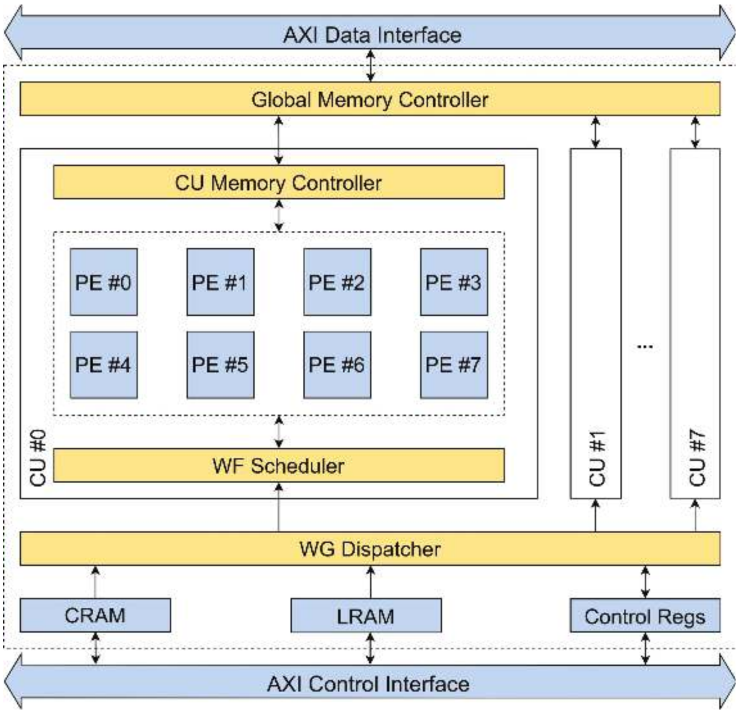


Figure 6.3 Overview of the GPU@SAT architecture. ⏎

Radiation-Tolerant Xilinx XQRKU060. These rates are applied to the various GPU components, estimating the worst-case SEU rate for each module based on the number of resources (e.g. LUTs, registers, BRAMs) used in each of them.

As SEU events can be modelled using a Poisson distribution [27], the firing time between them follows an exponential one. With this information, we modelled the entire GPU architecture to estimate a baseline reliability level using the Möbius tool. Table 6.3 presents the reliability estimation results for the GPU and its primary sub-components over 60 days.

The reliability analysis shows that the GPU has a lower reliability than its individual sub-components due to its high vulnerability to SEUs. The Global Memory Controller and CU Memory Controllers were identified as the least reliable parts, with high SEU sensitivity. Applying Triple Modular Redundancy (TMR) directly to these components would be too costly in terms of area and power, so partial redundancy techniques like Distributed TMR and Block TMR with voters are suggested. The results show that,

Table 6.3 Reliability values for the GPU and its components over a 60-day span. ⏎

Atomic Model	**Time-Averaged Reliability**
Workgroup Dispatcher	9.855E-01
Wavefront Scheduler	9.637E-01
Global Memory Controller	5.411E-01
RTM	9.752E-01
PE Array	6.372E-01
Control Interface	8.943E-01
CU Memory Controller	5.011E-01
GPU w/ 1 CU	2.421E-01

while no components exhibited extremely low reliability, strategic application of fault mitigation techniques can significantly increase system robustness without introducing prohibitive overhead.

Given the high resource consumption of a GPU, applying fault tolerance measures universally would be inefficient. Hence, we propose a classification-based methodology [28] that evaluates each component based on:

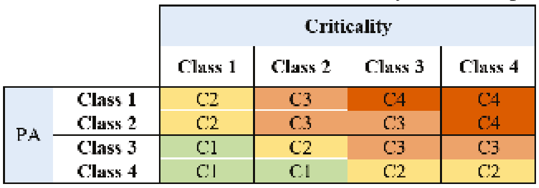
- **Criticality**: How essential the component is for correct operation.
- **Power/Area Impact**: How much power and area each component consumes on the FPGA.

Each component is classified into four criticality and power/area classes (C1-C4). The classification helps prioritize which components to protect more rigorously and which ones require less redundancy. The criticality of each component is assessed based on:

- **Fault Impact**: Evaluating both the spatial and temporal importance of the module in the GPU architecture.
- **SEU Sensitivity**: Based on the type of primitives (e.g., BRAM, registers, combinatorial logic) and their sensitivity to SEUs.
- **Component Type and Importance**: Importance is determined by how critical a module's function is for the correct operation of the GPU (e.g., if it's responsible for task dispatching or controlling memory access).
- **Power/Area (PA) Impact**: This accounts for how much area and dynamic power each component consumes on the FPGA. The analysis is based on post-place-and-route resource utilization and switching activity extracted from simulation benchmarks (e.g., matrix multiplication, convolution). Components that consume more resources are prioritized for less costly fault mitigation techniques to avoid excessive congestion and power consumption on the FPGA.

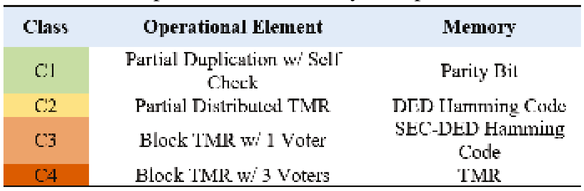
After analysing the criticality and PA impact, each component is assigned a suitable fault mitigation technique. These techniques are categorized into four classes, with higher classes corresponding to more impactful and resource-intensive solutions. Two distinct classification tables have been created to differentiate between operational and memory components, as these typically require the application of different mitigation techniques. Table 6.4 and Table 6.5 summarize the proposed techniques associated with different levels of criticality.

Table 6.4 Classification based on criticality, area, and power. ⏎



		Criticality			
		Class 1	Class 2	Class 3	Class 4
PA	Class 1	C2	C3	C4	C4
	Class 2	C2	C3	C3	C4
	Class 3	C1	C2	C3	C3
	Class 4	C1	C1	C2	C2

Table 6.5 Operational & Memory Components Classes. ⏎



Class	Operational Element	Memory
C1	Partial Duplication w/ Self Check	Parity Bit
C2	Partial Distributed TMR	DED Hamming Code
C3	Block TMR w/ 1 Voter	SEC-DED Hamming Code
C4	Block TMR w/ 3 Voters	TMR

6.5 CGR-AI: Innovative Coarse-Grained Reconfigurable Array Platform for Computing Artificial Intelligence On-Board

While FPGAs offer significant advantages for implementing AI in space, they have limitations. Performance and power efficiency are often constrained, and the cost per device, particularly for Radiation-Tolerant (RT) and Radiation-Hardened (RH) chips, can be substantial [29], [30]. To address these challenges, the long-term solution may lie in developing a specific hardware platform tailored for AI applications in space. Such a platform, built on RT/RH technology, could offer the throughput required by applications and the power efficiency required by the environment, justifying the large-scale adoption of AI in space missions [31].

However, maintaining a degree of flexibility is crucial. Designing an Application-Specific Integrated Circuit (ASIC) for a specific Neural Network (NN) accelerator is not sustainable due to the diverse and evolving nature of AI applications [32]. Instead, future hardware platforms should balance high performance with adaptability, supporting various AI models and the ability to update and reconfigure as mission requirements change. This approach would ensure that space AI systems remain robust, efficient, and versatile, capable of meeting the demanding conditions of space exploration and observation [33].

The purpose of the CGR-AI project is to develop an AI engine IP, Complementary to RISC-V/FPGA for AI workload data crunching, powered by a CGRA accelerator, to be implemented in std-cell technology. Such engine shall overcome existing solutions in terms of:

- Flexibility: the platform may be easily adapted to a given application by executing specific firmware to schedule memory transactions and load CGRA configurations.
- Time Predictability: the CGRA-based architecture performs operations in a fixed number of clock cycles. Given the global memory timing model, each AI Engine operation is completed within a deterministic time bound, ensuring algorithm execution for time-critical applications.
- Reliability: library and architectural solutions to withstand radiations typical of higher orbits. Furthermore, CGRA resources can be used to apply dynamic redundancy to the data path, exploiting dedicated hardware voters for error correction.

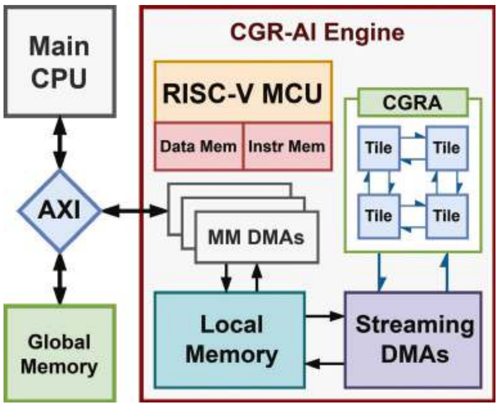


Figure 6.4 CGR-AI Engine block diagram. ⏎

Figure 6.4 depicts the proposed CGRA-based AI Engine architecture. The CGRA [34], [35] is an AI accelerator capable of performing linear and approximated nonlinear vector operations with dynamic element width. The RISC-V CPU runs the firmware loaded into the Tightly-Coupled Memories (TCMs) and overlooks all the operations within the architecture. The streaming Direct Memory Accesses (DMAs) provide custom pattern access to/from local memory, and the Memory-Mapped (MM) DMAs provide access to/from external memory. AXI bus is implemented to allow communication with multi-core application class CPU, commercial peripherals, and main memory/DDR controller. The AI Engine can parallelize common workloads in neural network algorithms, like convolutions, pooling, and activation functions. The spatial architecture makes the engine capable of distributing resources for different algorithms, enabling load balancing during different mission stages with dynamic requirements.

The project is currently in development, and we are in the process of setting up an FPGA prototype and characterising the processing core on std-cell technology. As preliminary results, in Table 6.6 we report synthesis data obtained on the 65nm technology for an operating frequency of 625 MHz.

The analysis is focused on the single Tile module since its results can be composed to form the desired matrix.

For each port configuration, we adapt the number of input and output ports of the PE accordingly to the number of Functional Units (FUs) implemented (third column), in order:

1. ambslx: all FUs are present.
2. ambsx: without LUT FU, which is the one with more utilization of resources.
3. ambs: without LUT and MUX FUs.
4. am: only with ADD and MUL (e.g., to perform MAC in convolutional layers)
5. lx: only with LUT and MUX (e.g., to execute activation functions)

We refer to (a) as throughput [GOps]; efficacy is reported in (b) as area efficiency [GOps/mm2], where the theoretical throughput of the configurations is divided by the related area and power metrics, and the best solutions are highlighted in bold, and (c) as energy efficiency [GOps/W], a direct measure of the energy needed for each operation: its inverse is W/GOps = W/(Gop/s) = J/Gop.

Table 6.6 Throughput compared to Area and Energy Efficiencies. ⏎

# In # Out	**FUs**	**625 MHz**		
		(a)	(b)	(c)
3 In 2 Out	ambslx	3.75	17.7	88
	ambsx	3.12	19.4	96.3
	ambs	2.50	22.2	111
	am	1.25	22.9	119
	lx	1.25	**27.4**	142
4 In 4 Out	ambslx	3.75	16.1	81
	ambsx	3.12	18.6	93.7
	ambs	2.50	21.4	110
	am	1.25	**26.5**	151
	lx	1.25	**24.8**	129
4 In 4 Out	ambslx	3.75	15.4	80.6
	ambsx	3.12	17.3	90
	ambs	2.50	18	94.9
	am	1.25	16.1	88.4
	lx	1.25	**22.7**	120

6.6 Discussion and Conclusion

The execution of AI algorithms in critical systems is becoming increasingly important as the demand for real-time data processing and autonomous decision-making grows. Satellites with advanced AI capabilities can enhance applications such as Earth observation, scientific research, and space exploration by providing timely insights and efficient data management. However, addressing the diverse requirements of performance, radiation tolerance, and cost requires various solutions.

The GPU@SAT approach represents an immediate answer for applications needing an SW-programmable system, whose reliability has been analysed and improved. Such a solution can be considered particularly interesting thanks to the support both from Rad-hard and rad-tolerant FPGAs, thus suitable for space applications.FPG-AI provides an immediate, flexible solution for these needs. The fact that it can afford high parallelism, ease of customization, and integration makes FPG-AI a very promising candidate for both present and near-future space missions. The radiation-hardened versions of FPGAs can therefore offer such resilience against the harsh conditions in space.However, these have their performance and power efficiency limitations, not to mention the high cost, hence the need for continued innovation. Overcoming these limitations will long-term be so crucial to sustain further advancements in AI across many critical applications. The development of a dedicated hardware platform like CGR-AI, therefore, emerges as a strong candidate for such a future direction. The objective is to develop a platform that integrates high performance and efficiency, state-of-the-art AI applications, flexibility in handling changing mission needs, and varied AI models.While it may be the case that GPU@SAT and FPG-AI can at present work as a robust and versatile solution for current AI needs onboard satellites, continuous research, and development into more specialized hardware platforms, such as CGR-AI, will be very important. These efforts will ensure that the potential of AI in space is fully realized, supporting more autonomous, efficient, and capable satellite systems that can meet the challenges of tomorrow's space exploration and Earth observation missions.

Acknowledgements

This work is partially supported by the European Space Agency (ESA) in the framework of the Open Space Innovation Platform (OSIP) co-funded research under the ESA contract No. 4000137503, 4000141235, and 4000141108. This study received funding from the European Union - Next-GenerationEU - National Recovery and Resilience Plan (NRRP) – MISSION 4 COMPONENT 2, INVESTMENT N. 1.1, CALL PRIN 2022 PNRR D.D. 1409 14-09-2022 – (OPERAND - a recOnfigurable Platform & framEwork foR Ai infereNce on the eDge) CUP N.I53D23006070001

References

[1] Izzo, Dario, et al. "Selected trends in artificial intelligence for space applications." *Artificial Intelligence for Space: AI4SPACE.* CRC Press, 2022. 21-52.

[2] Q. Li, S. Wang, X. Ma, A. Zhou, and F. Yang, "Towards Sustainable Satellite Edge Computing," arXiv (Cornell University), Sep. 2021, doi: https://doi.org/10.1109/edge53862.2021.00010.

[3] S. Ao, "Building Safe and Reliable AI Systems for Safety Critical Tasks with Vision-Language Processing," *Lecture notes in computer science*, pp. 423–428, Jan. 2023, doi: https://doi.org/10.1007/978-3-031-28241-6_47.

[4] T. Wang, C. Wang, X. Zhou, and H. Chen, "An Overview of FPGA Based Deep Learning Accelerators: Challenges and Opportunities," IEEE Xplore, Aug. 01, 2019. https://ieeexplore.ieee.org/abstract/document/8855594 (accessed Aug. 14, 2023).

[5] El and Mohsine Eleuldj, “Survey of Deep Learning Neural Networks Implementation on FPGAs,” Nov. 2020, doi: https://doi.org/10.1109/cloudtech49835.2020.9365911.

[6] “Space for a green future – ESA Vision.” https://vision.esa.int/space-for-a-green-future/

[7] “VectorBlox^{TM} Accelerator SDK,” *Microchip.com*, 2025. https://www.microchip.com/en-us/products/fpgas-and-plds/fpga-and-soc-design-tools/vectorblox

[8] “AMD Technical Information Portal,” *Amd.com*, 2024. https://docs.amd.com/r/en-US/ug1414-vitis-ai

[9] “Deep Learning HDL Toolbox,” *Mathworks.com*, 2025. https://it.mathworks.com/products/deep-learning-hdl.html

[10] W. Powell, M. Campola, T. Sheets, A. Davidson, and S. Welsh, “Commercial Off-The-Shelf GPU Qualification for Space Applications,” Sep. 2018.

[11] Leonidas Kosmidis, J. Lachaize, J. Abella, Olivier Notebaert, F. J. Cazorla, and D. Steenari, “GPU4S: Embedded GPUs in Space,” *UPCommons institutional repository (Universitat Politècnica de Catalunya)*, Aug. 2019, doi: https://doi.org/10.1109/dsd.2019.00064.

[12] D. Izzo, A. Hadjiivanov, D. Dold, G. Meoni, and E. Blazquez, “Neuromorphic Computing and Sensing in Space,” *CRC Press eBooks*, pp. 107–159, Nov. 2023, doi: https://doi.org/10.1201/9781003366386-4.

[13] Y. Luo *et al.*, “ML-CGRA: An Integrated Compilation Framework to Enable Efficient Machine Learning Acceleration on CGRAs,” pp. 1–6, Jul. 2023, doi: https://doi.org/10.1109/dac56929.2023.10247873.

[14] Y.-H. Chen, T.-J. Yang, J. S. Emer, and V. Sze, “Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, Jun. 2019, doi: https://doi.org/10.1109/jetcas.2019.2910232.

[15] S. I. Venieris, A. Kouris, and Christos-Savvas Bouganis, “Toolflows for Mapping Convolutional Neural Networks on FPGAs,” ACM Computing Surveys, vol. 51, no. 3, pp. 1–39, Jun. 2018, doi: https://doi.org/10.1145/3186332.

[16] T. Pacini, E. Rapuano, and L. Fanucci, “FPG-AI: A Technology-Independent Framework for the Automation of CNN Deployment on FPGAs,” IEEE Access, vol. 11, pp. 32759–32775, 2023, doi: https://doi.org/10.1109/ACCESS.2023.3263392.

[17] T. Pacini, E. Rapuano, L. Tuttobene, Pietro Nannipieri, L. Fanucci, and S. Moranti, “Towards the Extension of FPG-AI Toolflow to RNN Deployment on FPGAs for On-board Satellite Applications,” Oct. 2023, doi: https://doi.org/10.23919/edhpc59100.2023.10396607.

[18] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Phillips, “GPU Computing Graphics Processing UnitsVpowerful, programmable, and highly parallelVare increasingly targeting general-purpose computing applications,” doi: https://doi.org/10.1109/JPROC.2008.917757.

[19] J. D. Owens *et al.*, “A Survey of General-Purpose Computation on Graphics Hardware,” *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, Mar. 2007, doi: https://doi.org/10.1111/j.1467-8659.2007.01012.x.

[20] A. Munshi, “The OpenCL specification,” Aug. 2009, doi: https://doi.org/10.1109/hotchips.2009.7478342.

[21] Kirk, David. “NVIDIA CUDA software and GPU parallel computing architecture.” ISMM. Vol. 7. 2007.

[22] M. Monopoli, Luca Zulberti, G. Todaro, Pietro Nannipieri, and L. Fanucci, “Exploiting FPGA Dynamic Partial Reconfiguration for a Soft GPU-based System-on-Chip,” Jun. 2023, doi: https://doi.org/10.1109/prime58259.2023.10161859.

[23] G. Todaro, M. Monopoli, G. Benelli, Luca Zulberti, and T. Pacini, “Enhanced Soft GPU Architecture for FPGAs,” Jun. 2023, doi: https://doi.org/10.1109/prime58259.2023.10161749.

[24] M. Monopoli, Luca Zulberti, Pietro Nannipieri, L. Fanucci, and S. Moranti, “Exploring Key Aspects of Soft GPGPU Computing for On-board Acceleration of Artificial Intelligence Algorithms in Space Applications,” *vol. 28, pp. 1–6, Oct. 2023,* doi: https://doi.org/10.23919/edhpc59100.2023.10396624.

[25] G. Benelli, G. Giuffrida, R. Ciardi, D. Davalle, G. Todaro, and L. Fanucci, “GPU@SAT, the AI enabling ecosystem for on-board satellite applications,” vol. 19, pp. 1–4, Oct. 2023, doi: https://doi.org/10.23919/edhpc59100.2023.10396289.

[26] G. Clark *et al.*, “The Mobius modeling tool,” Nov. 2002, doi: https://doi.org/10.1109/pnpm.2001.953373.

[27] Gercek, Gokhan. “A method to compute SEU fault probabilities in memory arrays with error correction.” Dual-Use Space Technology Transfer Conference and Exhibition, Volume 2. 1994.

[28] Monopoli, M., Biondi, M., Todaro, G., Nannipieri, P., Moranti, S., Bernardeschi, C., & Fanucci, L. (2024). *Enhancing a Soft GPU IP*

Reliability Against SEUs in Space: Modelling Approach and Criticality Analysis on a Radiation-Tolerant FPGA. https://doi.org/10.36227/techrxiv.173273657.76155928/v1

[29] E. Ibe, H. Taniguchi, Yasuo Yahagi, Kenichi Shimbo, and T. Toba, "Impact of Scaling on Neutron-Induced Soft Error in SRAMs From a 250 nm to a 22 nm Design Rule," vol. 57, no. 7, pp. 1527–1538, May 2010, doi: https://doi.org/10.1109/ted.2010.2047907.

[30] T. Li, H. Liu, and H. Yang, "Design and Characterization of SEU Hardened Circuits for SRAM-Based FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 6, pp. 1276–1283, Jun. 2019, doi: https://doi.org/10.1109/tvlsi.2019.2892838.

[31] G. Furano *et al.*, "Towards the Use of Artificial Intelligence on the Edge in Space Systems: Challenges and Opportunities," *IEEE Aerospace and Electronic Systems Magazine*, vol. 35, no. 12, pp. 44–56, Dec. 2020, doi: https://doi.org/10.1109/MAES.2020.3008468.

[32] Y. Duan, J. S. Edwards, and Y. K. Dwivedi, "Artificial Intelligence for Decision Making in the Era of Big Data – evolution, Challenges and Research Agenda," *International Journal of Information Management*, vol. 48, no. 1, pp. 63–71, Oct. 2019, doi: https://doi.org/10.1016/j.ijinfomgt.2019.01.021.

[33] L. Mandrake *et al.*, "Space Applications of a Trusted AI Framework: Experiences and Lessons Learned," *IEEE Xplore*, Mar. 01, 2022. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9843322

[34] Luca Zulberti, M. Monopoli, Pietro Nannipieri, L. Fanucci, and S. Moranti, "Highly Parameterised CGRA Architecture for Design Space Exploration of Machine Learning Applications Onboard Satellites," *INDIGO (University of Illinois at Chicago)*, pp. 1–6, Oct. 2023, doi: https://doi.org/10.23919/edhpc59100.2023.10396632.

[35] Luca Zulberti, Monopoli, M., Pietro Nannipieri, Moranti, S., Thys, G., & Fanucci, L. (2025). Efficient Coarse-Grained Reconfigurable Array architecture for machine learning applications in space using DARE65T library platform. *Microprocessors and Microsystems*, 105142–105142. https://doi.org/10.1016/j.micpro.2025.105142

7

Model Selection and Prompting Strategies in Resource Constrained Environments for LLM-based Robotic System

Toms Eduards Zinars, Oskars Vismanis, Peteris Racinskis, Janis Arents, and Modris Greitans

Institute of Electronics and Computer Science (EDI), Latvia

Abstract

Large Language Models (LLMs) are increasingly becoming adopted for various applications in information processing and content generation. Coupled with the diverse availability of model weights, quantization, and fine-tuning, it is desirable to find the best-performing LLM within a certain memory budget. Prompting strategy plays a similarly major role in the quality of generation, with various methods existing that attempt to induce desired behaviour, requiring a major time investment to develop. As new models with better performance to memory ratio become available, it may be tempting to implement them into an existing system for potential performance improvements. In this work we explore the process of changing weights and note that weights from larger models or of different quantization precision are unable to replace the original model without modifications to the prompt contents, which in turn implies complications in developing modular, weight agnostic systems.

Keywords: large language models (LLM), natural language processing (NLP), high-level planning (HLP), robotics.

DOI: 10.1201/9788743808862-7

7.1 Introduction and Background

The interest to employ Large Language Models for various tasks has grown over the past couple of years [1]. Among these are applications in robotic systems, including tasks such as high-level planning [2], low-level control [3] and semantic map creation [4], but most of these approaches rely on top-of-the-line LLMs provided as a cloud service [5]. To increase the autonomy of such systems, the inference process must be brought as close as possible to the physical robotic agent. On-device LLMs for smartphones have received official support from Google with Gemini Nano 2 for Android [6] and from Apple with Apple Intelligence for MacOS ecosystem [7], a major step for on the edge deployment.

Other LLM developers have published their model weights with relatively open licenses, though such models are typically less capable than the ones offered exclusively through the cloud [8]. One of the main characteristics of an LLM is its size, which is measured in billions of parameters ("LLM 7B" denotes a model with 7 billion parameters). Currently, graphics cards (GPU) serve as the primary means of deploying these models, as they offer faster inference speeds than processor (CPU) based solutions [9]. Because of the standard 16-bit precision used for LLM weights even a relatively small model of 7B parameters (GPT-3, the "first" LLM, had 175B [10]) requires at least 14GB+ of VRAM, which currently is only available on the upper end of consumer grade GPUs [11]. Various Post-Training Quantization (PTQ) methods are available that can successfully reduce the precision down to 4-bits per weight while still retaining most of the performance [12], crucial for deploying as close to the edge as possible.

The autoregressive nature of decoder-only transformer architecture that underpins most LLMs [1] means that they generate the next (sub-)word based on all the input text before it according to the defined sampling parameters. As such the contents that are fed into the LLM are of high importance and has led to the emergence of the discipline of "Prompt Engineering" - the process of developing the textual content of the prompt to invoke the desired behaviour in the generation. Determining a prompting strategy for a particular task includes not only deciding on the contents of the prompt but also considerations such as if the task would be better executed over several separate prompts, a method known as "prompt chaining" [13], and what patterns [14] and methods [15] to employ for each prompt.

Creating an LLM-based high-level planner in a local context for a robotic system places several limitations on model selection which in turn heavily influences the prompting strategy. The ideal solution would provide a valid plan in the fastest time within the smallest amount of VRAM spending least amount of energy. As response times are a major consideration, even though GPUs are an additional power consuming component, currently it is not viable to create the system without one, and even the smallest models can strain and quickly deplete batteries of purely edge systems like smartphones [16]. Because the context window (the maximum amount of text a model can process at once) also contributes to memory usage, it may be necessary to reduce it, which in turn limits the number of examples that can be given and how complex of an instruction can be provided.

In this work we test how practical it is to change the underlying model of an already developed LLM-based system as a way to gain improvements. To this end we take our Natural Language Processing-High Level Planning (NLP-HLP) module for a mobile manipulator system [17] that uses an LLM for inference and replace it with different quantization precision weights and from different model families to see if the prompts carry over between different weights. A brief overview of LLMs, prompting strategies and LLM use in robotics is given in chapter 1.2. We create a validation set for each of our prompting steps and compare how well each set of weights processes these test questions, with chapter 1.3 elaborating on the experimental setup. In chapter 1.4 we display the results from our tests that were carried out on the set of different precisions and on the set of different models. In chapter 1.5 we give our conclusions and outlook for further development.

7.2 Related Work

LLM-based systems, regardless of use case, are built on top of various prompting strategies, which in turn rely on quality LLM weights to perform the generation. The landscape of LLMs has developed rapidly since early 2023, when “LLaMA” model weights were leaked to the public [18], beginning a wave of interest in local deployment of LLMs that as of October 2024 has resulted in over 140 thousand “Text Generation” weights of various model families, finetunes and their quantizations being uploaded to the public Hugging Face model library [19].

7.2.1 Local Large Language Models

Large Language Models as a distinct category emerged after the release of GPT-3 [10]. The initial focus was placed on training ever larger models

[20], as parameter count correlates with both performance and the concept of "emergent abilities" - at a certain scale, models become able to perform tasks that with fewer parameters they could not, thought [21] has questioned how "emergent" these abilities really are. As larger models are more resource intensive to both train and run, to the benefit of local deployment the focus shifted away from scale of the model to the scale and quality of training, such as the focus of the compact "Phi" model series [22]. The increase of scale can be demonstrated by comparing GPT-3's training of 300 billion tokens versus the training of "Llama 3" series models of up to 15 trillion [23] and "Qwen2.5" with 18 trillion [24], with both model families providing weight sizes that can be comfortably deployed locally. Besides requiring more memory, larger weights tend to be slower, which is a problem that "mixture of experts" (MoE) models try to overcome by mixing the knowledge of a larger with the speed of a smaller model, as done with [25]. This approach has limited relevancy for use in edge computing applications, as the memory requires all the parameters to be loaded in memory while actively only using a smaller portion of them at any one moment.

To truly place these models on the edge, quantization of weight precision is necessary to maximally reduce the memory requirements while still maintaining most of the performance [12]. Several methods for this process have emerged with often different priorities, but two common inference and quantization frameworks are the speed oriented, GPU-only ExLlamaV2 [26] with its EXL2 format and the more general llama.cpp [27] with GGUF format weights that also heavily invest in CPU-only and hybrid inference set ups.

7.2.2 Prompting

Prompting is the primary means of interacting with the LLM as the textual content is the basis upon which new information is generated. Developing a prompt that induces the desired behaviour is a sizable part of deploying these LLM-based systems. A variety of methods have been developed by the LLM research and open-source community to improve the generation quality [15]. Two noteworthy aspects that often get explored include determining what kind information should go into a prompt and how much LLM calls per an input should be done.

A common approach is known as "few-shot" or "n-shot" prompting [10], where "n" designates how many examples are used in the prompt to condition generation towards a particular output format. Another approach is to have the model first generate additional information about the input, such as the

"Chain-of-Thought" ("CoT") [28] method, where each example includes an intermediate reasoning step that is generated before the final answer. Various prompting patterns are highlighted in [14] which can be used to invoke specific behaviour in the models, such as having the model be the one asking questions.

LLMs generate their answers auto-regressively one new token at a time, with all the previous tokens affecting the probability of the next. If the model gets distracted and begins generating "incorrect" data, it is unlikely to recover [29]. Various methods attempt to mitigate this by generating several outputs, like it is done with "Self-Consistency" [30], which picks the most common answer amongst the outputs. "Least-to-Most" [31] prompting seeks to decompose a problem into subproblems and solves them sequentially. "Tree-of-Thought" [32] for each decomposition step generates several possible paths that are explored with search algorithms.

"Chain prompting" [13] is another fundamental skill utilized by many methods and systems, as it allows creating complex interactions by breaking up any task into several specialized prompts, which also allows creation of more modular systems.

7.2.3 LLM in Robotics

In general, the versatility of LLMs has been explored in wildly differing robotics roles. An implementation sceptical of LLMs ability to plan was explored in [33], where the model was used to translate natural language inputs into the structured planning language, which is then passed to a traditional planner. LLM as a planner was integrated into [2], where it generates the whole plan which is then verified for geometric feasibility and performs replanning if necessary. Implementation from [34] uses the LLM to rate a list of predefined actions on their probability as the next action needed for plan completion, while [35] uses an LLM to provide closed loop feedback (including dialog). LLMs have also been used to generate code directly for robot control, as shown in [36], while [3] has the LLM generate rewards for low-level control functions. LLMs have also been used for mapping, such as iteratively searching over a graph as presented in [37] or as in [4] where the LLM is used to embed semantic information inside the map itself, making it searchable through language.

All the examples provided above rely on using some of the largest models available at the time, which helps showcase the performance of the method. However, in order to create an autonomous robotic system that doesn't rely on a constant internet connection, local LLM application, especially of the smaller sized models, is of interest. Methods that attempt to rely on a smaller LLM as the primary inference engine are less known. In [38] a LLM is combined with a vision encoder to detect and reason about manipulation task failures, while [39] uses the LLM to guide the learning process of new long-horizon tasks. In [40] an LLM is one of the two main models tested for generation planning steps that are then evaluated based on feasibility and reward, whereas [41] notes the lower performance of open LLMs, but still invest in fine-tuning one to act as communication and planning manager for cooperative agents.

These methods rely on what now would be considered outdated models, given the rapid development of openly distributed LLMs over 2023 and 2024 [42], so it seems as if switching to newer models should provide an improvement in performance and make these smaller weights more enticing to use. However, a big part of performance comes from the contents of the prompt which are usually developed to maximize the quality of results out of specific weights. If the LLM-based system consists of a lot of complicated prompts and the new model does not immediately outperform the old, it may require practically rebuilding the whole system. Therefore, we want to explore how well prompts developed for one model perform on different models and quantization precisions.

7.3 Experimental Setup

After a model is chosen, a lot of time is invested in developing the prompt contents such that they invoke the desired generation. Prompting performance is typically framed as being mainly affected by model size and the model training data from which the various abilities are believed to emerge from [20], [28]. As new models with better performance to memory ratios emerge it can be worth exploring them as a potential avenue of upgrade for the system. Another avenue might be to increase the available memory and move to a higher precision quantization, which in theory should also result in improvements [12]. However, this would mainly be practical if the prompt contents can be retained for the new weights. If the validation shows no improvements, it then implies that in certain applications model weights are not “hot swappable”, as the prompt contents need to be rewritten which reduces the systems overall modularity.

7.3.1 System Description

The experiment is based on a previously developed HLP system, which utilizes *Llama 3 8B Instruct Q4_K_M* [43] weights, so the prompts have been developed in tandem with these specific weights. The HLP is designed to fulfil the planning phase for a mobile manipulator system (MMS) performing actions in a vaguely structured indoor environment. In the MMS's low-level planner, a set of action primitives is defined, that describes the system's abilities in a general manner or as highly adaptive functions [17]. The user inputs a NL command that is sent to the HLP. The HLP then generates a plan based on the MMS's set of abilities. This system consists of a prompt chain with 4 steps split into 3 stages: Filtering ("categorization" and "mobility"), Structuring ("preplanning") and Planning ("planning"):

- The "categorization" step determines the type of the incoming request between 4 categories: "mobile manipulation", "question answering", "casual discussion", "any other query". The rest of the system is only interested in "mobile manipulation". The prompt is 4-shot – containing examples for all categories.
- Because the currently used demonstrator is a static industrial arm, the "mobility" step is used to determine in a "True/False" fashion if moving between working positions is required or not. Once the demonstrator is

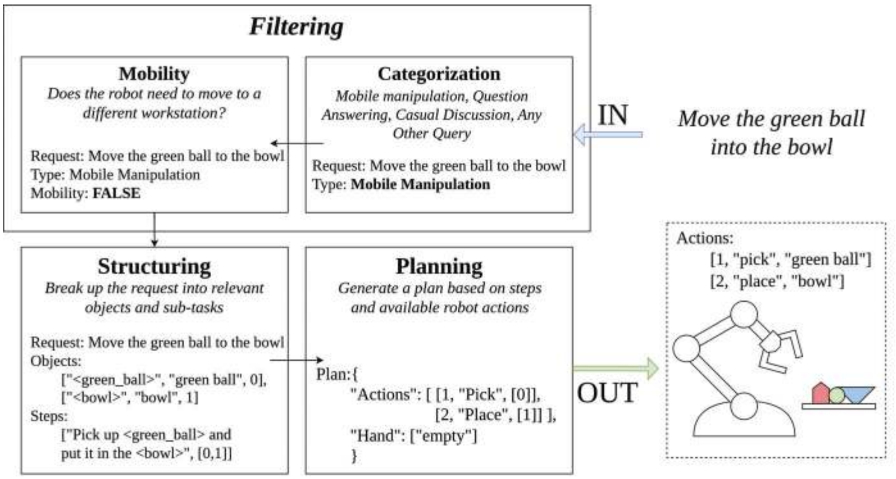

Filtering
Mobility
Does the robot need to move to a different workstation?
Request: Move the green ball to the bowl
Type: Mobile Manipulation
Mobility: FALSE
Categorization
Mobile manipulation, Question Answering, Casual Discussion, Any Other Query
Request: Move the green ball to the bowl
Type: Mobile Manipulation
IN
Move the green ball into the bowl
Structuring
Break up the request into relevant objects and sub-tasks
Request: Move the green ball to the bowl
Objects:
["<green_ball>", "green ball", 0],
["<bowl>", "bowl", 1]
Steps:
["Pick up <green_ball> and put it in the <bowl>", [0,1]]
Planning
Generate a plan based on steps and available robot actions
Plan:{
"Actions": [[1, "Pick", [0]], [2, "Place", [1]]],
"Hand": ["empty"]
}
OUT
Actions:
[1, "pick", "green ball"]
[2, "place", "bowl"]

Figure 7.1 Scematic of the MMS first demonstrator's HLP system.

transferred to a mobile platform, this step can simply be taken out. This prompt is zero-shot and is the shortest of the ones tested.

- The "preplanning" performs two tasks: creating a description (consisting of a marker, original text and a unique ID) for each object relevant to the request, formatted as "*["<object_name>", "object name", 0]*"; breaking up the request into smaller sub-tasks that can be planned for individually - "*["Pick up <large_cactus>", [1]], ["Place <large_cactus> on <small_tile>", [0]]*". The prompt uses 3 examples of increasing difficulty.
- The "planning" step creates a plan for each subtask that consists of assigning relevant action primitives to respective objects in a reasonable order. The input consists of the sub-task text, a list of objects and the status of the hand (*["empty"]* or *["<obj>", "obj, 0]"*). An input "*put the <box> on the <table>*" should output a plan like "*[["pick", [0]], ["place", [1]]]*". This is the most complex task of the set as it requires some form of reasoning, which smaller models are much less capable of [28]. This step employs CoT style step-by-step generation of step-by-step plan, where it is asked to first restate the task, check its hand status and then generate the step: "*1. Pick up <obj>. My Hand: <obj>*". It also relies on 3 examples with increasing difficulty.

By defining standardized input and output formats for each step, it is possible to integrate them together in a prompt chain in such a way that the intermediate product can be processed by traditional programming. This also allows the creation of a testing set for each step that is necessary to both validate and track the performance of the weights and prompts, though an alternative for tasks that are hard to standardize is to use another LLM to act as a judge [44].

7.3.2 Testing process

The validation process rates the generated output as follows: if the output could not be successfully processed by the script functions, then it is rated as "-1"; if the output could be processed but was wrong or unexpected it is rated as a "0"; if the output was processed and aligned with the expected result, it is rated with "1" and considered acceptable. Scoring only takes into consideration expected results and no malus is applied for failed generations. The validation set was created specifically for this test, however it is relatively simple and small.

The initial validation set for all tasks consist of 25 test inputs for each output type. The “categorization” set consists of 25 requests for each of the categories, to test how well the model can identify the correct category. “Mobility” consists of separate sets for “True” and “False”. Both “preplanning” and “planning” consist of 25 each. “Preplanning” is tested on “objects” (if the right objects were generated), “object length” (if no extra objects were created) and “Steps” (if the expected number of sub-tasks were generated). “Planning” is evaluated on “Plan” (if the expected plan was generated) and “Hand” (if the correct status of the hand was registered). In total 11 positions are evaluated across the current implementation of the system, for a total of 275 points.

7.3.3 Model selection

A selection of different models using Q4_K_M GGUF quantization and alternate quantization precisions for the “Llama 3 8B Instruct” model are put through the validation system to see if there is transferability of the designed prompts in an application with strictly defined input/output formats.

GGUF quantization method was chosen for its versatility to be deployed for CPU, GPU or hybrid inference, however only GPU inference has been used for the development and testing of the system. The chosen precisions are: “Q8_0”, “Q6_K”, “Q5_K_M”, “Q5_K_S”, “Q3_K_L”, “IQ3_XXS”, “IQ2_M” and “IQ2_XXS” [43], [45] , which roughly cover the range of available precisions.

The choice of models for testing was based on their size (able to fit into a single consumer-grade GPU) and their status as weights released directly by the developers. Third party finetunes were not considered for testing. Overall, the selected models represent a size range from 1B to 15B parameters.

Several benchmarks exist that attempt to quantify and compare the performance of different LLM models. The Open LLM Leaderboard [47] combines 6 such benchmarks to provide thorough evaluation of open-source models, while Berkeley Function Calling Leaderboard (BFCL) [48] specializes in testing model ability on tool use or function calling, which is a skillset comparable to the tasks expected of the planner. A different kind of benchmark is Chatbot Arena [8], which lets users compare two models and rate which they believe is better, but as such can be susceptible to user preference, “Style Control” is used to help mitigate it. These sorts of benchmarks play a role in informing decisions about which models to choose and create expectations about the model relative performance.

7.3.4 Testing environment

The testing is performed on *Ubuntu 22.04* using an *Nvidia GTX 3060 12GB* graphics card, running *CUDA version 12.3* and *Python 3.11.5*. The models were loaded using llama.cpp [27] with the *llama-cpp-python* [53] wrapper, which uses GGUF format weights and serves as a quantization method. The used maximum context window is fixed at 2048 tokens across all models. Sampling parameters include setting "temperature" to "0.0" (greedy sampling for most deterministic generation), which also removes random seeds as a factor in sampling. Despite this, tests were run at least 3 times with different seeds to verify no change between generations.

7.4 Results

With "Llama 3 8B Inst Q4_K_M" serving as the base line, two sets of weights were tested against it. The first set was used to determine if it is possible to shift the system to a different quantization precision either to free up memory by going to a lower precision or to improve performance by going to a higher one. The second test looked at the viability of replacing the LLM with a different model series all together. At temperature "0.0" no changes were noticed between the generations in either batch.

7.4.1 Differences between quantization precisions

Results from the first test shown in figure 7.2 visualize how 3- and 2-bit precisions rapidly degrade in quality, which aligns with general understanding of their behaviour [12], as smaller models incur more loss at lower precision. 4-bit and higher precisions however underperform expectations as none of them manage to noticeably improve upon the default weights – Q4_K_S (smaller than default) match results (245 points), but both models make different mistakes. The larger "Q6_K" and "Q8_0" weights score lower (237 and 242 respectively), while in theory these models should maintain more of the original non-quantized performance. "Q4_K_M", "Q4_K_S" and "Q5_K_M" were the only weights to not suffer from any completely failed generations.

Comparing the specific outputs of "Q4_K_S" and "Q4_K_M" for "mobility" tests, both models make 3 mistakes, but only 1 is different between them. The questions that are failed by both are "Find the basketball" for "False" and "Move the chairs" for "True", which might point to semantic ambiguity

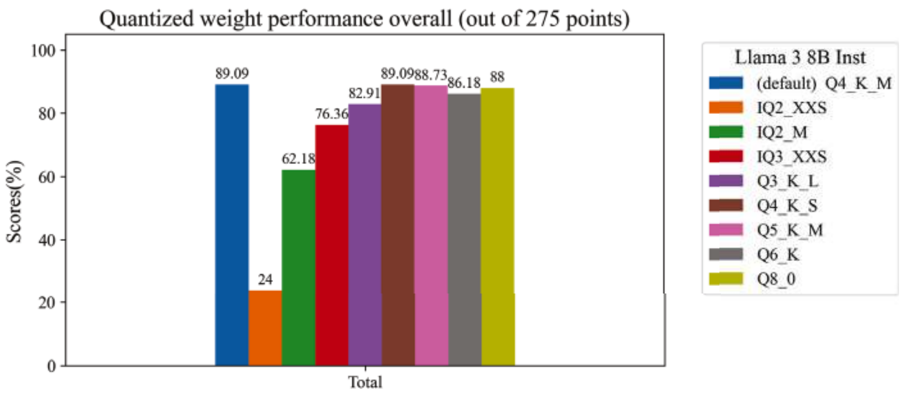


Figure 7.2 Correctly passed tests for quantization precision comparison. ⏎

in the validation set. "Q4_K_M" failed on "Flip the book to the next page" for "False" while "Q4_K_S" failed on "Carry this chocolate cake for me" for "True". Such minor differences can be a cause of concern, as it can mean that a new model could pass the validation test but then fail on tasks the previous model had no problem doing. This implies the need for prompting methods that can minimize such differences as well as potentially maintain a history of previously performed actions that a promising model can be further tested on.

7.4.2 Differences between models

During the second test, as seen in figure 7.3, none of the tested models were able to outperform the default model, despite several of them being larger in size and having a higher score on the Open LLM and BFCL benchmarks (Table 7.1). The further development of Llama 3 - "Llama 3.1 8B Instruct" - seems to perform about as well as "Gemma 2 9B instruct", while "Qwen2.5 14B Instruct" takes second place behind the default model. Model size plays a noticeable role as the larger models performed better, though not to the degree it would be expected. The prompting template also did not seem to factor in, considering that "Gemma 2" uses a "user/model" style prompting versus "Llama 3" that uses "system/user/model", which effects how prompts are structured. Overall, a similar conclusion can be drawn to what was seen with the quantization precision – prompts are unlikely to carry over to smaller models, but larger models do not guarantee superior results despite technically better performance.

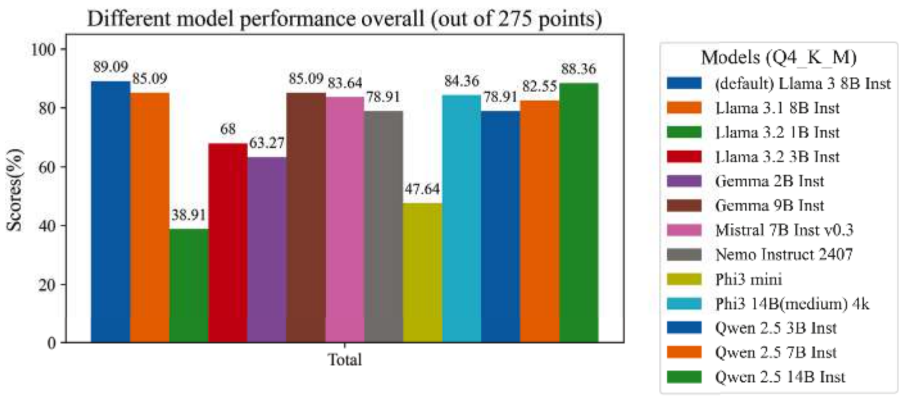


Figure 7.3 Correctly passed tests by various LLMs. ⏎

Table 7.1 Model Selection (all models using GGUF type Q4_K_M, all weights used in tests sourced from **[46]** ⏎

Model Name	**Parameters (in Billions)**	**OpenLLM score [47]**	**BFCL Overall Acc [48]**	**Chatbot Arena score (Style Controlled)** [8]
Llama 3 8B Instruct [23]	8.03	23.91	34.32	1143
Llama 3.1 8B Instruct [23]	8.03	27.77	50.87	1133
Llama 3.2 1B Instruct [49]	1.24	13.76	20.59	1045
Llama 3.2 3B Instruct [49]	3.21	23.85	46.92	1094
Gemma 2 2B Instruct [50]	2.61	17.05	22.38	1116
Gemma 2 9B Instruct [50]	9.24	28.86	51.59	1179
Mistral 7B Instruct v0.3 [51]	7.25	19.11	-	-
Nemo-Instruct-2407 [52]	12.2	23.53	42.56*	-
Phi 3.5 mini instruct [22]	3.82	27.4	-	-
Phi 3 medium 4K instruct [22]	14.0	32.67	-	1118
Qwen 2.5 3B Instruct [24]	3.09	21.03	47.09	-
Qwen 2.5 7B Instruct [24]	7.62	26.87	53.69	-
Qwen 2.5 14B Instruct [24]	14.8	32.11	57.68	-

* - value taken from “Open-Mistral-Nemo-2407”

Looking at individual categories, certain model performance can vary between different tasks drastically. As seen in figure 7.4, “Gemma 2 9B Instruct” has very strong results when it comes to identifying the status of the robot’s hand after the conclusion of the plan but has some of the weakest results when it comes to the actual planning. Further investigation reveals that the model struggles with placing the correct item IDs in the respective actions, a critical failure with no discernible cause.

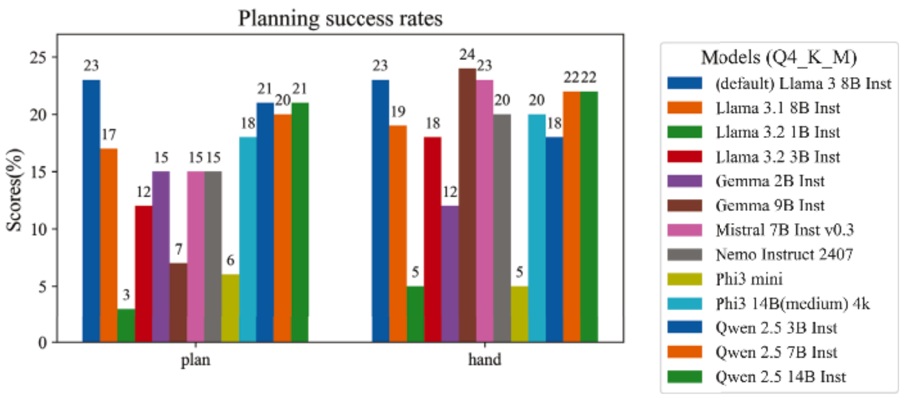


Figure 7.4 Planning success rates for the various models. ⏎

7.4.3 Result comparison to VRAM usage

As the available memory budget is an important metric to consider in the choice of models, the maximum amount of VRAM used during testing was noted for all weights tested and is relative to the model size in parameters. VRAM usage peaks during the "plan" step, as it has the longest instruction prompt.

Figure 7.5 shows how the larger models that require more memory seem to plateau and perform on par with the middle of road models, meanwhile the performance of the sub-7B models is more scattered, where the largest of this group ("Phi3.5 mini Q4_K_M) shows the second worst results and is outperformed by the almost twice as small "Gemma 2B Inst Q4_K_M". Its benchmark scores seen in Table 1.1 do not correlate well with real results seen in figure 7.3 and figure 7.4. Further investigation reveals complete failure of the "mobility" step, with which other sub-7B models did not struggle as much. An example comparison for the "mobility" question "shuffle the deck of cards", "Llama 3.2 3B inst Q4_K_M" answers with "{*"Mobile": False*}", while "Phi3.5 mini Q4_K_M" answers with "'{*"Mobile": True*}\n*The action of shuffling a deck requires moving the cards, which implies that you would need to leave your current workstation (the table) where they are placed. Therefore, this task necessitates mobility beyond just manipulating objects on-table without leaving it entirely.*" A model being too verbose is a problem from performance perspective, as even if the model can generate the correct answer, the additional time spent generation unnecessary tokens slows the overall execution time of the system. It also should be noted that all the

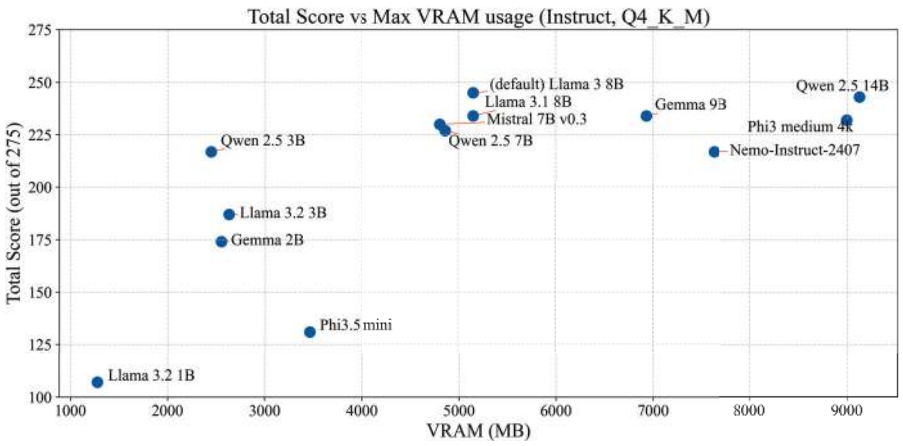


Figure 7.5 VRAM usage of the tested models. ⏎

text generated after the answer has no contribution to it, rather the model is "justifying" why it printed the answer it did.

7.4.4 Result comparison to Benchmark performance

Comparing the test scores against the BFCL benchmark results, it can be seen in figure 7.6 that the models with higher benchmark scores show better results overall, while the default model massively over performs. The "Qwen2.5 3B Inst Q4_K_M" performs well on the "planning" tasks which most other small models struggle with which is reflected in the BFCL results, however on further inspection of the generated outputs, it seems that all "Qwen2.5" models have a problem with following the defined structure of "CoT" reasoning,

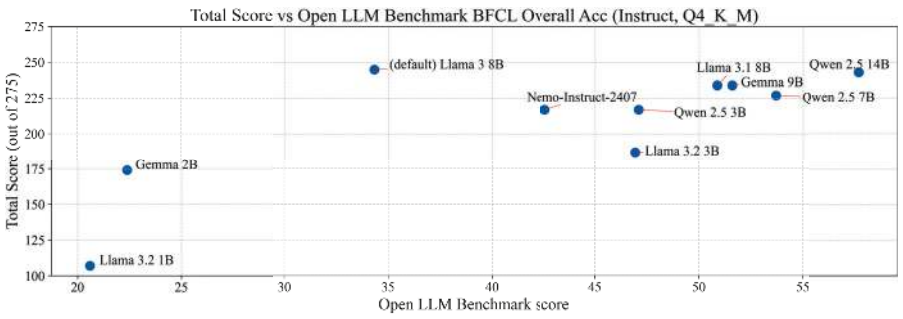


Figure 7.6 Testing results relative to model performance on the BFCL. ⏎

typically ignoring the “My hand: <object>” part, which all the other models, even the ones that fail most of the plans, do maintain, which might imply a different training approach from the other model families.

7.5 Conclusions

Development of an LLM-based system in a resource constrained environment faces many challenges. Besides the obvious limitations on model size, a general problem that such systems may encounter is the over-specializations of prompts for certain applications to not only a specific model (Llama 3 8B Instruct), but to specific weights (Q4_K_M precision) which complicates the adaptation of the system to different models. Tests performed showed that larger models with higher rated benchmark performance were unable to surpass the performance of the default model, falling short. Perhaps more concerning is the fact that the models that almost matched the performance typically did so by making a different set of mistakes. A similar trend was seen between different quantization precisions – lower precisions lose performance, yet going higher does not seemingly improve it. While larger precisions did show better results than larger models, considering that both would require rewrites of the prompt content, what should be the criteria used to decide where to invest time in? As locally testing all the various models and their finetunes is simply not viable, trusted benchmarks are a critical aspect when it comes to deciding on what weights to spend development time. But it seems that certain prompt content may favour one model over another.

This implies the need to solve two problems. The first problem is to eliminate the variance that comes from weights making minor but different mistakes, which requires developing the existing system further with more advanced prompting methods, however the main drawback will be the increase of execution times, which at some point may no longer be practical for a robotic system. Second problem relates to weight “onboarding” to an existing LLM-based system by a more elaborate validation system that is capable of rating new weights without bias for specific weights. Both changes should strive to reduce the need for rewriting of prompts and to improve trust in the reliability of LLMs to perform these tasks.

More research is needed to better understand the deployment of the more compact LLM models in robotic systems and on the edge in general, but this impression may simply be the result of the research cycle, given how recent many of the innovations and performance jumps in local LLMs have been.

Acknowledgements

This work was supported by Chips Joint Undertaking EdgeAI project. The project EdgeAI "Edge AI Technologies for Optimised Performance Embedded Processing" is supported by the Chips Joint Undertaking and its members including top-up funding by Austria, Belgium, France, Greece, Italy, Latvia, Netherlands, and Norway under grant agreement No 101097300.

References

[1] W. X. Zhao et al., "A Survey of Large Language Models," arXiv, arXiv:2303.18223 [cs.CL], 2023.

[2] K. Lin et al., "Text2Motion: from Natural Language Instructions to Feasible Plans," Autonomous Robots, vol. 47, no. 8, pp. 1345–1365, Nov. 2023, doi: 10.1007/s10514-023-10131-7.

[3] W. Yu et al., "Language to Rewards for Robotic Skill Synthesis," arXiv, arXiv:2306.08647 [cs.RO], 2023.

[4] Q. Gu et al., "ConceptGraphs: Open-Vocabulary 3D Scene Graphs for Perception and Planning," arXiv, arXiv:2309.16650 [cs.RO], 2023.

[5] Y. Hu et al., "Toward General-Purpose Robots via Foundation Models: A Survey and Meta-Analysis," arXiv, arXiv:2312.08782 [cs.RO], 2023.

[6] T. Darra, "Gemini Nano is now available on Android via experimental access," Android Developers Blog: Oct. 17, 2024. [Online]. Accessed: Oct. 17, 2024. Available: https://android-developers.googleblog.com/2024/10/gemini-nano-experimental-access-available-on-android.html

[7] "Introducing Apple's On-Device and Server Foundation Models," Apple Machine Learning Research, Jun. 10, 2024. Accessed: Oct. 17, 2024. [Online]. Available: https://machinelearning.apple.com/research/introducing-apple-foundation-models

[8] W.-L. Chiang et al., "Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference," arXiv, arXiv:2403.04132 [cs.AI], 2024.

[9] "GPU inference," Hugging Face, 2024. Accessed: Oct. 17, 2024. [Online]. Available: https://huggingface.co/docs/transformers/perf_infer_gpu_one

[10] T. Brown et al., "Language Models are Few-Shot Learners," arXiv, arXiv:2005.14165 [cs.CL], 2020.

[11] X. Dai, "GPU-Benchmarks-on-LLM-Inference" GitHub, 2023. Accessed: Oct. 17, 2024. [Online]. Available: https://github.com/XiongjieDai/GPU-Benchmarks-on-LLM-Inference

[12] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, “GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers,” arXiv, arXiv:2210.17323 [cs.LG], 2023.

[13] T. Wu, M. Terry, and C. J. Cai, “AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts,” in Proc. 2022 CHI Conf. Hum. Factors Comput. Syst., New Orleans, LA, USA, 2022, pp. 385:1-385:22, doi: 10.1145/3491102.3517582.

[14] J. White et al., “A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT,” arXiv, arXiv:2302.11382 [cs.SE], 2023.

[15] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, “A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications,” arXiv, arXiv:2402.07927 [cs.AI], 2024.

[16] S. Laskaridis, K. Katevas, L. Minto, and H. Haddadi, “MELTing point: Mobile Evaluation of Language Transformers,” arXiv, arXiv:2403.12844 [cs.LG], 2024.

[17] T. E. Zinars, O. Vismanis, P. Racinskis, J. Arents, and M. Greitans, “Natural language conditioned planning of complex robotics tasks,” in Advancing Edge Artificial Intelligence, pp. 131–151, River Publishers, 2024. e-ISBN 9781003478713, doi: 10.1201/9781003478713-6.

[18] A. Hern, “TechScape: Will Meta’s massive leak democratise AI – and at what cost?,” The Guardian, Mar. 07, 2023. Accessed: Oct. 17, 2024. [Online]. Available: https://www.theguardian.com/technology/2023/mar/07/techscape-meta-leak-llama-chatgpt-ai-crossroad

[19] “Models,” Hugging Face. Accessed: Oct. 17, 2024. [Online]. Available: https://huggingface.co/models

[20] J. Wei et al., “Emergent Abilities of Large Language Models,” arXiv, arXiv:2206.07682 [cs.CL], 2022.

[21] R. Schaeffer, B. Miranda, and S. Koyejo, “Are Emergent Abilities of Large Language Models a Mirage?,” arXiv, arXiv:2304.15004 [cs.AI], 2023.

[22] M. Abdin et al., “Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone,” arXiv, arXiv:2404.14219 [cs.CL], 2024.

[23] A. Dubey et al., “The Llama 3 Herd of Models,” arXiv, arXiv:2407.21783 [cs.AI], 2024.

[24] QwenLM, “Qwen2.5,” GitHub, Accessed: Oct. 17, 2024. [Online]. Available: https://github.com/QwenLM/Qwen2.5

[25] A. Q. Jiang et al., “Mixtral of Experts,” arXiv, arXiv:2401.04088 [cs.LG], 2024.

[26] turboderp, “turboderp/exllamav2,” GitHub, Accessed: Oct. 17, 2024. [Online]. Available: https://github.com/turboderp/exllamav2

[27] G. Gerganov, “llama.cpp,” GitHub, Accessed: Oct. 17, 2024. [Online]. Available: https://github.com/ggerganov/llama.cpp

[28] J. Wei et al., “Chain of Thought Prompting Elicits Reasoning in Large Language Models,” arXiv, arXiv:2201.11903 [cs.CL], 2022.

[29] L. Huang et al., “A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions,” ACM Trans. Inf. Syst., Nov. 2024, doi: 10.1145/3703155.

[30] X. Wang et al., “Self-Consistency Improves Chain of Thought Reasoning in Language Models,” arXiv, arXiv:2203.11171 [cs.CL], 2022.

[31] D. Zhou et al., “Least-to-Most Prompting Enables Complex Reasoning in Large Language Models,” arXiv, arXiv:2205.10625 [cs.AI], 2022.

[32] S. Yao et al., “Tree of Thoughts: Deliberate Problem Solving with Large Language Models,” arXiv, arXiv:2305.10601 [cs.CL], 2023.

[33] B. Liu et al., “LLM+P: Empowering Large Language Models with Optimal Planning Proficiency,” arXiv, arXiv:2304.11477 [cs.AI], 2023.

[34] M. Ahn et al., “Do As I Can, Not As I Say: Grounding Language in Robotic Affordances,” arXiv, arXiv:2204.01691 [cs.RO], 2022.

[35] W. Huang et al., “Inner Monologue: Embodied Reasoning through Planning with Language Models,” arXiv, arXiv:2207.05608 [cs.RO], 2022.

[36] J. Liang et al., “Code as Policies: Language Model Programs for Embodied Control,” arXiv, arXiv:2209.07753 [cs.RO], 2022.

[37] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf, “SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Robot Task Planning,” arXiv, arXiv:2307.06135 [cs.AI], 2023.

[38] J. Duan et al., “AHA: A Vision-Language-Model for Detecting and Reasoning Over Failures in Robotic Manipulation,” arXiv, arXiv:2410.00371 [cs.RO], 2024.

[39] J. Zhang et al., “Bootstrap Your Own Skills: Learning to Solve New Tasks with Large Language Model Guidance,” arXiv, arXiv:2310.10021 [cs.RO], 2023.

[40] R. Hazra, P. Zuidberg, and D. R. Luc, “SayCanPay: Heuristic Planning with Large Language Models using Learnable Domain Knowledge,” arXiv, arXiv:2308.12682 [cs.AI], 2023.

[41] F. Joublin et al., “CoPAL: Corrective Planning of Robot Actions with Large Language Models,” arXiv, arXiv:2310.07263 [cs.RO], 2023.

[42] H. Naveed et al., “A Comprehensive Overview of Large Language Models,” arXiv, arXiv:2307.06435 [cs.CL], 2023.
[43] Bartowski, “bartowski/Meta-Llama-3-8B-Instruct-GGUF,” Hugging Face, 2024. Accessed: Oct. 18, 2024. [Online]. Available: https://huggingface.co/bartowski/Meta-Llama-3-8B-Instruct-GGUF
[44] L. Zheng et al., “Judging LLM-as-a-judge with MT-Bench and Chatbot Arena,” arXiv, arXiv:2306.05685 [cs.CL], 2023.
[45] “GGUF,” Hugging Face, 2024. Accessed: Oct. 17, 2024. [Online]. Available: https://huggingface.co/docs/hub/gguf
[46] Bartowski, “bartowski (Bartowski),” Huggingface.co, 2024. Accessed: Oct. 17, 2024. [Online]. Available: https://huggingface.co/bartowski
[47] C. Fourrier, N. Habib, A. Lozovskaya, K. Szafer, and T. Wolf, “Open LLM leaderboard v2,” Hugging Face, 2024. Accessed: Jan. 20, 2025. [Online]. Available: https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard
[48] F. Yan et al., “Berkeley Function Calling Leaderboard,” Accessed: Jan. 20, 2025. [Online]. Available: https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html
[49] Meta AI, “Llama 3.2: Revolutionizing edge AI and vision with open, customizable models,” Meta.com, Sep. 25, 2024. Accessed: Oct. 18, 2024. [Online]. Available: https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/
[50] Gemma Team, “Gemma 2: Improving Open Language Models at a Practical Size,” arXiv, arXiv:2408:00118 [cs.CL], 2024.
[51] A. Q. Jiang et al., “Mistral 7B,” arXiv, arXiv:2310.06825 [cs.CL], 2023.
[52] Mistral AI, “Mistral NeMo,” Mistral AI, Jul. 18, 2024. Accessed: Jan. 20, 2025. [Online]. Available: https://mistral.ai/news/mistral-nemo/
[53] Andrei, “Llama Python Bindings for llama.cpp,” GitHub. Accessed: Oct. 17, 2024. [Online]. Available: https://github.com/abetlen/llama-cpp-python

8

Optimising ViT for Edge Deployment: Hybrid Token Reduction for Efficient Semantic Segmentation

Mathilde Proust[1], Martyna Poreba[1], Calvin Galagain[1,2], Michal Szczepanski[1], and Karim Haroun[1,3]

[1]Université Paris-Saclay, CEA-List, France
[2]ENSTA Paris, France
[3]Université Côte d'Azur, France

Abstract

Vision Transformers (ViTs) achieve high accuracy in multiple vision-related tasks; however, substantial computational and memory demands limit their deployment on resource-constrained edge devices. ViTs process images by splitting them into uniform patches, treating each patch as a separate token. Since not all regions are equally important—detailed areas may require more tokens, while broader regions need—fewer optimizing token processing is essential for improving efficiency. To enhance computational performance, a hybrid token reduction approach is implemented, integrating token merging and pruning strategies. The strengths of CTS, which merges semantically similar and adjacent patches using a CNN-based policy network, and DToP, which halts the processing of tokens that can be predicted with sufficient accuracy in the early layers of the network, are combined in this method. A reduction in computational complexity of up to $2\times$ is shown by the experimental results, with only an approximate 1% drop in accuracy observed on the NVIDIA Jetson AGX Orin 64GB. Exporting a pruned PyTorch model to TensorRT remains a challenging task that requires considerable effort. The difficulties involved are emphasized, and additional work needed to achieve full compatibility with ONNX export standards is outlined.

DOI: 10.1201/9788743808862-8

Keywords: vision transformer, semantic segmentation, token reduction, token merging, model optimization, computational efficiency, computational complexity, edge device.

8.1 Introduction and Background

Vision Transformers (ViTs) have achieved outstanding results in various vision tasks, but their substantial computational and memory requirements pose major obstacles to deployment on resource-constrained edge devices. A combination of software and hardware innovations has emerged to tackle these challenges, focusing on reducing computational complexity, memory consumption, and improving power efficiency. For example, ViTA [1] introduces a dedicated hardware accelerator that optimises ViT inference for real-time applications on edge devices, reducing computational overhead and enhancing efficiency. Another approach [2] utilises an integer-only systolic array accelerator to minimise power consumption and computational demands. Additionally, the ME-ViT accelerator [3] offers a memory-efficient FPGA-based solution that optimises data flow and storage, lowering memory usage and power consumption. The 109-GOPs/W FPGA-based accelerator [4] marks significant progress by incorporating a weighted data flow mechanism that minimizes energy consumption. This approach prioritizes data reuse, optimizing resource efficiency and reducing power usage. On the other hand, researchers have explored various optimization techniques, including quantization, distillation, and pruning to bridge the gap between the high performance of ViTs and the constraints of edge environments, making them more practical for resource-limited settings. For instance, MobileViT [5] introduces a variant of ViTs that merges convolutional neural networks (CNNs) with transformers, resulting in a lightweight model that maintains high accuracy while being suitable for mobile and edge devices. TinyViT [6] employs knowledge distillation to create a smaller, more efficient transformer model that retains high performance, making it ideal for edge applications. Similarly, EdgeViTs [7] are specifically designed for edge devices, incorporating optimized attention mechanisms and downsampling strategies.

ViTs typically generate visual patches by splitting an image into a uniform, fixed grid, where each grid cell is treated as a distinct token. Though straightforward, this approach overlooks the varying complexity of image content, as certain regions can be represented with fewer tokens due to their homogeneity. For example, in an image depicting a busy street, tasks such as identifying vehicles and pedestrians may necessitate a higher density of tokens. In contrast, broader areas of the image, such as the sidewalk or the sky, may require significantly fewer tokens. This disparity in token necessity raises an important question: is it truly essential to process such a large number of tokens at every layer of the network? Given that the computational complexity of ViT scales quadratically with the length of input sequences, a reduction in the number of tokens presents a viable strategy for decreasing computational costs. By intelligently selecting and utilising tokens based on their relevance to the task, performance can be optimised while simultaneously reducing the resource demands on the model.

In this context, our work introduces a hybrid token reduction mechanism aimed at enhancing the efficiency of ViTs for semantic segmentation tasks. This method integrates two cutting-edge techniques: patch merging and early-pruning. A class-agnostic CNN-based network, trained independently from the ViT, merges semantically similar and adjacent patches, while early-pruning stops the processing of tokens that can be confidently predicted in the early layers, reducing unnecessary computations. We implement this method with semantic segmentation transformer models, specifically ViT-Base and ViT-Tiny, and perform experiments on the NVIDIA Jetson AGX Orin 64GB platform.

8.2 Related Work

Token reduction techniques are generally tailored to the specific task they address. State-of-the-art methods predominantly focus on classification. In this case, token pruning methods often permanently eliminate tokens, as they no longer affect the outcome. However, in dense prediction tasks like semantic segmentation, patches cannot be completely discarded, as each one plays a role in the pixel-level predictions needed for detailed results. For such tasks, ViTs handle a large number of tokens, where both the size and number of tokens must be carefully selected to preserve essential details while minimizing computational complexity. Given the demands of dense prediction tasks, not all token reduction methods are suitable, with merging techniques generally proving more effective than pruning approaches. Unlike pruning, which irreversibly discards tokens and risks losing critical information, merging aggregates similar patches, retaining essential details. This approach allows the model to maintain accuracy while reducing computational complexity by carefully selecting which tokens to combine based on their relevance, thereby providing the flexibility needed to adapt to the complexities of different image content. Among the token reduction methods extended to support dense prediction tasks is DynamicViT [8], [9] that employs a dynamic token selection mechanism. Similarly, ToFu [10] has produced notable results in image generation tasks, highlighting its potential in areas requiring dense, detailed predictions. The authors of TCFormer [11] propose their method as a general solution applicable to a wide range of vision tasks, such as object detection and semantic segmentation. Nonetheless, TCFormer faces a major drawback: the computational complexity of its KNN-DPC algorithm increases quadratically with the number of tokens, which undermines its efficiency, especially when handling high-resolution images.

To the best of our knowledge, only three token reduction methods have been specifically designed for the segmentation. One such approach is Content-aware Token Sharing (CTS) [12], which introduces a class-agnostic policy model using a CNN network trained separately from the ViT. CTS identifies whether adjacent image patches belong to the same semantic class; if they do, they can share a common token. This is achieved through binary classification to form fixed-size groups of patches within the input image, ensuring spatial coherence while eliminating the need to process unnecessary tokens. Another approach, Dynamic Token Pruning (DToP) [13], enables early-pruning for tokens, allowing simpler tokens to complete their predictions earlier in the network. DToP divides the transformer into distinct stages and utilizes auxiliary blocks for early prediction generation. It also incorporates the attention-to-mask (ATM) module [14] as the segmentation head, which improves its efficiency in handling dense, pixel-level predictions. Finally, SVIT [15] introduces an innovative method that utilizes a lightweight two-layer MLP (Multi-Layer Perceptron) to dynamically select tokens for processing within the transformer block. One of its key features is that it prunes tokens while retaining them in feature maps, enabling their reactivation in later layers. This ensures that important information is preserved, even if some tokens are not processed in the early stages of the network.

8.3 Methodology

Re-evaluating the traditional fixed-grid approach in ViTs paves the door to more efficient architectures that can handle diverse visual tasks with greater precision and reduced computational overhead. In the vast majority of images, there exist homogeneous regions where it is unnecessary to process redundant patches separately. By minimizing the number of input patches, we can reduce the total number of tokens handled by the ViT blocks. This approach helps prevent the system from expending resources on superfluous tokens, leading to lower energy consumption. This drives our investigation into improving the efficiency of ViTs through a token merging and pruning strategy tailored for inference on edge devices, specifically aimed at enhancing performance in semantic segmentation. Our method integrates the strengths of two state-of-the-art techniques: content-aware patch merging through CTS and early token pruning via DToP. Figure 8.1 outlines the proposed hybrid token optimization mechanism. Tokenization initiates the process, dividing the image into a regular grid of patches. To minimise the number of patches that need processing, we utilise a class-agnostic CNN network to merge neighboring, semantically similar patches. Next, the token-sharing module transforms these non-uniform size patches into tokens Zi

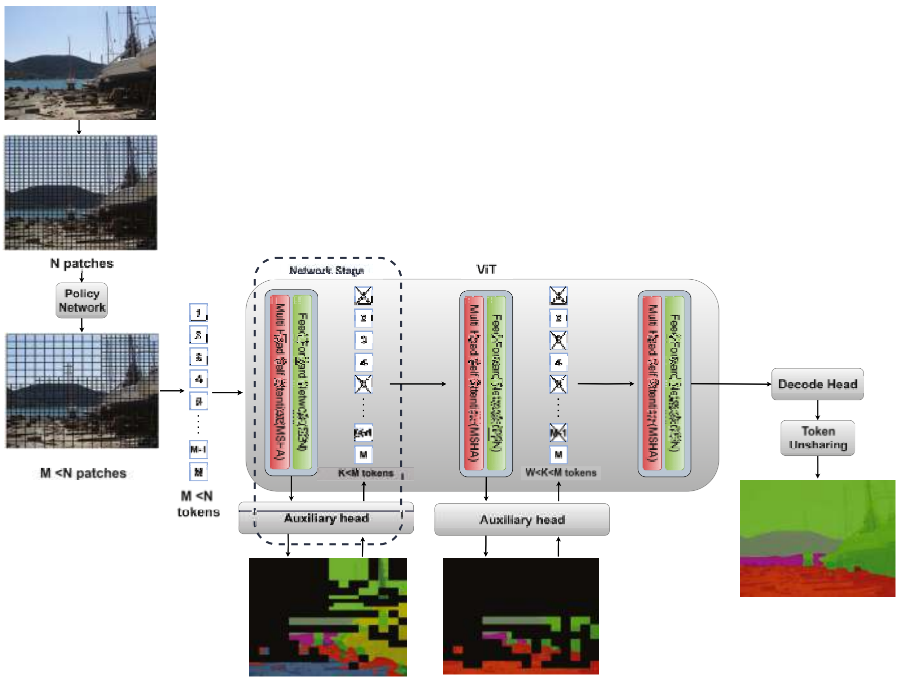


Figure 8.1 Outline of the Proposed Hybrid Token Optimization Technique. ⏎

using a linear embedding function as follows:

$$Z_i = f_{\text{embed}}\,(P_i) \tag{8.1}$$

where Pi represents the group of patches obtained from the image, in which each patch $pi \in P$ is defined as a sub-region of the image, and $\mathrm{f}_{\text{embed}}$ (.) the embedding function that maps into supertokens Zi.
As in DToP, the ViT backbone is organised into M stages, with auxiliary heads identifying high-confidence tokens that are masked and excluded from further calculations. Let C denote the set of high-confidence tokens, where each token is determined by a confidence score $ck \in C$:

$$c_k = f\,(z_i) \text{ if confidence } (z_i) > \text{ threshold} \tag{8.2}$$

This operation is performed on carefully selected layers, specifically after a certain number of transformer blocks. Finally, the model processes the remaining tokens to generate the final output through per-token predictions.

8.3.1 Content-aware Patch Merging

To apply the CTS method to any conventional transformer-based model, it is necessary to incorporate a token sharing function Zi, a token unsharing function, and a policy model. The class-agnostic policy network determines which patches are eligible to share a token prior to their entry into the ViT. It focuses on grouping only square neighboring regions, facilitating the seamless reassembly of tokens at the output of the ViT backbone. CTS comes with a lightweight CNN network to generate probability scores for each 2×2 patch group. It is based on the EfficientNetLite0 model [17], pre-trained on ImageNet-1K [18]. This model predicts a similarity score S for a window of n patches $\omega j = \{p1, p2, \dots pn\}$:

$$S = \sigma \;\; W^T(\omega) \tag{8.3}$$

where Wp is the learned weight matrix of the policy network and $\sigma(.)$ is the sigmoid activation function.

Finally, only the top 103 patch windows ωj are merged into 2×2 groups, based on the highest-ranked probabilities. As a result, the number of patches that are converted into tokens is significantly reduced (Figure 8.2). For example, a 512×512 resolution input image traditionally produces 32×32 patches, with each patch covering 16×16 pixels, resulting in 1 024 patches

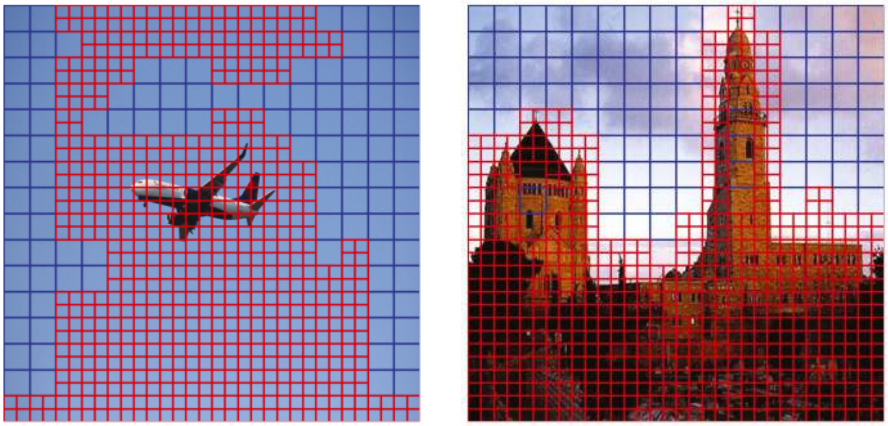


Figure 8.2 Results of Patch merging: grouped patches in blue, individual patches in red. ⏎

to process. After applying the CTS method, only 715 patches are sent to ViT, reducing the number of tokens by 30%.

8.3.2 Early-Pruning

The core concept of DToP is to identify easy, high-confident tokens in the intermediate layers and exclude them from further computations. After a predetermined number of attention block layers, the model directs tokens to an auxiliary segmentation head, which adapts the ATM, and applies a stopping criterion based on the confidence of its predictions. Specifically, at stage M, a confidence score c(m) is calculated for each token Zi, which is formalized as follows:

$$Z(m+1) = \{zi \mid c(m) < \theta\} \tag{8.4}$$

where $Z(\mathrm{m}+1)$ represents the set of tokens passed to the next stage. Tokens with confidence scores exceeding a predefined threshold θ are classified as high-confidence tokens and are discarded, while the low-confidence tokens proceed further through the network. This underscores the significance of strategically positioning auxiliary heads within the network. Placing them too early could make it difficult for the model to accurately predict the class of any tokens. We adopt the recommendations from the original DToP paper concerning hyperparameters and the positioning of auxiliary heads, acknowledging that they may not be optimal in all scenarios.

8.4 Experiments

We integrate our hybrid token reduction mechanism into the SegViT semantic segmentation framework [14], which serves as the baseline for our performance comparison study. All experiments are performed using the MMSegmentation (mmseg) toolbox [19], which allows for easy customization of models by combining different backbones. We integrate ViT-Base, which includes 12 encoder layers, a 768-dimensional hidden layer, and 12 attention heads, alongside ViT-Tiny, which features 12 encoder layers, a 192-dimensional hidden layer, and 3 attention heads. Both process images by dividing them into 16×16 pixel patches. We follow the standard training settings in mmseg and use the same hyperparameters as the original papers. For DToP, we adopt the configuration recommended by the authors, and split the ViT backbone into three stages with token pruning occurring at the 6th and 8th layers for ViT-Base. This setup is intended to achieve an effective balance between computational cost and segmentation accuracy. Additionally, we choose to examine a model divided into two stages and position the pruning head after the 8th layer. Since the authors did not provide configurations for ViT-Tiny, we applied the same configuration as ViT-Base, as ViT-Tiny contains the same number of blocks. Experiments are conducted on ADE20k [20], a dataset focused on semantic segmentation. Mean Intersection over Union (mIoU) assesses segmentation accuracy, while giga floating-point operations (GFLOPs), measured with fvcore package [16], reflect model complexity, and frames per second (FPS) indicates throughput.

For inference on the NVIDIA Jetson AGX Orin 64GB, we primarily use PyTorch because of its flexibility and ease of use during model development. To optimize performance and fully leverage the hardware capabilities of the NVIDIA Jetson platform, TensorRT is the preferred option. However, we encountered several challenges when exporting pruned models to ONNX and TensorRT. While PyTorch 2.4 supports all necessary layers, it presents compatibility issues with the OpenMMLab libraries. Specifically, the mmseg framework, which depends on MMCV (a foundational library for computer vision tasks) and MMEngine (a runtime engine for managing training, validation, and inference loops), complicates cross-compilation with the latest Python and the preferred CUDA version. Although we ultimately succeeded in validating the ONNX export, TensorRT indicated a size mismatch in one of the backbone layers. It appears that a specific layer contains parameters not supported by TensorRT, necessitating further investigation to find a solution.

Table 8.1 Performance of Token Reduction Method integrated with ViT-Base ⏎

Method	mIoU [%]	GFLOPs	FPS
SegViT	48.3	112.8	6.8
+CTS	47.8	75.4	13.3
+DToP@[6,8]	46.1	86.3	3.9
+CTS&DToP@[6,8](ours)*	47.2	63.0	12.7
+CTS&DToP@[6,8](ours)	47.7	62.1	4.5
+CTS&DToP@[8](ours)	48.3	68.3	6.5

**on a single A100GPU*

Table 8.2 Performance of Token Reduction Method integrated with ViT-Tiny ⏎

Method	mIoU [%] e	GFLOPs	FPS
SegViT	37.8	12.0	15.6
+CTS	37.3	7.6	15.4
+DToP@[6,8]	38.0	9.9	8.2
+CTS&DToP@[6,8](ours)*	37.7	6.8	19.0
+CTS&DToP@[6,8](ours)	37.7	6.8	9.3
+CTS&DToP@[8](ours)	38.5	6.9	13.1

**on a single A100GPU*

Table 8.1 and Table 8.2 summarise the performance achieved by the model in FP32 format. The results show that integrating our hybrid token reduction method into SegViT allows us to maintain a comparable mIoU, with segmentation accuracy loss kept within a maximum of 1%. This method achieves a reduction in complexity of up to 45% for ViT-Base and 42% for ViT-Tiny. By applying only the token merging via CTS, we observe a reduction in computational complexity for ViT-Base and ViT-Tiny of 33% and 37%, respectively. The early-pruning technique via DToP impacts both computational complexity and inference speed, with the number of auxiliary heads playing a crucial role. Although placing the pruning heads at the 6th and 8th positions yields a 23% reduction in GFLOPs for ViT-Base. This advantage comes at the expense of increased inference time, which can slow the process down by nearly a factor of two.

Figure 8.3 illustrates the inference time for each layer of the model, including the auxiliary heads used for pruning. It shows that pruning tokens with segmentation heads equipped with ATM modules tends to be excessively slow, underscoring the need for future work to focus on optimization. Given this observation, a single auxiliary head presents the best trade-off between reducing complexity and time inference.

Figure 8.4 and Figure 8.5 display visualized predictions, where the number of pruned tokens increases from bottom to top. In “easy” samples, most

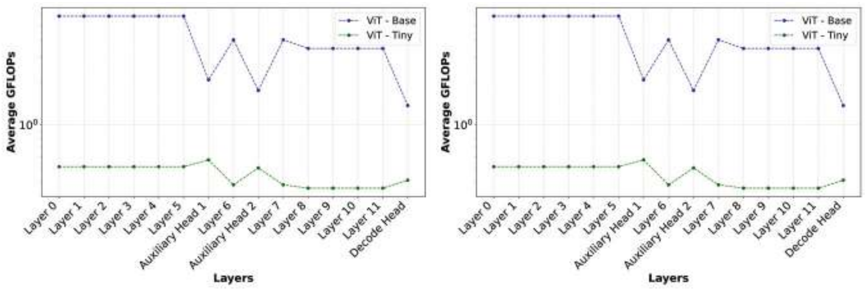


Figure 8.3 Layer-by-layer analysis considering GFLOPs and Throughput (FPS) for pruning heads placed at positions 6 and 8 ⏎

tokens are pruned after the 6th ViT block, while in “hard” cases, the majority of tokens are retained until the final layer. The second auxiliary head (at the 8th layer) was often unable to prune a significant number of tokens, as it was placed too soon after the first head. This highlights that using two pruning heads in smaller networks like ViT-Base and ViT-Tiny is not always necessary or effective.

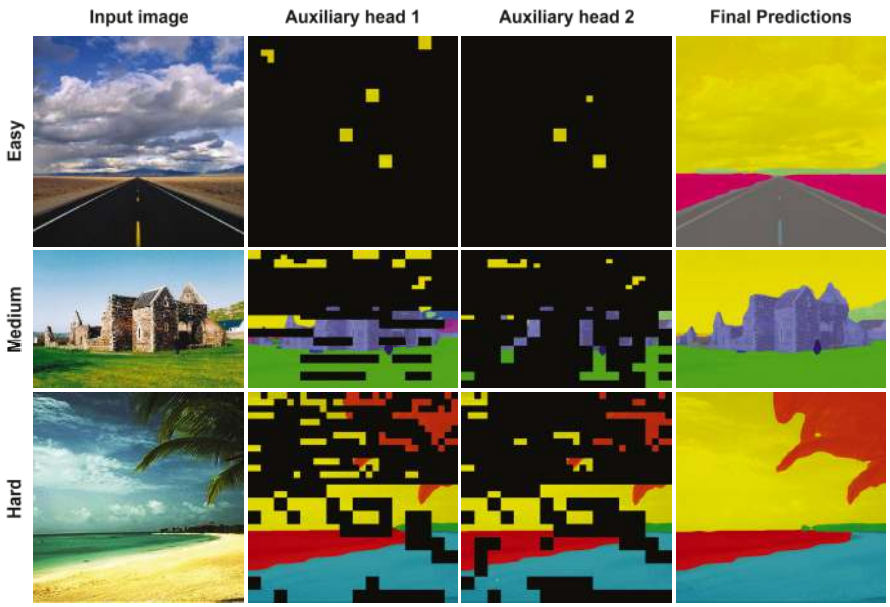


Figure 8.4 ViT-Base segmentation results with pruned tokens masked in black ⏎

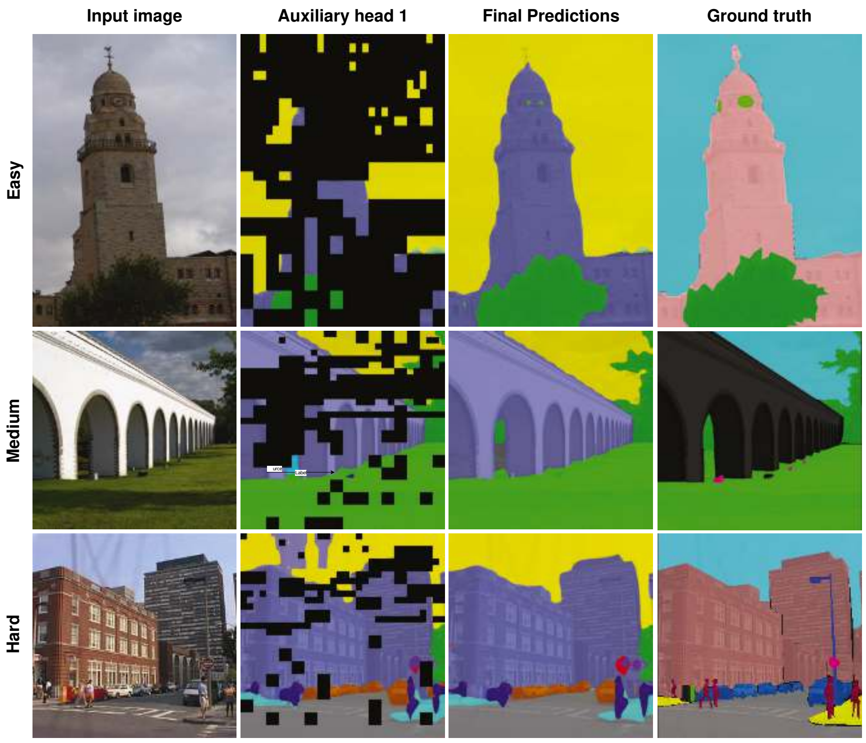


Figure 8.5 ViT-Tiny segmentation results with pruned tokens masked in black ⏎

8.5 Conclusion

We introduced a hybrid token optimization mechanism specifically designed for semantic segmentation, which merges semantically similar neighboring patches and incorporates dynamic token pruning based on an early-pruning strategy. Implementing our on-the-fly pruning approach significantly influences architectural design, requiring careful attention to resource allocation and dynamic token management. Nevertheless, proposed token reduction mechanism can seamlessly transition to a fixed-token strategy. By simply fixing the number of top-k most confident tokens pruned by each auxiliary head, rather than relying on the threshold θ, we unlock several advantages. This streamlines hardware design by providing predictable resource allocation and optimizing performance. It also enhances memory management, improves scalability, minimizes overflow risks, and enables parallel processing. Our token reduction technique has been integrated into transformer models (ViT-Base and ViT-Tiny) within the mmseg framework. Through experiments conducted on established segmentation benchmark with an NVIDIA Jetson AGX Orin 64GB, we showed that this optimization method can lower computational costs by up to 45% while maintaining accuracy with minimal impact. Nevertheless, while using auxiliary heads to prune high-confidence tokens lowers computational complexity, it significantly affects inference speed. We suggest that future work concentrate on exploring methods to optimize the architecture of auxiliary heads. Despite its advantages, the complex mmseg framework and the dynamic pruning can complicate model export, as both ONNX and TensorRT require a consistent model structure. Future work will tackle these challenges, aiming to create a more seamless and efficient export pipeline. Efforts will focus on verifying the compatibility of the pruned models with TensorRT and ensuring consistent shapes for all inputs to conditional layers. This may involve modifying the mmseg framework to include shape-alignment operations or developing custom ONNX operations to address shape mismatches.

Acknowledgements

This work was conducted as part of the AI optimization roadmap within the LIAE laboratory of the DSCIN department at CEA-LIST.

References

[1] S. Nag, G. Datta, S. Kundu, N. Chandrachoodan and P. Beerel, “ViTA: A Vision Transformer Inference Accelerator for Edge Applications”, in 2023 IEEE International Symposium on Circuits and Systems (ISCAS), pp.1-5, 2023, https://doi.org/10.1109/ISCAS46773.2023

[2] M. Huang, J. Luo, Ch. Ding, Z. Wei, S. Huang, H. Yu, “An Integer-Only and Group-Vector Systolic Accelerator for Efficiently Mapping Vision Transformer on Edge,” in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 70, no. 12, Dec. 2023, pp. 5289-5301. https://doi.org/10.1109/tcsi.2023.3312775

[3] K. Marino, P. Zhang and V. Prasanna, “ME- ViT: A Single-Load Memory- Efficient FPGA Accelerator for Vision Transformers,” in 2023 IEEE 30th International Conference on High Performance Computing, Data, and Analytics (HiPC), Goa, India, 2023 pp. 213-223. https://doi.org/10.1109/HiPC58850.2023.00039

[4] S. Li, C. Chen, L. Yu, X. Wang, and H. Zhang, “A 109-GOPs/W FPGA-based Vision Transformer Accelerator with Weight-Loop Dataflow Featuring Data Reusing and Resource Saving,” in Proceedings of the 2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2024, pp. 1-12.

[5] S. Mehta and M. Rastegari, “MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer,” openreview.net, Mar. 04, 2022.

[6] K. Wu et al., “TinyViT: Fast Pretraining Distillation for Small Vision Transformers,” Lecture notes in computer science, pp. 68–85, Jan. 2022, doi: https://doi.org/10.1007/978-3-031-19803-8_5.

[7] J. Pan et al., “EdgeViTs: Competing Light-weight CNNs on Mobile Devices with Vision Transformers.” Accessed: Sep. 16, 2024. Available online:https://www.ecva.net/papers/eccv_2022/papers_ECCV/papers/136710294.pdf

[8] Y. Rao, W. Zhao, B. Liu, J. Lu, J. Zhou, and C.-J. Hsieh, “DynamicViT: Efficient Vision Transformers with Dynamic Token Sparsification,” arXiv (Cornell University), vol. 34, Dec. 2021.

[9] Y. Rao, Z. Liu, W. Zhao, J. Zhou, and J. Lu, “Dynamic Spatial Sparsification for Efficient Vision Transformers and Convolutional Neural Networks,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 45, no. 9, pp. 10883–10897, Apr. 2023, https://doi.org/10.1109/tpami.2023.3263826

[10] M. Kim, S. Gao, Y.-C. Hsu, Y. Shen, and H. Jin, “Token Fusion: Bridging the Gap between Token Pruning and Token Merging,” Jan. 2024, https://doi.org/10.1109/wacv57701.2024.00141.

[11] W. Zeng et al., “Not All Tokens Are Equal: Human-centric Visual Analysis via Token Clustering Transformer,” 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2022, https://doi.org/10.1109/cvpr52688.2022.01082

[12] C. Lu, D. de Geus and G. Dubbelman, “Content-aware Token Sharing for Efficient Semantic Segmentation with Vision Transformers,” in 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 2023 pp. 23631-23640.

[13] Q. Tang, B. Zhang, J. Liu, F. Liu and Y. Liu, “Dynamic Token Pruning in Plain Vision Transformers for Semantic Segmentation,” in 2023 IEEE/CVF International Conference on Computer Vision (ICCV), Paris, France, 2023 pp. 777-786.

[14] B. Zhang, et al., 2024. “SegViT: semantic segmentation with plain vision transformers”. in Proceedings of the 36th International Conference on Neural Information Processing Systems (NIPS ’22). Curran Associates Inc., Red Hook, NY, USA, Article 359, pp. 4971–4982.

[15] Y. Liu, M. Gehrig, N. Messikommer, M. Cannici and D. Scaramuzza, “Revisiting Token Pruning for Object Detection and Instance Segmentation,” in 2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), Waikoloa, HI, USA, 2024 pp. 2646-2656.

[16] facebookresearch, “GitHub - facebookresearch/fvcore: Collection of common code that’s shared among different research projects in FAIR computer vision team.,” GitHub, 2019. Available online: https://github.com/facebookresearch/fvcore

[17] M. Tan, and Q.V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. Proceedings of the 36th International Conference on Machine Learning, ICML 2019, Long Beach, 9-15 June 2019, pp. 6105-6114.

[18] O. Russakovsky et al., “ImageNet Large Scale Visual Recognition Challenge,” International Journal of Computer Vision, vol. 115, no. 3, pp. 211–252, Apr. 2015, https://doi.org/10.1007/s11263-015-0816-

[19] MMSegmentation Contributors, “OpenMMLab Semantic Segmentation Toolbox and Benchmark,” GitHub, Jul. 01, 2020. https://github.com/open-mmlab/mmsegmentation

[20] B. Zhou et al., “Semantic Understanding of Scenes Through the ADE20K Dataset,” International Journal of Computer Vision, vol. 127, no. 3, pp. 302–321, Dec. 2018, https://doi.org/10.1007/s11263-018-1140-0

9

Recent Trends in Edge AI: Efficient Design, Training and Deployment of Machine Learning Models

Mark Deutel[1*], Maen Mallah[2*], Julio Wissing[2*], and Stephan Scheele[3]

[1]Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
[2]Fraunhofer IIS, Fraunhofer Institute for Integrated Circuits, Germany
[3]Ostbayerische Technische Hochschule Regensburg, Germany
[*]These authors contributed equally to this work.

Abstract

With a rising demand for ubiquitous smart systems, processing and interpreting large quantities of data generated on the edge at a high velocity is becoming an increasingly important challenge. Machine learning (ML) models such as Deep Neural Networks (DNNs) are an essential tool of today's artificial intelligence due to their ability to make accurate predictions given complex tasks and environments. However, Deep Learning is computationally complex and energy intensive. This seems to contradict the characteristics of many edge devices, which have only limited memory, computational resources, and energy budget available. To overcome this challenge, an efficient ML model design is crucial that incorporates available optimization techniques from hardware, software, and methodological perspective to enable energy-efficient deployment and operation on the edge. This work comprehensively summarizes recent techniques for training, optimizing, and deploying ML models targeting edge devices. We discuss different strategies for finding deployable ML models, scalable DNN architectures, neural architecture search, and multi-objective optimization approaches, to enable feasible trade-offs considering available resources and latency. Furthermore, we give insight into DNN compression methods such

DOI: 10.1201/9788743808862-9

as quantization and pruning. We conclude by investigating different forms of cascaded processing, from simple multi-level approaches to highly branched compute graphs and early-exit DNNs.

Keywords TinyML, energy efficient AI, neural architecture search, pruning, quantization, cascaded processing.

9.1 Introduction

With the rise of the Internet of Things (IoT), the need to interpret large quantities of data from embedded sensor systems has become ubiquitous. The possible application domains are nearly endless, ranging from Smart Cities over sustainable resource use to agriculture and many more. Yet, including computationally complex data processing with models like Deep Neural Networks (DNN) seems contradictory to the characteristics of small, embedded devices, which usually have substantial limitations regarding energy usage and computing capabilities. To overcome this challenge, multiple approaches have been taken to reduce the energy footprint of machine-learning models, often referred to as TinyML. This work will survey recent literature concerning the most prominent aspects in this field with a focus on embedded sensor systems, which we define as:

- Processing element embedded in a device made for a specific application
- Computations are done by a microprocessor, microcontroller, or FPGA
- Optionally includes an acceleration unit (NPU, DSP, etc.)
- Reduced or no operating system
- Battery or energy-limited

The study starts with scalable DNN architectures in Section 2 Scalable Deep Neural Architecture search to find efficient network topologies in Section 3. The review DNN compression methods to optimize neural networks during and after their training are reviewed e.g., Pruning in Section 4 and Quantization in Section 5. The survey is concluded by looking at cascaded systems in Section 6, which allow partial execution of machine-learning models. Section 7 provides a short summary and discussion.

9.2 Scalable Deep Neural Network Architectures

Scaling DNNs, i.e., increasing their capacity, is a commonly used technique to control the trade-off between the performance of a DNN, e.g. accuracy, and its resource requirements. The most common form of scaling is achieved either by adding more layers, changing the resolution of the input, or changing the network's with, e.g. by increasing the number of filters in a DNNs convolutional layer, see Figure 9.1. In the following we provide an overview of relevant scalable DNN architectures commonly used on edge platforms.

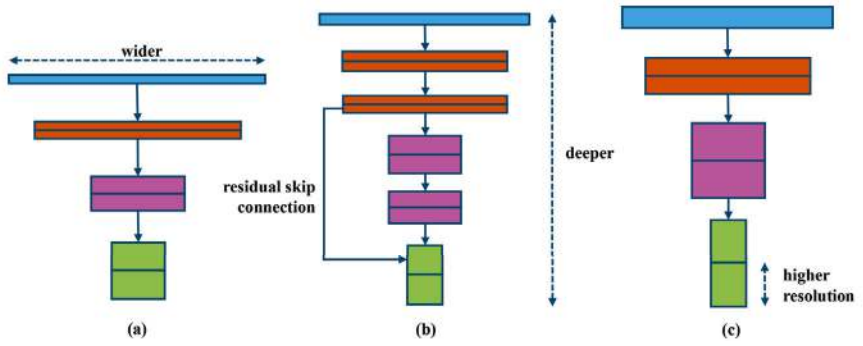


Figure 9.1 Illustration of commonly used approaches for DNN scaling. DNNs can be scaled by either (a) widening the input of the DNN, (b) deepening the DNN by adding more layers and residual skip connections, or (c) increasing the resolution of the feature maps by adding more filters. ⏎

9.2.1 Residual networks

Residual networks [1] have been proposed to facilitate the training of very deep neural network architecture. Training such architecture is often hindered by problems such as the vanishing/exploding gradient problem or degradation problems caused by the depth of the DNN [2]. ResNet attempts to address these problems by explicitly fitting layers to a residual mapping instead of the entire underlying mapping. The authors hypothesize that it is easier to optimize the residual mapping than the original unreferenced mapping. They argue that "to the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers". In feedforward networks, residual mappings can be realized by introducing "shortcut connections," which are connections that bypass one or more layers. In the case of ResNet, the shortcut connection simply performs an identity mapping and is then added to the result produced by the stacked layers it bypasses. Therefore, introducing it adds almost no computational complexity (just an elementwise added operation).

The standard "residual block" used by the authors in their proposed ResNet architectures consists of two fully convolutional layers and a bypass connection. However, given the exploding training times for very deep neural networks, the authors also propose an alternative "bottleneck" block. It consists of three convolutional layers (instead of two), where the first and the last are 1×1 pointwise convolutions, leaving the intermediate layer as a bottleneck with smaller input/output dimensions.

9.2.2 MobileNet

MobileNet [3] is a class of DNNs that specifically focuses on deployment on edge platforms, both from a size and runtime perspective. One of the main features of the architecture is the heavy use of depth-separable convolutions, "which is a form of factorized convolutions which factorize a standard convolution into a depth wise convolution and a 1x1 convolution called a pointwise convolution". This means that instead of performing both filtering and combining inputs into a new set of outputs as one operator (standard convolution), the operation is factorized into a separate layer for filtering and a separate layer for combining. The main advantage of this approach is that the factorized representation drastically reduces both the computational complexity and the size of a DNN model compared to using standard convolutions.

To enable scalability, the authors introduce a width multiplier parameter. The role of the parameter is to thin a network uniformly at each layer. Furthermore, they introduce a depth scalar parameter, which is multiplied by the spatial dimensions of the initial input of the network and as a result also scales down the activation tensors of all subsequent layers. In recent years, the authors of the original MobileNet paper have presented two updated versions of the MobileNet architectures. MobileNetV2 [4] is very similar to its predecessors, except that it uses "inverted residual blocks with bottlenecking features". As in the original MobileNet architectures, these blocks consist of a depth-separable convolution but also include a third linear 1x1 convolutional layer that is not followed by any nonlinearity, e.g., ReLU. The authors claim that "experimental evidence suggests that using linear layers is crucial as it prevents nonlinearities from destroying too much information". In addition, the blocks introduce a shortcut connection like that of residual blocks. However, the authors explain that while in regular residual blocks the expansion layers, i.e. the high number of channels, are connected by the shortcut, here the bottleneck layers, i.e. the low number of channels, are connected. Hence the name "inverted residual block". The advantage of this inversion is a much more memory efficient design.

MobileNetV3 [5] improves on its predecessors by introducing lightweight attention modules based on squeeze and excitation networks [6] in the bottleneck structure. Furthermore, all layers are upgraded with a modified version of the swish nonlinearity (h-swish), which the authors claim is faster to compute and more quantization-friendly. Both the swish operator and the squeeze and excitation layers use the sigmoid operator. Since this can be expensive to compute on some platforms, it is replaced by the hard sigmoid (piecewise linear approximation of the sigmoid function). In addition, particularly expensive layers of MobileNetV2 have been redesigned to be more efficient. This includes the last layers of the previous architecture, where the authors describe that they were able to „drop three expensive layers at the end of the network at no loss of accuracy."

9.2.3 EfficientNet

EfficientNet [7] is a class of eight differently scaled DNN architectures (B0-B7). The scaling is performed as a combination of depth, width, and resolution scaling, which the authors call compound scaling, and which they control by an additional parameter φ. The authors note that increasing the three available scaling dimensions comes at different costs which means that a trade-off between the parameters must be made. For example, the authors mention that for regular convolutional operations, doubling the network depth parameter will double the FLOPS, but doubling the network width parameter or resolution parameter will even quadruple the FLOPS.

As their baseline network, the authors use a regular CNN (B0) targeting 400M FLOPS, which the authors searched for by using a multi-objective neural architecture search that optimizes both accuracy and FLOPS, and which they based on their previous work [8]. The authors used $ACC\,(m) * \left(\frac{FLOPS(m)}{T}\right)^{w}$ as their optimization objective, where $ACC\,(m)$ and $FLOPS(m)$ denote the accuracy and FLOPS of the model while m and T denotes the target FLOPS, and w $= -0.007$ is an additional hyperparameter used to control the trade-off between the two metrics. From their baseline network (B0), the authors then use a two-step process to derive all seven other architectures (B1-B7):

- *STEP 1*: assuming twice as many resources are available as a starting point and then do a small grid search of the three scaling parameters for depth, width, and resolution.

- *STEP 2*: fix the best-found depth, width, and resolution parameters as constants and scale the baseline mesh with different φ using to get the final set of architectural trade-offs B1-B7.

9.2.4 Scalable weights

Another common approach to achieve dynamic scalability at runtime is to adapt trainable parameters (“weights”) based on the received input. A conditionally parametrized convolutional layer (CondConv) is proposed in [9] that can be used as a drop-in replacement for standard convolutional layers. Their design parametrizes the kernels of a regular convolution as a linear combination of n experts $W_1, \ldots, W_n$ weighted by $\alpha_1, \ldots, \alpha_n$ functions of the input learned by gradient descent. The authors argue that “this is much more computationally efficient than increasing the size of the convolutional kernel itself, because the convolutional kernel is applied at many different positions within the input, while the experts are combined only once per input”. In conclusion, CondConv can introduce the same amount of additional capacity as MoE while being more computationally efficient by requiring only one convolution.

Deformable convolutions [10], [11] implement the concept of dynamic receptive fields by sampling feature pixels during the computation of convolutions from adaptive locations based on the input. The offset of each sample is generated using an additional convolution that generates an offset field based on the input feature map. As a result, deformable convolutions generalize various transformations for scale, (anisotropic) aspect ratio and rotation. The concept of weight prediction, first proposed by [12] allows for directly modifiable weights in a feedforward network that are contextually modified at runtime by a second controller network. The authors propose this architecture as a memory efficient alternative for representing temporal information compared to recurrent neural networks. The idea was adapted to more modern DNN architectures by [13], [14]. Here, the filters of a convolution are dynamically generated from the input using an additional “filter generating network”. The two inputs of the convolution can be either identical or different, depending on the task at hand.

9.2.5 Practical Considerations

Scalable DNN architecture helps to easily adapt a network to a given set of resource constraints, often by simply tuning a set of hyperparameters that do not require extensive prior knowledge about the internal structure of the DNN. This makes the architecture presented in this section ideal candidates for deployment on edge systems or other resource constraint environments. In addition, scalable DNN architectures are versatile in that they can easily be used in combination with other techniques such as neural architecture search, pruning, and quantization, which we will discuss later in this review.

9.3 Neural Architecture Search for Resource Aware DNN Deployment

The search for a feasible DNN architecture that can be deployed on an edge system with given resource constraints is typically formulated as a multi-objective hyperparameter optimization (HPO) problem [15], solving a black-box function that maps from a set of DNN architecture-specific hyperparameters to a set of target metrics such as accuracy, memory consumption, latency, or throughput, on which platform-specific constraints are defined. In this definition, sampling the black box is equivalent to training a DNN with a specific configuration on a given dataset. Often synonymously used with HPO is Neural Architecture Search (NAS). NAS describes the process of finding good DNN architectures in an automated, non-human controlled way. Although it is primarily concerned with only maximizing DNN performance, i.e., solving a single-objective optimization problem, most concepts can be intuitively extended to a resource-aware multi-objective search. As a result, multi-objective NAS for edge platforms has recently become more of a focus as well, e.g. [16, 8]. There are three approaches to NAS that are prominently discussed in the literature: First, black-box HPO [17]-[20], second, differentiable NAS [21], [22] and third, zero-cost NAS [16], [23], [25]-[30]. Black-box HPO works reliably and can be easily extended to the multi-objective case, but it is also slow and often inefficient sample, requiring many different DNNs to be trained and evaluated.

Differentiable NAS relaxes the optimization problem so that the architecture can be optimized as part of the regular training of a DNN. However, recent research suggests that differentiable NAS has stability problems and does not generalize well [24]. Finally, zero-cost NAS is very time-efficient since it does not train DNNs directly but uses an empirical surrogate model. As a result, it does not provide accurate information about the performance of an architecture, but only simple statistics derived from the surrogates [31]. In the following, we discuss each of the three approaches and their implications for resource-aware NAS.

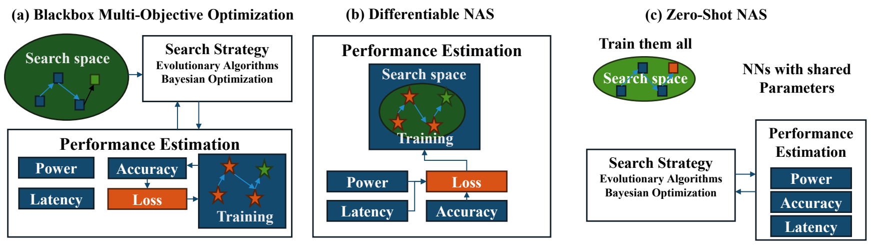


Figure 9.2 Three types of NAS are considered in the literature: (a) For black-box multi-objective optimization, many DNNs must be trained. However, optimisation results in a Pareto set of exact trade-offs between the different objectives. (b) Differentiable NAS returns only a single trade-off but is time and resource efficient because it optimizes the DNN during a single training run. (c) Zero-shot NAS allows fast DNN specialization for deployment goals, but the performance of the proposed trade-offs is only estimated.

9.3.1 Black-Box Multi-Objective optimization

The most common approach to solving HPO problems is black-box optimization. A traditional approach to tackle black-box (multi-objective) optimization is evolutionary algorithms (MOEA), one of the most common being NSGA-II [32]. As an example, [33] explore hyperparameter and compression parameter optimization using evolutionary algorithms for DNN deployment and combine it with pruning and quantization. A drawback of MOEA is that due to their population-based nature, they are sample inefficient, which can be time and resource consuming in the case of NAS, where many DNNs need to be trained and evaluated.

To overcome this problem, Bayesian black-box optimization [34] can be considered as a more sample-efficient alternative to MOEAs. Here, cheap to evaluate surrogate models, usually Gaussian processes (GPs), are used to approximate the black-box objective functions. During optimization, for each trial performed, a nested optimization is performed on the GPs, which have been previously fitted with an acquisition function based on all trials observed so far. The result of this nested optimization is in turn used to propose the next parameterization (or set of parameterizations) to the outer optimization loop, which in turn evaluates them on the actual objective functions, thus repeating the iterative optimization process.

Black-box optimization for HPO has also been regularly combined with reinforcement learning (RL). For example, [17] observed that a DNN architecture can be defined by a variable-length string, and that it is therefore possible to use a recurrent neural network (RNN) to generate such strings.

The authors call such an RNN a “controller”. By training the DNNs (child networks) generated from the strings emitted by the controller, accuracy on a data set can be obtained, which in turn can be used as a reward signal. This signal can then update the controller (i.e., the controller learns to improve its search over time through RL).

Another approach to neural network design with reinforcement learning is based on using a Q-learning agent that samples a CNN topology conditioned on a predefined behavior distribution and the agent’s prior experience [18]. The authors define the layer selection process as a Markov decision process. The agent sequentially selects layers via an ϵ-greedy strategy until it reaches a termination state. Like [17], the reward given to the agent is constructed from the validation accuracy of the generated network.

In addition, the authors use a replay buffer that stores the network topology and predict performance on a validation set for all sample models. This means that if an already trained model is resampled, it is not retrained, but the previously found validation accuracy is presented to the agent.

A major problem with all black-box HPO approaches is that they require many different DNNs to be trained and evaluated, which can become a time and resource consuming task. In addition, black-box optimization struggles with large parameter spaces, as the sampling complexity increases exponentially with each parameter added to the search space, an effect informally referred to as the “curse of dimensionality” [35].

9.3.2 Differentiable NAS

Unlike Black-box HPO, Differentiable NAS [21], [22] does not search over a discrete set of candidates architectures but instead relaxes the search space to be continuous. As a result, the target architecture can be optimized with respect to the performance of the validation/test dataset using gradient descent, solving the problem of having to evaluate many different architectures. Since gradient descent is more data efficient than regular black box search, differentiable NAS can achieve competitive performance to NAS while requiring much less computational resources.

The search space considered by differentiable NAS consists of computational cells as building blocks of the final architecture. Each cell is an acyclic graph containing an ordered sequence of nodes. Each node is a latent representation, e.g. a feature map, and directed edges connecting them are operations that transform the latent representation. This means that each intermediate node is computed based on all its predecessors.

Differentiable NAS can be seen as a bi-level optimization problem. The goal for the architecture search is (a) to find architecture, i.e., a combination of building blocks that minimise computational complexity and (b) a set of trained weights associated with the architecture that minimizes the training loss.

Another variant of differentiable NAS is hierarchical NAS [36], [8]. It not only searches at a cell structure level but also at network structure level, thereby formulating a hierarchical search space. The authors noticed that in modern CNN design the outer network level controls spatial resolution changes, while the inner cell level controls specific layer-wise computations. Regular NAS follows this principle as well but only automatically searches the inner cell level while the outer network level is designed by hand. This can become problematic for use cases which are sensitive to spatial resolution changes.

9.3.3 Zero-Cost neural architecture search

Like differential NAS, zero-cost NAS attempts to minimize the search costs associated with black-box optimization. However, unlike differential NAS, it focuses on a redundant learning strategy to minimize computational cost by using a well-trained weight-sharing model, i.e., a supernet from which many architecture variants, i.e., subnets, can be drawn at zero-cost. To find the best architecture α^* under given resource constraints, a zero-cost proxy metric is used that can score a subnet without training and validation on a dataset.

Several proxies have been discussed in research: [26] use the overlap of activations between data points in untrained networks, [27] quantify the expressiveness of an architecture, which they call Zen-Score, while other works used gradient-based zero-cost proxies such as the gradient norm to estimate model performance [28–20].

In recent research, zero-cost NAS approaches have been proposed to easily allow for specialization of network architectures to different target platforms. This allows for an efficient search of architectural variants for differently scaled edge systems.

For example, once-for-all$\sim$networks [16] provide an optimization strategy tailored to a multi-objective search space with different device and resource constraints. To achieve this, the authors decouple network training and search. During the training phase, the authors train a single network using a technique they call *Progressive Shrinking*. Once the single network is trained, different submodels are sampled from it and evaluated, requiring no further (re)training. These subnetworks can have different input resolutions, different kernel sizes or filters in their convolutional layers, and a different number of layers.

Another example of network specialization using zero-cost NAS is Pre-NAS [25], where the authors improve the efficiency of regular black-box optimization (which the authors call "one-shot NAS") by combining it with zero-cost NAS. To improve search efficiency, the authors propose to construct a reduced ("preferred") search space based on high-quality architectures selected under various resource constraints using zero-cost NAS, which is easier to search by black-box optimization than the original "full" search space.

9.3.4 Practical considerations

One of the biggest challenges that must be considered when using NAS as part of DNN deployment for embedded targets is the typically large amount of time required by the techniques we presented in this section. This is especially true for the black-box optimization which, while generally providing consistently good results, requires extensive sampling of the underlying search space which is the most expensive part in NAS. Other techniques presented in this chapter, namely Differential NAS and Zero-Cost NAS attempt to alleviate this problem, either by relaxing the problem to be differentiable so that it can be solved as part of network training or by relying on cheap-to-sample, often model-based surrogates of the expensive to sample search space. However, both techniques have caveats that should be considered: Differential NAS has been shown to have robust problems with optimal architectures often found to generalize poorly [24]. Furthermore, Differential NAS does not produce a Pareto front from which an optimal trade-off can be selected, but only a single architecture that is considered optimal after training. Many Zero-Cost NAS techniques, while allowing extremely fast DNN specialization for different deployment targets and resource constraints, report only simple statistics about found architecture candidates, not actually trained and evaluated networks. To summarize this section, choosing the best NAS strategy in the search for DNN architectures feasible for edge deployment is a trade-off between search time and the quality of the result obtained. For the best possible results, black-box optimization, ideally combined with Bayesian optimization, should be chosen, while for fast but less accurate results, zero-cost NAS techniques should be preferred.

9.4 Deep Neural Network Pruning

A common way to achieve compression of DNN to allow its use in resource-constrained environments is DNN pruning. Pruning is based on the idea that some of the trained parameters of a DNN can be removed, i.e. pruned, without significantly degrading the accuracy of the network. It is based on the understanding that most neural networks are overparameterized and have redundancy in their trained weights [37]. The approach is well known and has been discussed by several authors as early as the late 1980s [38]-[40].

While at first pruning was mainly proposed with better network generalization, less overfitting, and improved learning speed in mind, today it has become a popular technique to achieve DNN compression with minimal or no loss of accuracy.

The simplest way to prune a DNN is to set a subset of its trainable parameters to zero during training resulting in sparse parameter tensors. By setting parameters to zero, they are removed from the scope of the optimizer used to train the DNN. Therefore, the removed parameters no longer affect the training and validation loss of the network.

How pruning is performed during training can be roughly characterized by a few different parameters, which we will discuss in the remainder of this section.

9.4.1 Pruning granularity

There are two common techniques to perform DNN pruning: *Unstructured Pruning* and *Structured Pruning*. Their main difference is the granularity at which the techniques introduce sparsity into a DNN.

Unstructured Pruning introduces sparsity by removing individual neurons from the parameter tensors of a DNN while structured pruning removes entire structures of neurons. Typically, Structured Pruning targets structures such as filters [41] or channels [42] in convolutional layers. However, it is possible to extend Structured Pruning to rows or columns of linear layers as well.

An advantage of Structured Pruning over Element Pruning in the context of DNN compression, is that pruned structures can not only be set to zero but can be completely removed from the parameter tensor. This is not possible with element pruning, which produces matrices of arbitrary sparsity. However, Structured Pruning also has drawbacks: Removing structures from a DNN is much more invasive than Element Pruning, and its implementation is not as simple as setting individual values to zero. In regular feed-forward DNNs, layers are usually connected in a sequence. Information is passed through the network from top to bottom. Therefore, the output feature map of one layer becomes the input feature map of the next. By changing the shapes of the parameter tensors in these layers, the shapes of their input and output feature maps change as well. Therefore, removing structures from layers inherently means removing the data dependencies between them. This makes the implementation of Structured Pruning complex and requires a global view of the DNN structure. Structured Pruning becomes even more complicated in branching feed-forward networks, such as residual networks [1]. Here, data dependencies can span multiple layers run in parallel instead of just sequential layers.

It is possible to apply Structured Pruning and Element Pruning to a DNN together. [43] describe how they first apply Structured Pruning to convolutional layers of a DNN and then apply Element Pruning to the remaining structures. They call this approach *Hybrid Pruning*. They argue that Structured Pruning is a coarse-grained method, while Element Pruning is a fine-grained one. Therefore, Element Pruning can be used to remove connections that would otherwise be missed by Structured Pruning. In addition, Element Pruning and Structured Pruning can be used together by applying them to different layers or layer types in a DNN.

9.4.2 Pruning heuristics and sensitivity analysis

A major challenge in DNN pruning is to decide which elements or structures have the least impact on the validation loss of a DNN when removed.

The most accurate approach to achieving optimal pruning would be to remove each structure or element of a network one by one and evaluate its impact on the loss. This is referred to as the oracle criterion [44]. However, the downside of this strategy is that it is extremely resource and time consuming. So much so that it is inapplicable in most scenarios.

Therefore, other heuristics have emerged to approximate optimal pruning in a more computationally efficient manner. The process of quantifying the importance of parameter tensors of DNNs is referred to as *Sensitivity Analysis* [45]. Much research has been done in recent years to find good approximations for both Element and Structured Pruning techniques. In the following, we give an overview of some of the most used heuristics.

9.4.3 Magnitude or threshold based heuristics

Early work focused heavily on second-order derivative-based heuristics to approximate pruning. For example, [39] and later [40] propose calculating saliency scores for elements to determine their usefulness. They then use these scores to zero out a certain number of elements that they consider to be the least useful. [39] call this approach *optimal brain damage* (OBD). To avoid having to compute the saliency using the oracle approach, their heuristic approximates the objective function of a neural network using a Taylor series. However, this method requires additional computation.

As a result, more recent work has focused on simpler heuristics to approximate the usefulness of the trained parameters. The most common ones use simple threshold functions [45], [46] They consider a parameter useful if its absolute value is above a certain fixed value. If instead the value of the parameter is below or equal to the fixed value, it is marked as a candidate for removal. The heuristics are generally based on the understanding that a higher value creates a greater activation and is therefore more likely to have a significant impact on a layer's output than a smaller value. However, manually defining good thresholds requires extensive network analysis and is not intuitive. To solve this problem, most of the authors mentioned above approximate the distribution of values in the weight tensors of neural networks with a Gaussian distribution with a mean of zero. They then use the standard deviation of the distribution in combination with a scaling factor to automatically define pruning thresholds on a per-tensor basis. As a result, the only unknown variables remaining are the scaling factors. [45] derive these factors by increasing the sparsity in the layers of an unpruned baseline network and monitoring its accuracy changes. However, this is expensive because the baseline network must be trained first.

9.4.3.1 L-Norm heuristics

Threshold and magnitude-based approaches are not only useful for evaluating the importance of single parameters, but also for ranking complete parameter structures of neural networks. Therefore, a common way to evaluate the importance of such structures is to sort them by their l1-norm. [41] describe how they use the norm to estimate the overall size of filters in convolutional layers. Furthermore, using the norm also gives an estimate of how large the values in the resulting output feature maps will be. In addition, the l2-norm is sometimes considered as a means of classifying parameter structures in neural networks.

9.4.3.2 Gradient Ranked Heuristics

Another heuristic for structure pruning has been proposed by [44]. The authors present a parameter structure ordering heuristic based on Taylor series derived from the difference in loss when certain parameters are removed from a network. The approach is like the OBD heuristic described earlier in this section. This is also explicitly stated by the authors. Their resulting heuristic accumulates the product of the activation tensors and the gradients of the cost function with respect to the activation tensors. The gradients can be easily obtained via the backpropagation algorithm. Intuitively, the heuristic considers structures to be less significant if their parameters are close to zero and have a flat gradient.

9.4.3.3 Activation based heuristics

Another approach to approximate the importance of parameter structures is proposed by [47]. The authors analyze the sparsity of activation tensors during training. They do this by monitoring the average percentage of zero values (APoZ) found in outputs computed by rectified linear unit (ReLU) activation functions throughout the network. The higher the percentage of zero values generated by the ReLU, the less significant the structure is considered by the heuristic. [44] also mention the ApoZ approach in their paper. However, the authors caution that the approach may not perform well on early layers of neural networks. They note that early layers are typically trained to detect less defined features, resulting in denser activation tensors. Only for later layers, when feature detection has become more precise, do the activation tensors become sparse.

9.4.3.4 Relevance-based heuristics

Another criterion for pruning neural network structures is based on Layer-wise Relevance Propagation (LRP) [48] originating from the field of explainable AI (XAI). This approach assigns relevance scores to individual neurons in a neural network. Traditionally LRP serves as an XAI method to highlight the relevant parts of a given input to generate a local explanation for interpreting complex non-linear machine learning models, e.g. the relevancy of pixels of an input image can be highlighted in terms of a heat-map representing the influence of input parameters that are decisive for an image classification.

This relevance-based pruning criterion for pruning CNNs works by iteratively pruning the least relevant units, i.e. weights or filters of a network. The relevance scores are obtained via back-propagation from the output layers to the input layers and are assigned to all neurons of all layers, where the main characteristic of LRP is the backward pass through a network that can be computed efficiently. The relevance scores are then used to identify the least relevant units, which are the ones that can be pruned to discard all aspects of a network that do not contribute to the decision of a model. A pruned network can then be fine-tuned to recover possibly lost accuracy. This process is repeated until the desired level of compression is achieved. The LRP criterion is shown to be effective in reducing computational and parameter costs. Furthermore, based on experiments, the authors in [49] show that this pruning criterion performs consistently well or even better than state-of-the-art pruning criteria when model refinement and fine-tuning is applied.

9.4.4 Pruning schedule

A pruning schedule, sometimes called a pruning recipe, describes when, how often, and how much of a network is pruned during training. A very straightforward approach to schedule pruning is referred to as *one-shot pruning*. Its use has been described in early papers such as [39]. The general idea is to first train a network until it reaches a reasonable accuracy. Then, the whole network is pruned using a certain heuristic to remove the structures or elements with the lowest score. Based on this score, several of them are removed. Additionally, the authors found it beneficial to retrain the network after pruning.

One-shot pruning schemes are also described in more recent papers, for example by [44, 45]. However, both authors also distinguish another type of pruning schedule. They call it the *iterative pruning schedule*. This approach focuses very strongly on the idea of pruning and retraining a network several times instead of just once. Therefore, not all parameters are removed at once, but step by step over several pruning iterations as part of training. This allows the network to adapt more gradually to the decreasing set of trainable parameters.

Both authors also suggest specifying the number of trainable parameters to be pruned on a layer-by-layer basis rather than for the entire network. This is based on the understanding that the layers of a neural network may have different sensitivities to pruning. [44] show that some layers can be pruned much more aggressively than others before any degradation in the accuracy of a trained model becomes noticeable.

An extension to iterative pruning schedules is presented by [50]. The authors build on the idea that the number should be gradually increased instead of removing a constant number of parameters in each pruning iteration. They propose an algorithm that automatically increases the number of pruned parameters in a DNN over a range of n pruning steps based on a few predefined parameters. They call this the *Automated Gradual Pruning* algorithm. They state that the intuition behind the algorithm is “to prune the network rapidly in the initial phase, when the redundant connections are abundant, and to gradually reduce the number of weights pruned each time as fewer and fewer remain in the network”.

9.4.5 Practical considerations

Pruning is a core technique when it comes to fine-tuning the size of a DNN for a given problem, i.e. dataset, during training. The technique has proven to be extremely versatile, even when combined with other compression techniques such as quantization or NAS. To conclude this section, we would like to point out some practical considerations when using pruning as part of a DNN deployment pipeline: First, structured pruning will lead to immediate performance gains because pruned structures can be transparently removed from tensors, resulting in fewer computations, without any changes to the environment used to run the DNN at runtime. Unstructured pruning, on the other hand, only creates sparse tensors that require support from the environment to be executed efficiently at runtime. Second, the choice of a reasonable pruning schedule is of great importance, with iterative pruning using Automated Gradual Pruning generally providing better results. Third, simpler heuristics such as L-normed-based heuristics offer a good trade-off between accuracy and speed, while LRP based heuristics can provide more informed decisions and tend to produce an overall better pruning result at the cost of often higher compute times. Still, the outcome of pruning using LRP depends highly on the network and layer types used, and multiple parameters need tuning to make it applicable in the general case, see [51].

9.5 Quantization

One of the main methods to reduce the energy consumption of DNNs is to reduce their size and computational complexity using quantization [52]. Quantization of DNNs refers to the quantization of both the parameters (weights, biases, scaling factor ...etc.) and activations. DNN Quantization corresponds to some of or all the following benefits (depending on the model of quantization):

- Reduction of the NNs’ size which reduces the required energy to transmit and store the trained models when deployed on edge.

- Reduction in the complexity of the operations (mainly Multiply and accumulate MAC) leading to reduced power consumption of these operations,
- Reduction in size (bit-width) of the NN parameters and activations leads to improved power efficiency regarding memory access, required buffers and moving the data around on the edge target device.

9.5.1 Quantizers

First, we would like to define what quantization is, its different types and all the relative terms. The mathematical operation maps the input space into quantified values in the output space and can be divided, in general, into uniform and non-uniform. As the name suggests, non-uniform quantization produces non-uniform output values which can be represented by a floating-point and achieved using lookup tables. This is part of the model compression which will not be covered in this section, as it helps to reduce the model size for transmission or storage purposes but does lead to computational benefits or reduction in energy consumption.

On the other hand, uniform quantization is the most common method used in DNN quantization. Here, the input space is mapped uniformly to the output space according to the following generic equation.

$$\mathrm{x_Q}= \mathrm{clamp}\left(\frac{\mathrm{round(x)}}{\mathrm{s}}+\mathrm{z, a, b}\right) \ , \tag{9.1}$$

where $round()$ is a generic rounding function, s is a scaling factor, z is a zero-point, a and b are the minimum and maximum that define the range, and $clamp(x,\ a,\ b)$ is a clipping function defined as follows:

$$clamp(x,a,b) = \left\{ \begin{matrix} a & if\ \ x \leq a \\ x & if\ \ a < x < b \\ b & otherwise \end{matrix} \right\} = min(max(x,a),b) \ . \tag{9.2}$$

The implemented Rounding has an important impact on the quantization procedure and results. Here we list some of the commonly user rounding schemes:

- **Nearest**: $round(x) \ = \ \lfloor x \rceil$.
- **Round down**: $round\,(x) = \lfloor x \rfloor$.
- **Round up**: $round\,(x) = \lceil x \rceil$.
- **Stochastic rounding** [53]: stochastic rounding rounds the values up or down stochastically, $round\,(x) \ = \ \lfloor x \rfloor + (p > x - \lfloor x \rfloor)$ where p is a random variable sampled from a uniform distribution.

- **Adaptive rounding** [54]: where each individual value is rounded up or down to minimize the overall loss in the network.

Equation (9.1) defines an affine uniform quantization A.K.A. asymmetric uniform quantization. It can be further restricted by limiting $a = 0$ and $b = 2^{bw} - 1$ to produce integers with bit-width of bw. In addition to the benefits of reducing the operands bit-width and thus the required energy for memory access, the integer operations consume less energy [55] - [57] leading to even further energy savings. Additionally, a symmetric singed-integer quantization can be achieved by restricting $a = -2^{bw-1}$, $b = 2^{bw-1} - 1$ and $z = 0$. Finally, we can restrict this to represent fixed-point arithmetic by limiting s to powers of 2. This restriction helps to reduce energy consumption further as it does not require any multiply or divide operations but rather only bit shifts.

The quantization operation defined in equation (9.1) is irreversible, but an approximation can be achieved with:

$$\mathrm{x} \approx \overline{\mathrm{x}} = \mathrm{s}\left(\mathrm{x_Q} - \mathrm{z}\right) = \mathrm{s}\left(\mathrm{clamp}\left(\frac{\mathrm{round(x)}}{\mathrm{s}}, \mathrm{a}, \mathrm{b}\right) - \mathrm{z}\right) \ . \tag{9.3}$$

The quantization error/noise is then measured using $q_e = x - \overline{x}$. This error affects the correctness of the operations, but it was found that NNs are resilient to such errors and the performance is negligible or can be managed using different methods.

9.5.2 Granularity

After we defined and explained quantization and rounding schemes, we will look at how to apply these to neural networks. The basic building block of NNs is the matrix multiplication A.K.A multiply-accumulate (MAC) defined as $O = \overrightarrow{b} + \boldsymbol{WX}$, where $\overrightarrow{b}$ is the bias vector, $\boldsymbol{W}$ is the weights array, and $\boldsymbol{X}$ are the inputs. This operation is then approximated as follows:

$$\mathrm{O} = \overrightarrow{\mathrm{b}} + \mathbf{W_q X_q} \quad , \tag{9.4}$$

where $\mathbf{W_q}$ and $\mathbf{X_q}$ are the quantized tensors w.r.t $\boldsymbol{W}$ and $\boldsymbol{X}$. It is worth noting that the quantization parameters such as scaling factor, zero-point and range (bit-width) are set separately for the wights and activations. Equation (9.4) quantizes both the weights and activations it is also possible to quantize only the wights [58] – [59]. Weight-only quantization is generally simpler and easier but misses on a lot of the benefits of additionally quantizing the activations as the first only makes use of smaller size and less memory access but still requires floating-point operations. On the other hand, quantizing activations normally requires a dataset (unlabeled) to determine the appropriate range and scaling factors for each layer. The bias was left out of quantization as it normally required more bits to maintain high accuracy. E.g. the authors in [60] use 8 bits for the weights and activations and 32 bits for the bias.

The quantizers (scale, zero-point, range, bit-width ...etc.) can be specified for each layer (weights and activation) separately, this is referred to as per-layer quantization. Moreover, others have shown that reducing the granularity even further and performing channel-wise quantization (i.e. specifying a different quantizer per output channel/kernel) improves the results [61]–[63]. Others went beyond and quantized per-group of weights or activations [64]–[65]. The trade-of here is often that increasing granularity improves performance and accuracy but comes with an extra overhead in the form of extra parameters per quantizer and thus more memory access and required buffers.

9.5.3 Methods

Here, we discuss the two main categories to quantize NNs with minimal performance degradation and how do they compare. The first approach (Post-Training Quantization PTQ) is simpler as it takes in a trained NN and performs the quantization after training, while the second (Quantization-Aware Training QAT) is more complex as the quantization must be accounted for during training.

9.5.3.1 Post-Training quantization

Early approaches of quantizing NNs relayed on post-training quantization due to its simplicity. In these methods, the NN expensive computations (training) is performed once and then the trained NN can be quantized using different techniques. Using this method, NNs can be quantized to 8-bit weights [62] and [66] or even 4-bit weights [59], [63], [64], [67] and [68] or a mix of the two [69] with zero or minimal degradation in accuracy and performance.

Additional activation quantization is more complicated than weight-only quantization as it produces more errors that propagate though the network. E.g. In [70] the authors use 8-bit and kept the mixed activations with INT8/FP16 due to the high required dynamic range. Moreover, unlike weight quantization, activation quantization often requires training or validation datasets to collect the statistics about the data to determine the quantization range and scaling required. Yao et al. Managed to quantize the same large language model to 8-bit activations using Token-wise Quantization and knowledge distillation [69]. To determine the activation range, [63] uses the min/max and therefore no clipping occurs, while [63] uses Analytical Clipping for Integer Quantization ACIQ to clip the outliers and achieve a tighter range producing less overall rounding error. In [71], the authors get around the need for training or validation data by using distilled dataset that is designed to match the statistics of the original data. On the other hand, [72] learned a parameterized (range) activation function during training that can be used to clip the values. This approach, however, might not be considered as a post-training quantization approach as it requires changes to the training procedure.

9.5.3.2 Quantization-Aware training

In the previous section, we explained how a trained network can be quantized to 8-bit and even in some cases 4-bit weights and activations with minimal or no loss of accuracy. However, in most cases, quantizing the NNs beyond 8-bits yields significant performance degradation. This degradation can be mitigated if the quantization is accounted for during training. This is commonly referred to as Quantization-Aware Training QAT [60], [73] and [74]. Simply put, the NN graph is updated with the same quantizers from Section 6.1 into the parameters (weights, bias, . . . etc.), activations or both during training. This way the training of the NN will account for the quantization error producing more robust NNs.

The biggest question in incorporating the quantization operation into training comes from back-propagation/gradient descent and its reliance on

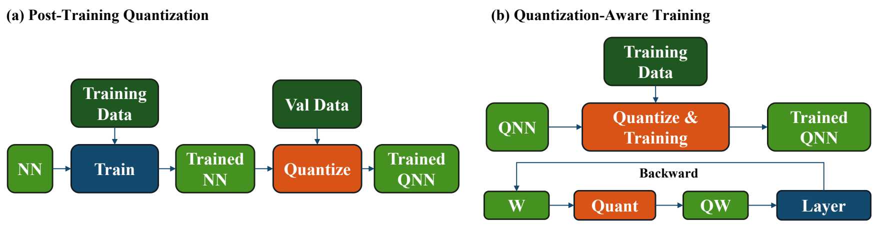


Figure 9.3 The workflow of the two main quantization methods: (a) Post-Training Quantization and (b) Quantization-Aware Training (including the straight throw estimator for the backpropagation)

derivatives. Mainly, the rounding operation in these quantizers has zero derivatives almost everywhere. Therefore, we cannot back propagate the error to update the networks parameters. To solve this question, Begio et al. analyzed and studied four different approximations [75]. Of which, the most promising is the straight-through estimator [76]. This estimator is quite common due to the simplicity as it replaces the derivative with the identity derivative, and it's proven to achieve the task of training.

9.5.4 Practical considerations

To make sure the review on NN quantization is comprehensive, a few practical considerations must be considered by the designers.

- **PTQ vs QAT**: PTQ is generally preferred over QAT when the goal is INT8 or even sometimes INT4 due to simplicity. However, QAT is preferred for sensitive applications or when severe quantization is required.
- **Floating-point vs Fixed-point quantization**: We have covered mostly Fixed-point quantization as it leads to reduction in model size and computational requirements. On the other hand, Floating-point quantization is helpful to reduce the model size for compression purposes but does not lead to a reduction in computation requirements. This is because the model must be uncompressed to the original values for inference.
- **Fusing layers**: The quantization operation can often be fused into the activation layer e.g. ReLU [63] to improve the quantization range as ReLU clips all the negative values anyways. Moreover, it can be fused into the normalization layer [69] and [77] to reduce the overhead cost as both operations require scaling and shifting the values.
- **Layer equalization**: It is observed that quantization to <INt8 often leads to degradation of accuracy, especially in depth-wise convolutions due to the high dynamic range within the layer [78]. One method to compensate for the high dynamic range is to use a channel-wise quantization as mentioned earlier which introduces complexity as each channel will require a different scale and bias parameters. Another solution is to perform cross layer equalization [78], Batch norm tuning [77], bias correction [79] or weight factorization [80].
- **Hyper-parameter selection**: Quantization of NNs and especially QAT adds few more parameters that need to be set/optimized for best performance (power/latency vs accuracy). This is not specific for NN

quantization, but it adds to the complexity for the optimization task. Refer to Chapter 4 for more details.

9.6 Cascaded Processing

In addition to the techniques mentioned above, it is also possible to reduce the energy footprint of a smart sensor system by introducing cascaded processing. Instead of using one big model, this approach relies on multiple models executed one after another (cf. Figure 9.4). By including exit points between models, the system can trigger the execution of the next one lazily, only using the energy needed for a particular decision. Each model in this chain should be designed in a way that builds upon the decision from the previous model(s). In this way, the processing of one model might suffice to conclude so that at every point, the execution may be aborted.

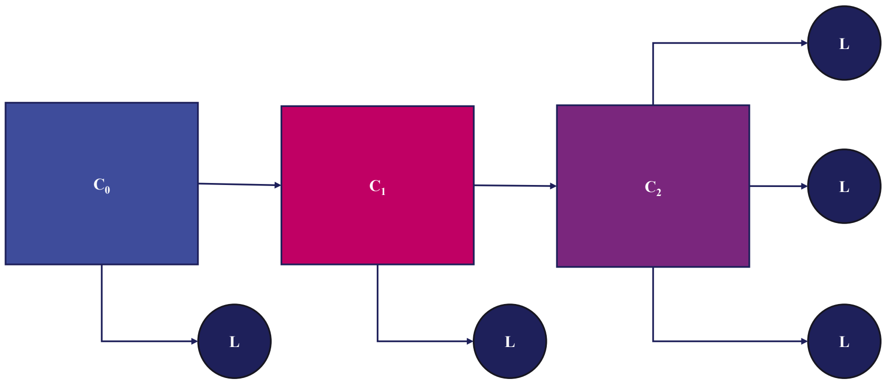


Figure 9.4 Example of simple cascade. Three classifiers are executed one after the other to classify three different labels. ⏎

This section will cover the current most relevant literature for cascaded processing, starting with traditional hierarchical models and their applicability to distributed sensor systems. Afterward, we will cover Early-Exit-NNs, which bring the same idea to neural networks.

9.6.1 Hierarchical systems

A typical example of a hierarchical system can be found in condition monitoring as depicted in Figure 9.5: Given a sensor system that should

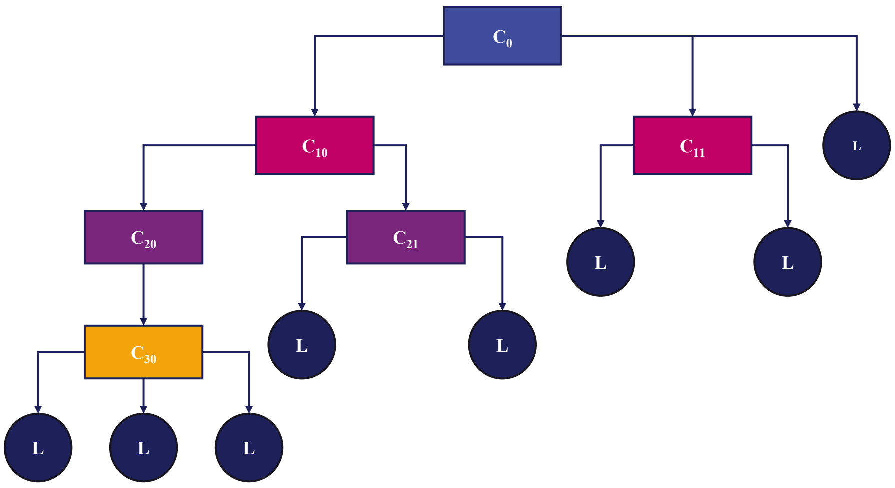


Figure 9.5 Complex hierarchy with multiple levels. The classification flow is separated into different branches. ⏎

classify the current machine status into different fault types and severities. The cascade's first model could detect anomalies, classifying abnormal and normal behavior. As this decision can be made quickly, the model should be small and can be run continuously. If a fault has been found, the following model classifies fault types. At this point, the processing can be stopped if the found anomaly has been a false positive. Otherwise, the second model is followed by an additional model for each fault type that classifies the severity of each fault. In that way, the models in the severity level can be specifically tailored towards a distinct fault type, leading to a more efficient model for the individual decision.

Building a foundation for hierarchical models, Silla and Freitas survey different applications for hierarchical classifications in [79]. While the possible application domains are interesting, this paper's most relevant work is how the authors generalize hierarchical classification in a unified framework. First, they introduce a class taxonomy as a poset $(C, \prec)$. C is a finite set of class concepts with a partial order $\prec$ over C, which is asymmetric, antireflexive, and transitive. This definition matches a directed acyclic graph (DAG) but is often simplified to a tree structure, where each node is limited to one parent node.Additionally, the authors cluster models for hierarchical classification problems into four groups: Local classifier per node (LCN), Local classifier per parent node (LCPN), local classifier per level (LCL), and global classifier (GC). LCN classifiers are found in models with a binary classifier at every node, whereas LCPN models can include multi-class classifiers. The LCL approach uses only one multi-class classifier per level, meaning no branching can be done with LCL classifiers. A GC uses just one big model to classify each label.Based on this definition, the authors categorize a hierarchical classification problem in a 3-tuple (Υ,Ψ,Φ) ,where Υ specifies the type of graph (DAG or Tree), Ψ whether a class label is associated with a single path (SPL) or multiple (MPL), and Φ the depth of labels from full depth (FD) to partial depth (PD). Additionally, they describe a categorization of different types of hierarchical algorithms as a 4-tuple $(\Delta,\Xi,\Omega,\Theta)$, where Δ in association with Ψ represents if the algorithm performs single path prediction (SPP) or multipath prediction (MPP), Ξ refers in line to Φ to mandatory leaf-node prediction (MLNP) or non-mandatory leaf-node-prediction (NMLNP). Ω reflects the model's ability to perform on tree or DAG-based problems as described with Υ. Θ reflects the classifier type (LCN, LCPN, LCL, or GC), as described above.

The here described framework was initially introduced to categorize any hierarchical classification problem. However, for the goal of reducing the energy footprint of a model, the possible model structure usually boils down to

$$(\Delta,\Xi,\Omega,\Theta) = (MPP, MLNP, T, LCPN) \ . \tag{9.5}$$

In this setting, problems like the above-mentioned condition monitoring use case can use hierarchical machine learning to save energy. In the example, the no-fault label could be reached by either the first or second classifier, making the model MPP. Additionally, MLNP is mandatory for saving energy, as we want to be able to stop calculating at any helpful point in the hierarchy. Even though a tree model is not entirely mandatory, allowing additional links to nodes other than child nodes often makes no sense and just improves complexity. Additionally, we want to be able to tailor each classifier in the hierarchy to a specific task. Allowing for additional links would bypass this principle. An LCPN classifier is the most flexible approach for structures, permitting asymmetric taxonomies with branching while allowing multi-class classifiers at each node.

With hierarchical predictions, another problem arises with partially correct predictions, which normally would result in a low score with standard metrics. In the condition monitoring example, the model might have correctly classified the fault but got the severity wrong. To approach this question, the authors recommend using the hierarchical precision, recall, and f-measure introduced by [80], where $\widehat{P_i}$ denotes a set consisting of the most specific classes predicted for the sample i as well as all its ancestors, and $\widehat{T_i}$ the set holding the true most specific classes for i. Considering the ancestors of class labels, these measures address the problem of partially correct classifications in the final scores.

$$hP=\frac{\sum_i |\widehat{P_i}\cap\widehat{T_i}|}{\sum_i |\widehat{P_i}|},\ hR=\frac{\sum_i |\widehat{P_i}\cap\widehat{T_i}|}{\sum_i |\widehat{T_i}|},\ hF=\frac{2\cdot hP\cdot hR}{hP+hR}\ . \tag{9.6}$$

The usability of hierarchical machine learning for energy saving, specifically for small sensor systems, has also been identified by [81] Here, the authors introduce a general description of a cascaded LCL model tailored towards lazily triggered stages. They differentiate between simple wake-up mechanisms, a two-stage hierarchy with only anomaly detection, and more sophisticated cascades. For the latter, they introduce a pass-on label, which a classifier can use to defer a decision. In this case, the decision is passed on to the next level, triggering further computations. A model's pass-on rate (POR) directly influences the hierarchical metrics described above. A POR of 1 would mean the decision is passed on to the final classifier for each sample. This leads to the hierarchical precision and recall metrics equaling their non-hierarchical counterparts for the last classifier.Additionally, the authors assign each stage a cost measure, as more memory is needed with increasing stages. Combining these thoughts about POR, Cost, and hierarchical metrics, the authors introduce an upper bound for optimizing a hierarchy wrt. computational costs and memory. With this upper bound and cost measure, they carry out proof of concept with synthetic test cases to optimize the number of stages in different scenarios. These scenarios are compiled to include class distributions ranging from equal to extremely skewed, where some classes are more likely than others. The authors conclude that optimisation can find lower-cost hierarchical models for more skewed class distribution. This effect becomes apparent when considering the locality of the most likely prediction point in the hierarchy. As some labels are more likely than others, detecting these in the early stages can save a lot of energy. For the condition monitoring example, this effect can be observed for the first two classifiers. If the no-fault label occurs most often, the hierarchy uses only the first (or, in some cases, the second) classifier. This leads to enormous energy savings as the computational complexity of each model should rise with the levels in the cascade. This phenomenon clearly shows the inherent data dependency of a hierarchical model regarding the reduction in energy consumption. The more uniform the data distribution of the underlying problem is, the less effective a hierarchical approach becomes.

One can also consider mixing different classifier types in the hierarchy to further elevate the benefits of hierarchical processing, as some decisions may need more complex models than others. However, with that idea in mind, the question of how to pick each model in the hierarchy arises. The authors in [82], [83] approach this optimization question with reinforcement learning. They train an agent to pick a model from a set of available models for each node in the hierarchy of an LCPN model. As the reward function, they use a cost measure based on the computational complexity of each possible model type, combined with the accuracy of the complete hierarchy. Therefore, the agent learns to make a trade-off between accuracy and energy consumption, which, in essence, leads to a multi-objective optimization problem. To improve that result, the computational complexity is replaced with a hardware-in-the-loop approach to accurately measure the resulting energy consumption during the agent's training in [84]. While this work mostly validates the findings of [82], it shows slightly different behavior of the found agents. With real measurements, the agent picks an MLP more often, while the agent with the complexity approximation tends to use Random Forests. This behavior might be linked to compiler optimizations and caching, which can be beneficial for some models. This result shows that energy measurements should always be included to some degree when optimizing for a constrained device, as a theoretical measure cannot cover every possible factor of real-life systems.

9.6.2 Distributed Computing

Another area where a hierarchical model can be beneficial is distributed computing for IoT devices. [85] introduces a distributed hierarchical inference approach, where some nodes in the hierarchy are computed locally and others in the cloud. This has the benefit of saving communication costs that might be unnecessary. They look at a scenario with multiple connected IoT appliances, which all have some parts of the information needed for home automation. In a classic approach, all these appliances would send data to a central hub or cloud infrastructure at every time step, which leads to an immense communication overhead. However, some decisions are possible without knowing the context of the other devices. In the hierarchical context, these decisions can be made with small models running locally on the IoT device. If this is not possible, the devices may pass the prediction to the cloud, triggering the next node in the hierarchy. The authors test this approach with three data sets from three application domains (urban energy demand, human activity, and server performance) with a decrease in system energy consumption of 62% for a taxonomy of MLPs.It should be noted that the authors used a DAG instead of a tree architecture for their models, as some nodes in the hierarchy connect to the same child node. This is because the cloud node(s) need information from multiple parent nodes because of the nature of IoT applications, where usually numerous sensors are required for a decision. Therefore, a typical hierarchy becomes a flipped tree and can only be described with a DAG.

The described benefits of IoT devices are tested in a case study in [86], where the authors use a physical system to measure the energy consumption of an LCL model for human activity recognition. The model uses multiple different classifiers in a cascade that are partially executed on the device. As a novelty, they introduce an offload controller that decides if the model should be run on the device or offloaded to a central hub. This controller consists of an additional classifier that has been trained on binary labels, which are extracted from the original multi-class problem. In addition, the controller uses only the features necessary to compute the next hierarchical layer that might be offloaded. This means that the controller performs a reduced complexity pre-evaluation of the problem solved by the following levels to decide if the computation needs to be offloaded. The paper shows energy reduction of 3 times over the baseline of a purely offloaded system, including communication costs, while improving accuracy slightly.

Taking the idea of split computing further, [87] introduces a dynamic split mechanism to split the computation between different models and a single DNN across multiple devices. In their proposed scenario, various IoT devices are connected to a hub. Based on the currently available network bandwidth, the introduced system can decide to hand off the computation to the hub. In addition, the authors train an agent with Q-learning to dynamically assign the network type (Wi-Fi, cellular, Bluetooth, etc.) for the data transfer of the remaining network layers. The authors can decrease the inference time by 13.5% compared to pure on-device execution, including communication time. However, it should be noted that no energy metrics are compared here, so while more time efficient, the shown approach might still draw more energy than pure on-device execution. Nonetheless, the approach should be considered when designing a hierarchical model.

9.6.3 Early-Exit Neural Networks

Most hierarchies mentioned before were constructed of classical machine-learning approaches like Support Vector Machines or Random Forests. Even though some architectures also used MLPs in their taxonomies, the work in [88] converts the concept of lazily triggered computations to Convolutional Neural Networks. Like the NMLP principle for classic hierarchies, the BranchyNet introduced here includes exit points in the neural network architecture (cf. Figure 9.6). The network can stop computing if a decision can be made in some earlier parts of the network based on a confidence score. Like nodes in the classical hierarchy, the authors define a branch as a non-overlapping subset of the complete network, with one entry point and multiple exit points. To train BranchyNet, the authors use a weighted formulation of the loss function which sums over the loss functions of each exit n with a weighting factor w_n. In this formulation, any arbitrary loss function that uses a one-hot encoded output vector for the prediction and ground truth $\hat{y}$ or y, respectively. θ describes the network's parameter of the branch before the exit point n. The choice of w_n influences the exit strategy of the network. Giving a high value for earlier exits causes more discriminative feature learning in the earlier branches. This leads to more early exiting but might compromise the accuracy. Contrarily, the network's accuracy increases with higher weights for the later exits, but exiting in the earlier stages becomes rare. During training, the network is trained as one unit without exiting early. However, for the inference, the authors use the entropy of the output at each exit point to determine if the calculation can be stopped. The computation is stopped if it is lower than a threshold T_n, reducing the inference time. Again, T_n is a hyperparameter that influences the trade-off between accuracy and inference time but might also be set automatically based on experimental results. The authors use three basic CNN architectures (LeNet, AlexNet, and ResNet) with added exit points to test the approach. They sweep through various values for T, which allows for a trade-off analysis of accuracy vs. runtime. For all three basic architectures, the BranchyNet approach leads to an optimal vector T that achieves a 2x-6x average runtime decrease while maintaining similar or superior accuracy compared to the non-branched networks. In addition to these results, the authors also observe a decrease in cache misses with more aggressive values for T. This leads to the conclusion that with a smart branch design, this behavior could be exploited to use the cache of a system effectively.

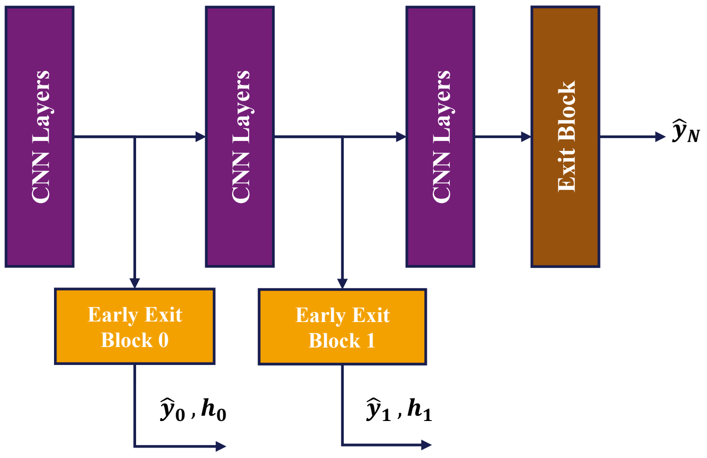


Figure 9.6 General structure of an Early-Exit CNN. After some CNN layers, exits can stop the computation based on a confidence metric ⏎

$$L_{branchynet}\left(\hat{y}, y, \theta\right) = \sum_{n=1}^{N} w_n L \quad \widehat{y_{exit_n}}, y; \theta \quad . \tag{9.7}$$

A similar approach is taken in [89]. The authors also introduce exit points in the network, but instead of the entropy, they use a learnable decision function $\gamma_n\left(\sigma_n\left(x\right)\right)$ as an exit strategy. Depending on the output of the previous CNN-layer σ_n, the decision function can decide to exit the execution. In addition, similar to mixed hierarchies with different model architectures, the authors propose a selection algorithm to fill branches in the early exit network with different architectures. They argue that easier classes might be classifiable with a simple AlexNet while others might need something more advanced like ResNet. With this approach, the authors achieve a speed-up of 1.5x to 7.53x, depending on the acceptable accuracy degradation. For example, the adaptive strategy can have a speed-up of around 2x compared to ResNet50 while staying in the range of 1% in terms of accuracy loss.

An additional implementation of an early exit strategy can be found in [90] called MSDNet. In their approach, branches are interconnected with a small classification network. The authors identify two problems with intermediate exits in standard architecture. Without any modifications, the early layers lack coarse-level features, which leads to worse classification results for earlier exists. Therefore, they introduce additional coarse features by adding parallel convolutional layers that calculate features on coarser scales. These features are concatenated with the ones from finer scales for intermediate classification. This addition leads to a substantial increase in accuracy for the exists.

The second problem is the interference of earlier classifiers with later ones. The authors noticed that adding intermediate classifiers for early exiting harms the accuracy at the final exit. Therefore, like a densely connected network, they add connections between earlier and later convolutional layers. In this way, the network should be able to forward information from earlier stages, which would otherwise be lost. After this change, the final exit becomes independent from the intermediate exits.

To optimize Early-Exit-NNs for on-device execution, [91] introduces an on-device transfer learning approach to fine-tune the intermediate exits in the field. The authors first train a generic MSDNet with equidistantly placed intermediate exits. On-device, they personalize the global pre-trained model by training only the intermediate exits. Due to the structure of MSDNet with bypass connection spanning the whole network, re-training the intermediate exits does not affect the final exit's performance. Therefore, they can use the classification result of the final prediction to teach the intermediate exists even if no labeled data is available in the field. They continue the personalization by tuning the threshold to decide if the calculation can be stopped at an intermediate exit. Like Multi-Objective Optimization, the authors use a small user-input data set to obtain a Pareto front to fine-tune the threshold. With this approach, the authors could measure a significant improvement in inference time and accuracy.

In addition, further field testing is done in [92] with a modified version of BranchyNet. Instead of solely relying on the entropy to decide on the next branch, the authors include energy-aware criteria based on the battery level. Only a shallow branch is computed if the remaining capacity is lower than a threshold. Otherwise, the decision is based on the entropy, as described in the BranchyNet paper. The authors also conduct thorough energy profiling and hardware-aware optimization steps that are out of the scope of this work. Nonetheless, it shows the importance of optimising the target platform, as the authors can deploy their approach to a processor with just 24kB SRAM and 128kB flash memory. In addition, they are also able to predict the battery size needed for their system.

9.7 Discussion

The highlighted literature shows that compared to flat classifiers, cascaded systems can decrease the latency by a substantial amount, which often translates into a decrease in energy consumption. The type of hierarchy can vary from simple cascaded structures to DAG or tree structures with multiple branches. The key here is always the uneven distribution of data in real-world systems, where some labels are either more frequent or easier to predict than others. This leads to a system that can abort computation early if certain criteria are met, which can be boiled down to the NMLNP property of hierarchical models. When interpreting nodes in a hierarchical model as sub-sets of a Neural Network, the NMLNP property can also be seen in early-exit-NNs, which also have the property to abort computation with intermediate exits. These architectures rely on course features computed by early CNN layers, which, for some labels, are enough to come to a correct prediction. Based on a confidence score, the system can stop computing at intermediate exits, which boils down to a hyperparameter optimization problem on thresholds. Depending on the thresholds, the network can trade between accuracy (regular usage of later exits) and energy efficiency (frequent usage of earlier exits).

Acknowledgements

This work was partially supported by the projects Center for Analytics-Data-Applications (ADA-Center) and MANOLO. ADA-Center is funded by the Bavarian Ministry of Economic Affairs, Regional Development and Energy within the framework of “BAYERN DIGITAL II” (20-3410-2-9-8). MANOLO is funded by the European Commission under the Horizon Europe programme Grant Agreement No.101135782.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” Conference on Computer Vision and Pattern Recognition, 2016.

[2] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” Journal of Machine Learning Research - Proceedings Track, vol. 9, pp. 249–256, 2010.

[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” Conference on Computer Vision and Pattern Recognition, 2017. 1003

[4] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” Conference on Computer Vision and Pattern Recognition, 2018.

[5] A. G. Howard, M. Sandler, L.-C. C. Grace Ch and, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, “Searching for mobilenetv3,” Conference on Computer Vision and Pattern Recognition, 2019.

[6] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in Conference on Computer Vision and Pattern Recognition, 2018.

[7] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in International Conference on Machine Learning, 2019.

[8] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, “Mnasnet: Platform-aware neural architecture search for mobile,” in Conference on Computer Vision and Pattern Recognition, 2019.

[9] B. Yang, G. Bender, Q. V. Le, and J. Ngiam, “Condconv: Conditionally parameterized convolutions for efficient inference,” in International Conference on Neural Information Processing Systems, 2019.

[10] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, “Deformable convolutional networks,” in International Conference on Computer Vision (ICCV), 2017.

[11] X. Zhu, H. Hu, S. Lin, and J. Dai, “Deformable convnets v2: More deformable, better results,” in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[12] J. Schmidhuber, “Learning to control fast-weight memories: An alternative to dynamic recurrent networks,” Neural Computation, 1992.

[13] D. Ha, A. M. Dai, and Q. V. Le, “Hypernetworks,” in International Conference on Learning Representations, 2017.

[14] B. De Brabandere, X. Jia, T. Tuytelaars, and L. Van Gool, “Dynamic filter networks,” in International Conference on Neural Information Processing Systems, 2016

[15] J. Bergstra, D. Yamins, and D. Cox, “Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures,” in International Conference on Machine Learning, 2013.

[16] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, “Once-for-all: Train one network and specialize it for efficient deployment,” in International Conference on Learning Representations, 2020.

[17] B. Zoph and Q. Le, “Neural architecture search with reinforcement learning,” in International Conference on Learning Representations, 2017.

[18] B. Baker, O. Gupta, N. Naik, and R. Raskar, “Designing neural network architectures using reinforcement learning,” in International Conference on Learning Representations, 2017.

[19] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, “Regularized evolution for image classifier architecture search,” in Proceedings of the AAAI conference on artificial intelligence, 2019.

[20] C. White, W. Neiswanger, and Y. Savani, “Bananas: Bayesian optimization with neural architectures for neural architecture search,” in Proceedings of the AAAI Conference on Artificial Intelligence, 2021.

[21] H. Liu, K. Simonyan, and Y. Yang, “DARTS: Differentiable architecture search,” in International Conference on Learning Representations, 2019.

[22] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, “Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search,” in IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2019.

[23] X. Shen, Y. Wang, M. Lin, Y. Huang, H. Tang, X. Sun, and Y. Wang, “Deepmad: Mathematical architecture design for deep convolutional neural network,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023.

[24] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter, “Understanding and robustifying differentiable architecture search,” in International Conference on Learning Representations, 2020.

[25] H. Wang, C. Ge, H. Chen, and X. Sun, “Prenas: Preferred one-shot learning towards efficient neural architecture search,” arXiv preprint arXiv:2304.14636, 2023.

[26] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley, “Neural architecture search without training,” in International Conference on Machine Learning. PMLR, 2021, pp. 7588–7598.

[27] M. Lin, P. Wang, Z. Sun, H. Chen, X. Sun, Q. Qian, H. Li, and R. Jin, “Zen-nas: A zero-shot nas for high-performance image recognition,” in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 347–356.

[28] M. S. Abdelfattah, A. Mehrotra, Ł. Dudziak, and N. D. Lane, “Zero-cost proxies for lightweight nas,” International Conference on Learning Representations (ICLR), 2021.

[29] N. Lee, T. Ajanthan, and P. H. Torr, "Snip: Single-shot network pruning based on connection sensitivity," International Conference on Learning Representations (ICLR), 2019.
[30] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," International Conference on Learning Representations (ICLR), 2020.
[31] C. White, M. Khodak, R. Tu, S. Shah, S. Bubeck, and D. Dey, "A deeper look at zero-cost proxies for lightweight nas," ICLR Blog Track, 2022.
[32] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-II," IEEE Transactions on Evolutionary Computation, 2002.
[33] Z. Wang, T. Luo, M. Li, J. T. Zhou, R. S. M. Goh, and L. Zhen, "Evolutionary multi-objective model compression for deep neural networks," IEEE Computational Intelligence Magazine, 2021.
[34] J. Knowles, "Parego: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems," IEEE Transactions on Evolutionary Computation, 2006.
[35] R. Bellman, "Dynamic programming," Science, vol. 153, no. 3731, pp. 34–37, 1966.
[36] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. Yuille, and L. Fei-Fei, "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," in IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2019.
[37] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," Advances in Neural Information Processing Systems, 2013.
[38] S. Hanson and L. Pratt, "Comparing biases for minimal network construction with back-propagation," Advances in Neural Information Processing Systems, vol. 1, 1988.
[39] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," Advances in neural information processing systems, vol. 2, 1989.
[40] B. Hassibi and D. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," Advances in neural information processing systems, vol. 5, 1992.
[41] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in International Conference on Learning Representations, 2016.

[42] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian, "Channel pruning via automatic structure search," in Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence, 2021, pp. 673–679.

[43] X. Xu, M. S. Park, and C. Brick, "Hybrid pruning: Thinner sparse networks for fast inference on edge devices," arXiv preprint arXiv:1811.00482, 2018.

[44] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in International Conference on Learning Representations, 2019.

[45] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," Advances in neural information processing systems, vol. 28, 2015.

[46] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," arXiv preprint arXiv:1510.00149, 2015.

[47] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," arXiv preprint arXiv:1607.03250, 2016.

[48] S.-K. Yeom, P. Seegerer, S. Lapuschkin, A. Binder, S. Wiedemann, K.-R. Müller, and W. Samek, "Pruning by explaining: A novel criterion for deep neural network pruning," Pattern Recognition, vol. 115, p. 107899, 2021.

[49] R. Xu, S. Luan, Z. Gu, Q. Zhao, and G. Chen, "LRP-based policy pruning and distillation of reinforcement learning agents for embedded systems," in IEEE 25th International Symposium On Real-Time Distributed Computing (ISORC), 2022, pp. 1–8.

[50] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," arXiv preprint arXiv:1710.01878, 2017.

[51] M. Kohlbrenner, A. Bauer, S. Nakajima, A. Binder, W. Samek, and S. Lapuschkin, "Towards best practice in explaining neural network decisions with lrp," in International Joint Conference on Neural Networks. IEEE, 2020, pp. 1–7.

[52] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," arXiv preprint arXiv:1806.08342, 2018. S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in International conference on machine learning. PMLR, 2015, pp. 1737–1746.

[53] M. Nagel, R. A. Amjad, M. Van Baalen, C. Louizos, and T. Blankevoort, "Up or down? adaptive rounding for post-training quantization," in International Conference on Machine Learning. PMLR, 2020, pp. 7197–7206.

[54] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC). IEEE, 2014, pp. 10–14.

[55] J. Johnson, "Rethinking floating point for deep learning," arXiv preprint arXiv:1811.01721, 2018.

[56] D. L. N. Hettiarachchi, V. S. P. Davuluru, and E. J. Balster, "Integer vs. floating-point processing on modern fpga technology," in 2020 10th Annual Computing and Communication Workshop and Conference (CCWC). IEEE, 2020, pp. 0606–0612.

[57] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in Proceedings of the European conference on computer vision (ECCV), 2018, pp. 365–382.

[58] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," arXiv preprint arXiv:2210.17323, 2022.

[59] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 2704–2713.

[60] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," arXiv preprint arXiv:1802.05668, 2018.

[61] J. H. Lee, S. Ha, S. Choi, W.-J. Lee, and S. Lee, "Quantization for rapid deployment of deep neural networks," arXiv preprint arXiv:1810.05488, 2018.

[62] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," Advances in Neural Information Processing Systems, vol. 32, 2019.

[63] M. G. d. Nascimento, R. Fawcett, and V. A. Prisacariu, "Dsconv: efficient convolution operator," in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 5148–5157.

[64] B. Darvish Rouhani, D. Lo, R. Zhao, M. Liu, J. Fowers, K. Ovtcharov, A. Vinogradsky, S. Massengill, L. Yang, R. Bittner et al., "Pushing the limits of narrow precision inferencing at cloud scale with microsoft

floating point," Advances in neural information processing systems, vol. 33, pp. 10 271–10 281, 2020.
[65] G. Xiao, J. Lin, M. Seznec, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," arXiv preprint arXiv:2211.10438, 2022.
[66] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev, "Low-bit quantization of neural networks for efficient inference," in 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW). IEEE, 2019, pp. 3009–3018.
[67] P. Wang, Q. Chen, X. He, and J. Cheng, "Towards accurate post-training network quantization via bit-split and stitching," in International Conference on Machine Learning. PMLR, 2020, pp. 9847–9856.
[68] Z. Yao, R. Yazdani Aminabadi, M. Zhang, X. Wu, C. Li, and Y. He, "Zeroquant: Efficient and affordable post-training quantization for large- scale transformers," Advances in Neural Information Processing Sys- tems, vol. 35, pp. 27 168–27 183, 2022.
[69] Y. Bondarenko, M. Nagel, and T. Blankevoort, "Understanding and overcoming the challenges of efficient transformer quantization," arXiv preprint arXiv:2109.12948, 2021.
[70] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer, "Zeroq: A novel zero shot quantization framework," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 13 169–13 178.
[71] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "Pact: Parameterized clipping activation for quantized neural networks," arXiv preprint arXiv:1805.06085, 2018. E. Park, S. Yoo, and P. Vajda, "Value-aware quantization for training and inference of neural networks," in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 580–595.
[72] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, "A white paper on neural network quantization," arXiv preprint arXiv:2106.08295, 2021.
[73] Y. Bengio, N. Lt'eonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," arXiv preprint arXiv:1308.3432, 2013.
[74] P. Yin, J. Lyu, S. Zhang, S. Osher, Y. Qi, and J. Xin, "Understanding straight-through estimator in training activation quantized neural nets," arXiv preprint arXiv:1903.05662, 2019.

[75] I. Hubara, Y. Nahshan, Y. Hanani, R. Banner, and D. Soudry, "Improving post training neural quantization: Layer-wise calibration and integer programming," arXiv preprint arXiv:2006.10518, 2020.

[76] M. Nagel, M. v. Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1325–1334.

[77] A. Finkelstein, U. Almog, and M. Grobman, "Fighting quantization bias with bias," arXiv preprint arXiv:1906.03193, 2019.

[78] E. Meller, A. Finkelstein, U. Almog, and M. Grobman, "Same, same but different: Recovering neural network quantization error through weight factorization," in International Conference on Machine Learning. PMLR, 2019, pp. 4486–4495.

[79] C. N. Silla and A. A. Freitas, "A survey of hierarchical classification across different application domains," Data Min Knowl Disc, vol. 22, pp. 31–72, 2011.

[80] S. Kiritchenko and F. Famili, "Functional annotation of genes using hierarchical text categorization," Proceedings of BioLink SIG, ISMB, 01 2005.

[81] K. Goetschalckx, B. Moons, S. Lauwereins, M. Andraud, and M. Verhelst, "Optimized hierarchical cascaded processing; optimized hierarchical cascaded processing," IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 8, 2018. [Online]. Available: http://www.ieee.org/publicationsstandards/publications/rights/index.html

[82] S. Adams, R. Meekins, P. A. Beling, K. Farinholt, N. Brown, S. Polter, and Q. Dong, "Hierarchical fault classification for resource constrained systems," Mechanical Systems and Signal Processing, vol. 134, p. 106266, Dec. 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0888327019304819

[83] S. Adams, T. Cody, P. A. Beling, S. Polter, and K. Farinholt, "Hierarchical classification for unknown faults; hierarchical classification for unknown faults," 2020 IEEE International Conference on Prognostics and Health Management (ICPHM), 2020.

[84] J. Wissing, S. Scheele, A. Mohammed, D. Kolossa, and U. Schmid, "Himledge – energy-aware optimization for hierarchical machine learning," in Advanced Research in Technologies, Information, Innovation and Sustainability, T. Guarda, F. Portela, and M. F. Augusto, Eds. Cham: Springer Nature Switzerland, 2022, pp. 15–29.

[85] A. Thomas, Y. Guo, Y. Kim, B. Aksanli, A. Kumar, T. S. Rosing, and U. S. Diego, "Hierarchical and distributed machine learning inference beyond the edge; hierarchical and distributed machine learning inference beyond the edge," 2019 IEEE 16th International Conference on Networking, Sensing and Control (ICNSC), 2019.

[86] F. Samie, L. Bauer, and J. Henkel, "Hierarchical classification for constrained iot devices: A case study on human activity recognition," IEEE Internet of Things Journal, vol. 7, pp. 8287–8295, 9 2020,

[87] J. Karjee, K. Anand, V. N. Bhargav, P. S. Naik, R. B. V. Dabbiru, and N. Srinidhi, "Split computing: Dynamic partitioning and reliable communications in iot-edge for 6g vision." IEEE, 8 2021, pp. 233–240. [Online]. Available: https://ieeexplore.ieee.org/document/9590311/

[88] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," vol. 0. Institute of Electrical and Electronics Engineers Inc., 1 2016, pp. 2464–2469.

[89] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, "Adaptive neural networks for efficient inference," in Proceedings of the 34th International Conference on Machine Learning - Volume 70, ser. ICML'17. JMLR.org, 2017, p. 527–536.

[90] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, "Multi-scale dense networks for resource efficient image classification," in International Conference on Learning Representations, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:3475998

[91] I. Leontiadis, S. Laskaridis, S. I. Venieris, and N. D. Lane, "It's always personal: Using early exits for efficient on-device cnn personalisation." Association for Computing Machinery, Inc, 2 2021, pp. 15–21.

[92] Y. Li, Y. Wu, X. Zhang, J. Hu, and I. Lee, "Energy-aware adaptive multi-exit neural network inference implementation for a millimeter-scale sensing system," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, pp. 849–859, 7 2022.

10

Scalable Sensor Fusion for Motion Localization in Large RF Sensing Networks

Fetze Pijlman[1,2]

[1]Signify, The Netherlands
[2]Eindhoven University of Technology, The Netherlands

Abstract

RF sensing in wireless communication networks is a novel approach for motion detection, but it faces challenges in accurately localising motion which is crucial for confinement in lighting control use cases. A probabilistic model enables motion localisation through sensor fusion. However, in probabilistic models the posterior estimations do not scale well with large networks as likelihoods of all possible system states need to be computed. It will be demonstrated that variational Bayesian techniques offer attractive approximations to the posteriors where the approximations require computational resources that scale with the number of nodes. This method is of general interest to large networks as it models nonlocal effects through localised updates.

Keywords: RF sensing, sensor fusion, probabilistic models, variational inference, confinement.

10.1 Motivation

In a smart lighting system, one can control the lighting by sensing motion through the wirelessly connected lights (nodes) which is also known as RF sensing. For some background on RF sensing the reader is referred to Liu [1] and Wu [2]. Fluctuations in RSSI values, Received Signal Strength

DOI: 10.1201/9788743808862-10

Indicator, between nodes are strong indicators of nearby motion. For lighting control, the system should be able to detect human arm motions and steps as specified by the NEN norms [3]. As major motions generally lead to large RSSI fluctuations, it becomes a challenge for lighting control to sense minor motions within a room while not being sensitive to major motions just outside the room. An example situation is given in Figure 10.1.

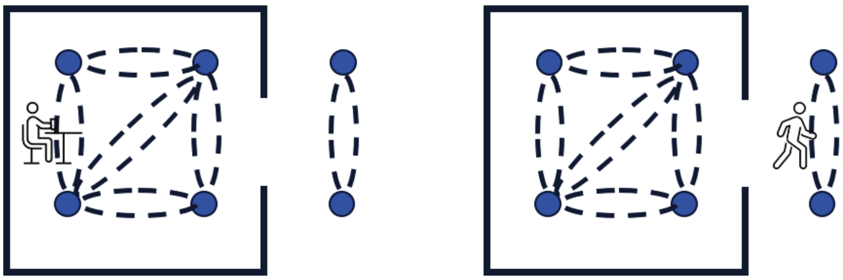


Figure 10.1 Downlights and indicated node pairs that are monitored for RSSI fluctuations. On the left a person working in an office and on the right a person walking on a corridor. ⏎

Probabilistic hidden Markov models are popular for motion sensing for lighting control. Typically, parameters such as motion rate during presence, average duration of presence, and probability of entering a room are introduced (see for example Papatsimpa [4]). These parameters can be used for propagating states and updating them with observations.

For modelling the signal variations due to motions in the surroundings, it is natural to extend the probabilistic models by modelling crosstalk. Motion directly underneath a node pair is visible, but a fraction of said motion is also visible to nearby other node pairs. The introduction of crosstalk complicates Bayesian inference as the number of different presence states now scale with 2^N where N is the number of presence areas. This essentially blocks the application of Bayesian inference for large networks. Another hurdle for large networks is the limited bandwidth for communication. In a mesh ZigBee network, the communication budget is 30-50 bytes per second per node, which limits the network size to about 50 nodes.

The goal of this paper is to apply a probabilistic model for motion/presence localisation by monitoring fluctuations in RSSI values between nearby node pairs. Using a variational Bayesian method, one can approximate posteriors by maximising the so-called free energy. As we will see, the maximisation is an iterative process that only involves local interactions. This leads to a scalable approach in which the computational power and memory scale with the number of nodes. Moreover, the calculations can be distributed over the nodes themselves, enabling a truly decentralised approach.

10.2 Spensor Fusion via a Probabilistic Model

For localising motion, a probabilistic model will be constructed (see Bishop [5], Murphy [6] and Morey [7] for an introduction to probabilistic models). The probabilistic model has the fluctuation of RSSI values of the node pairs as observational data *X*. By monitoring the fluctuations in RSSI values of a node pair one effectively obtains a motion sensor. In the remainder of the article node pairs will be referred to as sensors.

The probabilistic model contains hidden states of the physics in the various areas. These hidden states describe at each time instant *t* for each area: the presence $s(t)$ and the motion $m\left(t\right)$. In addition, at each time instant the visibility of a motion from an area *i* to a particular sensor *j* is described by $C_{ij}(t)$.

In order to simplify the calculation, one would like to make the Markov assumption which influences the choice for the value of the time-step. A simple model can be obtained by choosing the time-step to be larger than 1 second in which case motion at time *t* only depends on the presence state at *t* but not on a previous motion state. As the RSSI fluctuations *X(t)* only depend on the ***m(t)*** and the probability that a motion is visible *C* to a sensor, one can write the distribution as

$$
\begin{aligned}
&P\left(X\left(t\right),\boldsymbol{m}\left(t\right),\boldsymbol{s}\left(t\right),\boldsymbol{s}\left(t-1\right)|C(t)\right)\\
&= P(X(t)|\boldsymbol{m}(t),C(t))\prod_{i} P\left(\boldsymbol{m_i}\left(t\right)\mid \boldsymbol{s_i}\left(t\right)\right)P\left(\boldsymbol{s_i}\left(t\right)\mid \boldsymbol{s_i}\left(t-1\right)\right)\\
&\qquad P\left(\boldsymbol{s_i}\left(t-1\right)\right).
\end{aligned}
$$

Note that in the equation above it was assumed for simplicity that the presences in each area are uncorrelated. In Figure 10.2 an example of the interactions is given at a time instant.

With the Markov assumption, the posterior can now be determined by Bayesian inference

$$
\begin{aligned}
&P\left(\boldsymbol{s}\left(t\right),\boldsymbol{m}\left(t\right),C\left(t\right)|X\left(:t\right)\right) &(10.1)\\
&= \frac{P\left(X\left(t\right)|\boldsymbol{s}\left(t\right),\boldsymbol{m}\left(t\right),C\left(t\right)\right)}{P\left(X\left(t\right)\right)}P\left(\mathrm{s}\left(t\right),\boldsymbol{m}\left(t\right),C\left(t\right)|X\left(:t{-}1\right)\right),
\end{aligned}
$$

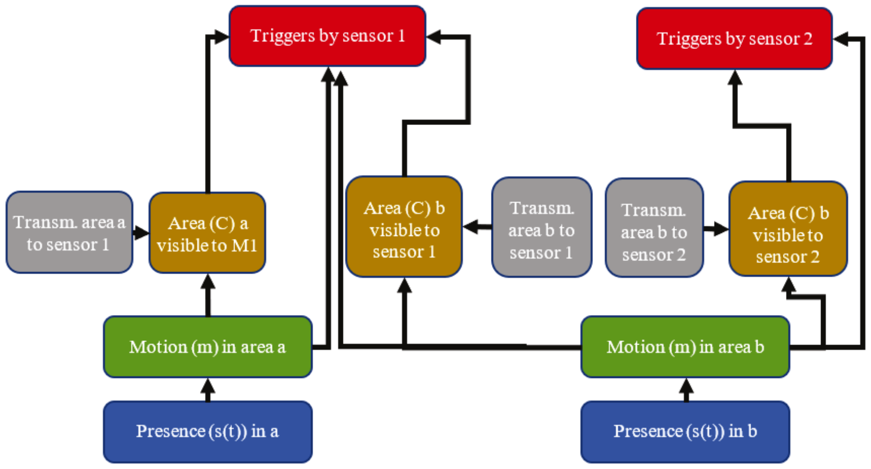


Figure 10.2 Bayesian network at a time instant showing the dependence between the various states and the sensor observations. ⏎

where X (: t) denotes all RSSI fluctuations up and including t, and X(t) denotes RSSI fluctuations at time t. Assuming the independence of presences in areas one finds for each area *i*

$$P\left(\boldsymbol{s_i}\left(t\right),\boldsymbol{m_i}\left(t\right)|X\left(:t{-}1\right)\right)=P\left(\boldsymbol{m_i}\left(t\right)\mid\boldsymbol{s_i}\left(t\right)\right)P\left(\boldsymbol{s_i}\left(t\right)|X\left(:t{-}1\right)\right)$$

$$P\left(\boldsymbol{s_i}\left(t\right)|X\left(:t{-}1\right)\right)=\underset{\boldsymbol{s_i}(t-1)}{}\quad P\left(\boldsymbol{s_i}\left(t\right)\mid\boldsymbol{s_i}\left(t-1\right)\right)P\left(\boldsymbol{s_i}\left(t-1\right)|X\left(:t{-}1\right)\right).$$

Note that the propagation of presence states involves a transition matrix $P\left(\boldsymbol{s_i}\left(t\right)\mid\boldsymbol{s_i}\left(t-1\right)\right)$.

The posterior in Equation 10.1 will be computed at each time-step using variational methods (see also Attias [8] and Jordan [9]). By maximising the free energy one can approximate the posterior $P\left(\boldsymbol{s}\left(t\right),\boldsymbol{m}\left(t\right),C\left(t\right)|X\left(t\right)\right)$ by a probabilitty distribution q in terms of the KL-divergence $D_{KL}(P||q)$. This yields

$$P\left(\boldsymbol{s}\left(t\right),\boldsymbol{m}\left(t\right),C\left(t\right)|X\left(t\right)\right)$$
$$\approx\underset{q}{\text{argmax}}\underset{\mathbf{s}(t),\mathbf{m}(t),C(t)}{}\quad q\left(\boldsymbol{s}\left(t\right),\boldsymbol{m}\left(t\right),C\left(t\right)\right)\left[\log\frac{P\left(X\left(t\right),\boldsymbol{s}\left(t\right),\boldsymbol{m}\left(t\right),C\left(t\right)\right)}{q\left(\boldsymbol{s}\left(t\right),\boldsymbol{m}\left(t\right),C\left(t\right)\right)}\right].\tag{10.2}$$

For notational simplicity the explicit time-dependence will be dropped in the remainder.

In order to solve the equation above one often makes additional assumptions on the probability distribution q. However, the mean-field approximation such as $q(\boldsymbol{s}(t), \boldsymbol{m}(t), C(t)) = q(\boldsymbol{s}(t))\, q(\boldsymbol{m}(t))\, q(C(t))$ is blocked as probability for motion during absence is zero. Zero probabilities lead to problems with the logarithms. Instead, we approximate q by

$$q(\mathbf{s}, \mathbf{m}, C) = \prod_{i,j} q(C_{ij}|m_i)\ q(m_i|s_i)\ q(s_i), \tag{10.3}$$

where $q(m_i = 1|s_i = 0) = 0$ and j refers to the sensor index. For notational simplicity the difference between the q's such as $q(\mathrm{s}_1)$ and $q(s_2)$ has been made implicit. Note that this approximation contains more distributions compared to the mean-field approximation; $q(m_i|s_i = 0)$ has no relation with $q(m_i|s_i = 1)$.

10.3 Update Equations

Equation 10.2 can be solved by iteratively maximizing the lower bound evidence. With each update for one of the $q's$ the lower bound evidence gets increased. Please see Figure 10.3 for a subset of the network that is relevant for determining the states. Please note that $m_{o_{ij}}$ is the visible motion towards sensor j excluding any motion from m_i.

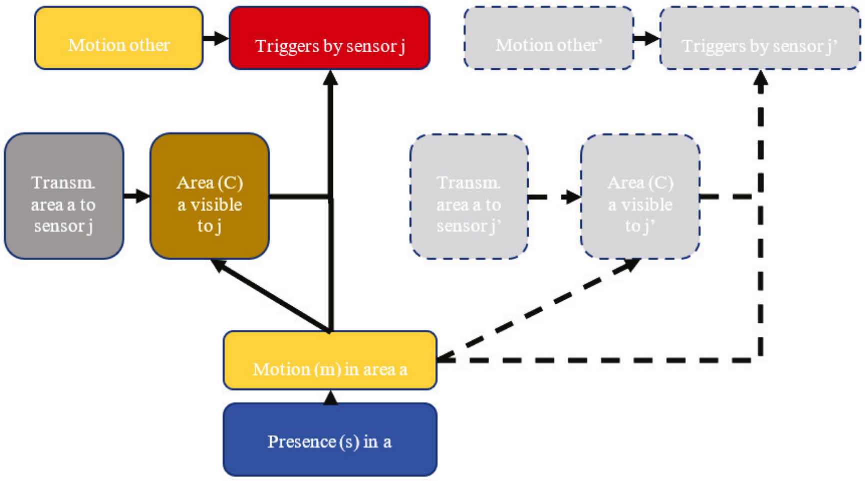


Figure 10.3 Isolated part of the network that is relevant for determining the states. ⏎

In the following subsections the update equations for the various factors will be determined by plugging Equation 10.3 in Equation 10.2. In the derivation the following relation will be often employed

$$\text{argmax}_{q(a)} \sum_{a} q\left(a\right) \left(\log P\left(X, a\right) - \log q\left(a\right)\right) = \frac{P(X|a)P(a)}{\sum_{a'} P\left(X|a''\right) P\left(a'\right)}. \tag{10.4}$$

The relation can be proven by using the method of Lagrange multipliers and treating $\sum_a q\left(a\right) = 1$ as a constraint (see also Arfken [10]).

It is instructive to work out the update equation in case a mean-field approximation would apply. Consider the example in Figure 10.4 and write $q\left(s,t,u\right)$ as $q\left(s\right) q\left(t\right) q\left(u\right)$. In this example the update equation for $q\left(t\right)$ would be

$$\begin{aligned} q\left(t\right) &= \text{argmax}_{q(t)} \sum_{s,t,u} q\left(s\right) q\left(t\right) q\left(u\right) \log \frac{P(X,s,t,u)}{q(s)q(t)q(u)} \\ &= \text{argmax}_{q(t)} \sum_{s,t,u} q\left(s\right) q\left(t\right) q\left(u\right) \log \frac{P(s|t)P(t|u))}{q(s)q(t)q(u)} \\ &= \text{argmax}_{q(t)} \sum_{t} q\left(t\right) \\ &\qquad \log \quad \underset{s}{} P(s|t)^{q(s)} \quad \underset{u}{} P(t|u)^{q(u)} \quad -\log q\left(t\right) \\ &= \frac{{}_{s} P(s|t)^{q(s)} \quad {}_{u} P(t|u)^{q(u)}}{\sum_{t'} \quad {}_{s} P\left(s|t'^{\prime}\right)^{q(s)} \quad {}_{u} P(t'|u)^{q(u)}} \end{aligned} \tag{10.5}$$

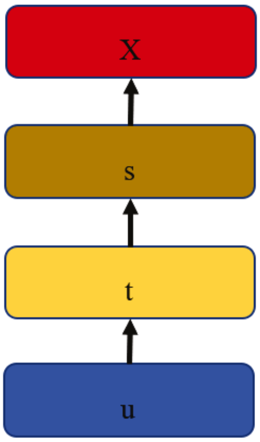


Figure 10.4 Example network with a mean-field assumption. ⏎

In case t only takes two values (0,1) and t is the opposite value of t then

$$\mathrm{q}\,(\mathrm{t}) = \mathrm{S}\left[\log\frac{q(t)}{1-q(t)}\ \right] = \mathrm{S}\ \ \sum_{\mathrm{s}} \mathrm{q}\,(\mathrm{s})\log\frac{P(s|t)}{P(s|t)} + \sum_{\mathrm{u}} \mathrm{q}\,(\mathrm{u})\log\frac{P(t|u)}{P(t|u)} \tag{10.6}$$

where S is the Sigmoid function.

Therefore, in order to update $q\,(t)$ one only need to know the approximated posteriors of its direct neighbours. Moreover, Equation 10.5 is just a Bayesian inference in which the prior and likelihood are products of powers by how much the surroundings appear which is according to expectation. In the following subsection the update equations will be derived for the various $q's$. In Figure 10.3 a part of the network is visualized which is useful when deriving the update equations.

10.3.1 Update equation for $q\,(C_{ij}|m_i\ =\ 0)$

Plugging Equation 10.3 in Equation 10.2 one finds

$$\begin{aligned} q\,(C_{ij}|m_i\ =\ 0)\ &= \underset{q(C_{ij}|m_i=0)}{\operatorname{argmax}} \sum_{s_i,m_i,m_{oij},C_{ij}} q\ \ m_{oij}\ \ q\,(s_i)\,q\,(m_i|s_i) \\ &\quad q(C_{ij}|m_i)\log\frac{P\big(X_j,m_{oij},s_i,m_i,C_{ij}\big)}{q\big(m_{oij}\big)q(s_i)q(m_i|s_i)q(C_{ij}|m_i)} \\ &= \underset{q(C_{ij}|m_i=0)}{\operatorname{argmax}} \sum_{s_i,m_{oij},C_{ij}} q(m_{oij})q\,(s_i)\,q\,(m_i=0|s_i) \\ &\quad q\,(C_{ij}|m_i{=}0)\log\frac{P\big(X_j,m_{oij},s_i,m_i{=}0|C_{ij}\big)P(C_{ij})}{q(m_{oij})q(s_i)q(m_i{=}0|s_i)q(C_{ij}|m_i{=}0)} \\ &= \underset{q(C_{ij}|m_i=0)}{\operatorname{argmax}} \sum_{s_i,C_{ij}} q\,(s_i)\,,q\,(m_i{=}0|s_i)\,q\,(C_{ij}|m_i{=}0) \\ &\quad \log\frac{P(C_{ij})}{q(C_{ij}|m_i{=}0)} \\ &= P\,(C_{ij}) \end{aligned} \tag{10.7}$$

The result above is according to expectation as one cannot estimate transparency/visibility in absence of a source m.

10.3.2 Update equation for $q\,(C_{ij}|m_i\ =\ 1)$

Plugging Equation 10.3 in Equation 10.2 one finds

$$\begin{aligned} q\,(C_{ij}|m_i\ =\ 1)\ &= \underset{q(C_{ij}|m_i=1)}{\operatorname{argmax}} \\ &\quad \sum_{s_i,m_i,m_{oij},C_{ij}} q(m_{oij})q\,(s_i)\,,q\,(m_i|s_i)\,q\,(C_{ij}|m_i) \end{aligned}$$

$$
\begin{aligned}
&\log\frac{P\left(X_j,m_{oij},s_i,m_i,C_{ij}\right)}{q(m_{oij})q(s_i)q(m_i|s_i)q(C_{ij}|m_i)}\\
&= \underset{q(C_{ij}|m_i=1)}{\operatorname{argmax}} \sum_{m_{oij},C_{ij}} q(m_{oij})q\left(C_{ij}|m_i{=}1\right)\\
&\quad\log\frac{P\left(X_j,m_{oij},m_i{=}1,s_i{=}1,C_{ij}\right)}{q(C_{ij}|m_i{=}1)}\\
&= \underset{q(C_{ij}|m_i=1)}{\operatorname{argmax}} \sum_{m_{oij},C_{ij}} q(m_{oij})q\left(C_{ij}|m_i{=}1\right) \qquad (10.8)\\
&\quad\log P\ \ X_j|m_{oij},m_i{=}1,C_{ij}\ \ -\log\frac{q(C_{ij}|m_i{=}1)}{P(C_{ij}|m_i{=}1))}\\
&=\frac{P(C_{ij}|m_i{=}1)\left(P\left(X_j|m_{oij}{=}0,m_i{=}1,C_{ij}\right)\right)^{q\left(m_{oij}=0\right)}}{\sum_{C'_{ij}}P(C'_{ij}|m_i{=}1)\left(P\left(X_j|m_{oij}{=}0,m_i{=}1,C'_{ij}\right)\right)^{q\left(m_{oij}=0\right)}}
\end{aligned}
$$

In the last line terms involving $q\ \ m_{oij}\ =\ 1$ can be omitted as they do not lead to differentiation for the different values of C_{ij}. The result can be understood as a Bayesian inference where the amount of observation is determined by $q\ \ m_{oij}\ =\ 0$.

10.3.3 Update equation for $q\left(m|s\ =\ 1\right)$

Plugging Equation 10.3 in Equation 10.2 one finds

$$
\begin{aligned}
&q\left(m_i|s_i\ =\ 1\right)\\
&= \max_{q(m_i|s_i=1)} \sum_{j,s_i,m_i,m_{oij},C_{ij}} q\ \ m_{oij}\ \ q\left(s_i\right)q\left(m_i|s_i\right)q\left(C_{ij}|m_i\right)\\
&\quad\log\frac{P\left(X_j,m_{oij},s_i,m_i,C_{ij}\right)}{q\left(m_{oij}\right)q(s_i)q(m_i|s_i)q(C_{ij}|m_i)}\\
&= \max_{q(m_i|s_i=1)} \sum_{j,m_i,m_{oij},C_{ij}} q\ \ m_{oij}\ \ q\left(m_i|s_i{=}1\right)q\left(C_{ij}|m_i\right)\\
&\quad\log\frac{P\left(X_j,m_i,C_{ij}|m_{oij},s_i{=}1\right)}{q(m_i|s_i{=}1)q(C_{ij}|m_i)}\\
&=\frac{{}_{j,m_{oij},C_{ij}}\ P(m_i|s_i{=}1)\ \ \frac{P\left(C_{ij}|m_i\right)}{q\left(C_{ij}|m_i\right)}P\left(X_j|m_i,C_{ij},m_{oij}\right)\ \ ^{q\left(C_{ij}|m_i\right)q\left(m_{oij}\right)}}{\sum_{m'_i}\ \ {}_{j,m_{oij},C_{ij}}\ P(m'_i|s_i{=}1)\ \ \frac{P\left(C_{ij}|m'_i\right)}{q\left(C_{ij}|m'_i\right)}P\left(X_j|m'_i,C_{ij},m_{oij}\right)\ \ ^{q\left(C_{ij}|m'_i\right)q\left(m_{oij}\right)}}\\
&=\frac{P(m_i|s_i{=}1)\ \ {}_{j,C_{ij}}\ \ \frac{P\left(C_{ij}|m_i\right)}{q\left(C_{ij}|m_i\right)}\ \ ^{q\left(C_{ij}|m_i\right)}\left(P\left(X_j|m_i,C_{ij},m_{oij}\right)\right)^{q\left(m_{oij}=0\right)}}{\sum_{m'_i}P(m'_i|s_i{=}1)\ \ {}_{j,C_{ij}}\ \ \frac{P\left(C_{ij}|m'_i\right)}{q\left(C_{ij}|m'_i\right)}\ \ ^{q\left(C_{ij}|m'_i\right)}\left(P\left(X_j|m'_i,C_{ij},m_{oij}\right)\right)^{q\left(m_{oij}=0\right)}}
\end{aligned}
$$

(10.9)

The result above contains terms that originate from complexity $P\left(C_{ij}|m_i\right)\,/\,q\left(C_{ij}|m_i\right)$.

10.3.4 Update equation for $q\,(s)$

Plugging Equation 10.3 in Equation 10.2 one finds

$$
\begin{aligned}
q\,(s_i) =& arg \max_{q(s_i)} \quad \sum_{j,s_i,m_i,m_{oij},C_{ij}} q \;\; m_{oij} \;\; q\,(s_i)\, q\,(m_i|s_i)\, q \;\; C_{ij}|m_i \\
& \log \frac{P \;\; X_j, m_{oij}, s_i, m_i, C_{ij}}{q \;\; m_{oij} \;\; q\,(s_i)\, q\,(m_i|s_i)\, q \;\; C_{ij}|m_i} \\
& \propto P\,(s_i) \prod_{j,m_i,m_{oij},C_{ij}} \\
& \left[\frac{P \;\; X_j|m_{oij}, m_i, C_{ij} \;\; P \;\; m_{oij} \;\; P\,(m_i|s_i)\, P \;\; C_{ij}|m_i}{q \;\; m_{oij} \;\; q\,(m_i|s_i)\, q \;\; C_{ij}|m_i}\right]^{q\left(m_{oij}\right)q(m_i|s_i)q(C_{ij}|m_i)} \\
& \propto P\,(s_i) \prod_{m_i,j} \quad P \;\; X_j|m_{oij}{=}0, m_i, C_{ij}{=}1 \quad ^{q\left(m_{oij}=0\right)q(m_i|s_i)q(C_{ij}=1|m_i)} \\
& \quad \frac{P\,(m_i|s_i)}{q\,(m_i|s_i)} \quad ^{q(m_i|s_i)} \times \prod_{C_{ij}} \left(\frac{P \;\; C_{ij}|m_i}{q \;\; C_{ij}|m_i}\right)^{q(C_{ij}|m_i)} .
\end{aligned}
\tag{10.10}
$$

10.4 Conclusions and Discussion

In this article, a probabilistic model is developed for localising motion in a wireless IoT network. A hidden Markov model is employed to propagate presence states over time using a transition matrix, forming a new prior for presence at the next time instant. At each time instant, a variational Bayesian approach was utilised to translate sensor data (RSSI values) into posterior presence and motion states, accounting for potential crosstalk. A first preliminary result is given in Figure 10.5.

The derived equations do not need to be applied to states in a particular order as each order will improve the lower bound evidence. This enables a distributed approach in which nodes apply asynchronous updates by taking information from their neighboring nodes. However, the iterations do increase network traffic, and it remains to be seen whether the network can handle the increase in communication.

The number of iterations needed for convergence will generally depend on how sensitive RSSI fluctuations of node pairs are to motion in the vicinity as the probabilistic model handles nonlocal effects via iterations involving

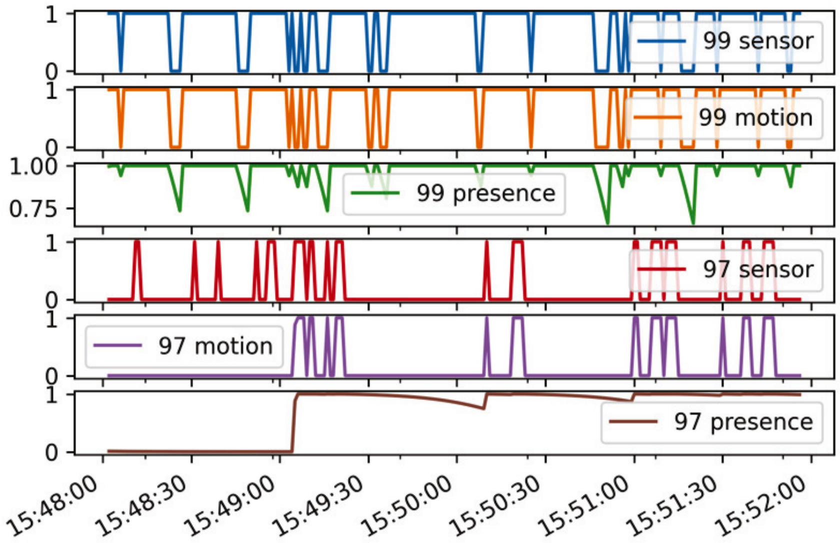


Figure 10.5 First results for two areas being named 99 and 97. The model parameters were chosen such that area 99 is a corridor while area 97 is a meeting room. In the figure one can see that isolated sensor events of 97 before 15:49 do not lead to presence in area 97 as it is more likely they originated from area 99. After probablity for presence is high in area 97 then isolated sensor events of 97 at 15:50:15 do lead to an increase in presence in area 97. ⏎

local updates. Future experimental studies are necessary to test the model for convergence and to quantify the improvements in motion localisation.

Centralized or distributed, the number of iterations needed for convergence may be reduced when considering techniques such as gradient descent. Using gradient descent one updates a group of states by a small amount. Iteratively updating (see also Kuusela [11]) may lead to faster convergence compared to iteratively updating each state.

Although the use of variational Bayesian techniques led to simple update equations, a problem with the approach is that the obtained posteriors are often narrower than what would be justified. This is an effect of minimizing the reversed KL-divergence. Obtaining over-confident posteriors is especially problematic for crosstalk parameters C_{ij} as it highly influences future updates. Therefore, it is recommended to use the whole time-series when updating model parameters with variational Bayes. As an alternative one may consider in future studies expectation propagation as proposed by Minka [12] in which sequential updates should be feasible. Similarly, it would be of interest to benchmark the obtained iterative algorithm with message passing algorithms such as RxInfer [13].

Acknowledgements

Jean-Paul Linnartz, Michel Wijker, Bert de Vries, Marco Haverlag, and Peet van Tooren are acknowledged for fruitful discussions.

EdgeAI "Edge AI Technologies for Optimised Performance Embedded Processing" project has received funding from Key Digital Technologies Joint Undertaking (KDT JU) under grant agreement No 101097300. The KDT JU receives support from the European Union's Horizon Europe research and innovation program and Austria, Belgium, France, Greece, Italy, Latvia, Luxembourg, Netherlands, Norway.

References

[1] H. Liu, H. Darabi, P. Banerjee, J. Liu, "Survey of Wireless Indoor Positioning Techniques and Systems", IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, 2007.

[2] Z-H. Wu, Y. Han, Y. Chen, K. J. R. Liu, "A Time-Reversal Paradigm for Indoor Positioning System", IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, 2015.

[3] "Methods of measurement and declaration of the detection range of detectors - Passive infrared detectors for major and minor motion detection", NEN-EN-IEC 63180, 2020.

[4] C. Papatsimpa, J.P. Linnartz, "Distributed Fusion of Sensor Data in a Constrained Wireless Network", MDPI Sensors, (2019).

[5] C. Bishop, "Pattern Recognition and Machine Learning", Springer, 2006.

[6] K.P. Murphy, "Machine learning - a probabilistic perspective". *The MIT Press*, 2012.

[7] R.D. Morey, J.W. Romeijn, J.N. Rouder, "The philosophy of bayes factors and the quantification of statistical evidence". *Journal of Mathematical Psychology*, 2016.

[8] H. Attias. "A variational bayesian framework for graphical models". *neurIPS* 1999.

[9] M.I. Jordan, Z. Ghahramani, T.S. Jaakola, L.K. Saul, "An introduction to variational methods for graphical models". *Springer* 1999.

[10] G.B. Arfken, H.J. Weber, “Mathematical Methods for Physicists.” *Academic Press*, 1995.

[11] M. Kuusela, T. Raiko, A. Honkela, J. Karhunen, “A Gradient-Based Algorithm Competitive with Variational Bayesian EM for Mixture of Gaussians”. Proceedings of International Joint Conference on Neural Networks 2009.

[12] T. Minka, “Expectation Propagation for approximate Bayesian inference”, Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence, 2001.

[13] D.V. Bagaev, A. Podusenko, B. de Vries, “RxInfer: A Julia package for reactive real-time Bayesian inference”. The Journal of Open Source Software, 2023.

11

Multi-Step Object Re-Identification on Edge Devices: A Pipeline for Vehicle Re-Identification

Tomass Zutis, Peteris Racinskis, Anzelika Bureka, Janis Judvaitis, Janis Arents, and Modris Greitans

Institute of Electronics and Computer Science (EDI), Latvia

Abstract

In modern computer vision tasks, the ability to identify and track objects across different scenes and environments has become important for numerous applications, especially in transportation. Inspired by this need, we propose a method that leverages a multi-step process focused on extracting and using object features for object re-identification.The proposed pipeline includes the following steps: detecting an object, converting its features into a vector embedding, storing this embedding in a vector database, and then querying the database to find the same or similar objects based on their feature embeddings. This approach enables us to identify the same object across different images or cameras, even in varying locations. This is essential in scenarios like Vehicle Re-Identification. For such scenario, implementing this process on edge devices is crucial. Therefore, ways to tailor the pipeline and its outputs for edge devices are outlined. The paper details the pipeline's structure along with the experimental setup demonstrating its application, particularly in vehicle re-identification. The pipeline achieves 70-80% re-identification precision when dealing with vehicle images from our network cameras and above 70% Rank-1 accuracy when dealing with a CityFlow video track scenario.

Keywords: vehicle re-identification, feature extraction, re-identification pipeline, computer vision, traffic monitoring.

DOI: 10.1201/9788743808862-11

11.1 Introduction

Object recognition in photos and videos has long been a key area of research, with significant advancement driven by computer vision [1]. Initially, image classification addressed the question, “What is in the image?” followed by object detection answering, “Where and what are the objects?”—largely thanks to Convolutional Neural Networks (CNNs) and their variants [2][3]. This paper focuses on object re-identification, which is a sub-problem of image retrieval. Object re-identification aims to distinguish instances of the same class, such as vehicles and persons (as illustrated in Figure 11.1), across different scenes despite changes in conditions like scene, lighting , or object pose [4][5]. However, truly impactful re-identification research in 2024 must support edge computing.

Because of the increase in processing latency and big data, edge computing is becoming essential for real-time re-identification, especially in applications like traffic monitoring, where network cameras should transmit only processed results [6][7]. The volume of available video footage, and the influx of sensory data have made the large-scale accumulation of big data inevitable [8]. This is why fully automated systems are needed for processing data and re-identifying objects in smart cities. Manual processing by humans is not feasible, especially when real-time decisions are required. We propose a pipeline to handle these tasks of efficiently processing live video feeds and identifying objects across multiple scenes. The novelty of this research lies in the integration of state-of-the-art methods into a unified pipeline, tested on

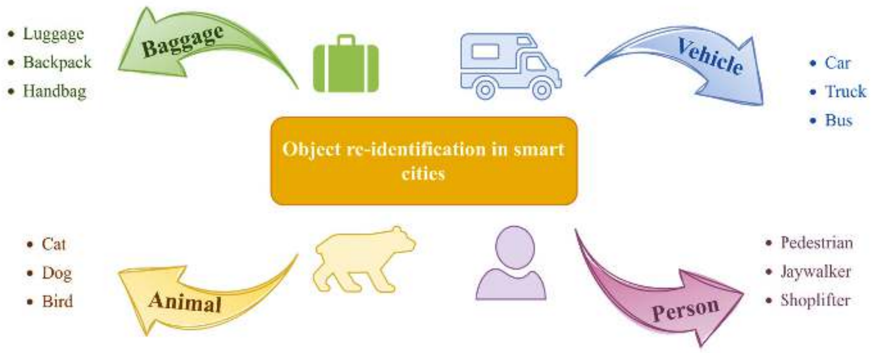


Figure 11.1 Re-identifiable classes in smart city environments. ⏎

real-world vehicle re-identification scenarios that fit into smart city initiatives and traffic management using edge computing solutions.

11.2 Related work and state of the art

11.2.1 Object detection

CNNs have been incredibly useful in computer vision tasks, including object detection. YOLO - the “You Only Look Once” model is one of the best performing and regularly updated choices. The latest YOLO v8 version has shown significant improvements in accuracy and speed [9], which is crucial for real-time applications like ours.

11.2.2 Object feature extraction

Feature extraction maps an image from its colour space to a higher-dimensional feature space [10]. Before feature extraction, multiple pre-processing stages are usually employed: normalization, thresholding, binarization, resizing and others. We can expect a model to extract colour, texture, shape, motion and localization features. A model learns intra-class variations as features when training a model on objects of one class like face features in the case of facial recognition.

As of 2024, CNNs became the predominant choice in object feature extraction thanks to their strong representation power and their ability to learn deep invariant embeddings [11].

11.2.3 Vehicle re-identification

There are multiple benchmarks for Vehicle Re-ID. MBR4B-LAI model [12] tops the VeRi-776 benchmark, “A strong baseline” model [13] tops the CityFlow benchmark and the VehicleNet model [14] is best at the VeRi benchmark. We pay particular interest to [14] by Zheng *et al.* because of the baseline model that is applicable to all types of object re-identification and feature extraction. In [15] the same author first introduces us to their baseline model and its architecture and demonstrates its capabilities specifically in pedestrian re-identification. In further papers, however, Zheng *et al.* demonstrate tailoring of this baseline to vehicle re-identification [16] and person re-identification [11]. We underline this baseline models usefulness by the versatility of its use in publications, its open code base and customizability and its entry into most benchmarks. The model is successful in Rank1 precision, according to the benchmark results on Papers with Code [17]. The model is implemented in Pytorch and is based on ResNet50 pre-trained on ImageNet, although this backbone is customizable. We use the surrounding project code for this model available on [18]. The author has provisioned tools to train, finetune, test and visualize the results of the inference process. We further refer to this model as "baseline model".

11.2.4 Available datasets

The VehicleID introduced in [19] has each image associated with a vehicle ID. It is the dataset with one of the biggest unique ID collections and features pictures with different resolutions and quality of visibility as well as vehicles in motion state. Vehicles are mostly seen, however, from the front and the back only. The VeRi-776 was introduced in [20]. Each image is attached with vehicle ID, bounding box, type, colour, brand. The dataset has a smaller set of unique id's compared to the VehicleID, but has better quality pictures, better visibility and vehicles from all angles not just back and front. The CityFlow dataset introduced in [21] is a traffic camera dataset consisting of synchronized HD videos from 40 cameras. The quality of images is high, and vehicles can be seen from many angles. The difference between this and the previous two widely used datasets can be seen in Figure 11.2.

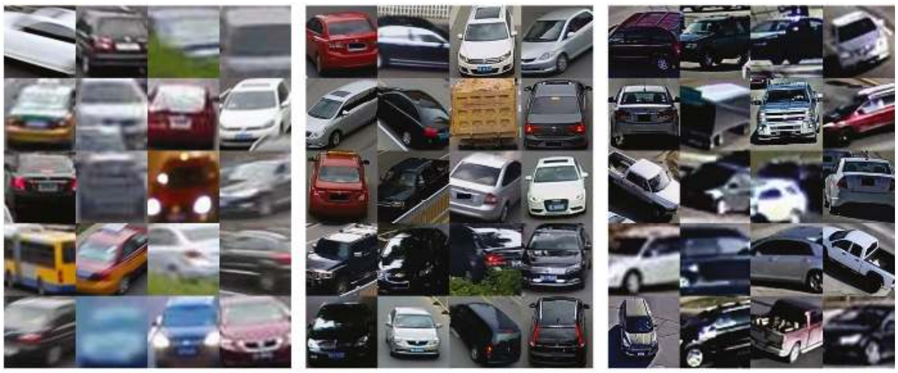


Figure 11.2 The difference between the distribution of vehicles for the (from the left) VehicleID, VeRi and CityFlow datasets. ⏎

The VehicleX synthetic data can supplement these three datasets. It contains generated images with domain adaptation from VehicleID, VeRi-776 and CityFlow [22].

11.2.5 Edge implementation

Requirements for a real-time system include implementing a network that follows the edge-computing paradigm. The edge-computing paradigm means that the video analytics run directly on the device and only the processed results and analytics are transmitted. [6] have developed a pilot project where they use mobility trackers using live CCTV feeds, with twenty sensors deployed over the city with the objective of citywide traffic monitoring in real-time. The devices had the ability to transmit the outputs either over Ethernet or LoRaWAN networks and had two main components: 1.) an NVIDIA Jetson TX2 high performance and power efficient embedded computing device with special units for accelerating neural network computations used for image processing and running Ubuntu 16.04 LTS and 2.) a Pycom LoPy 4 module handling the LoRaWAN communications.

11.3 Proposed methodology

We propose a sequence of processes for object re-identification in the context of a smart city environment. It takes in video frames from camera *x*, detects the vehicles in the frames, crops images of the vehicles and saves them, turns the images into feature embeddings and saves them in a vector database. The same process is repeated for a different camera *y* ... *z*, so that vehicles can be re-identified from camera *x* to *y* ... *z* or vice versa. This has been illustrated in Figure 11.3.

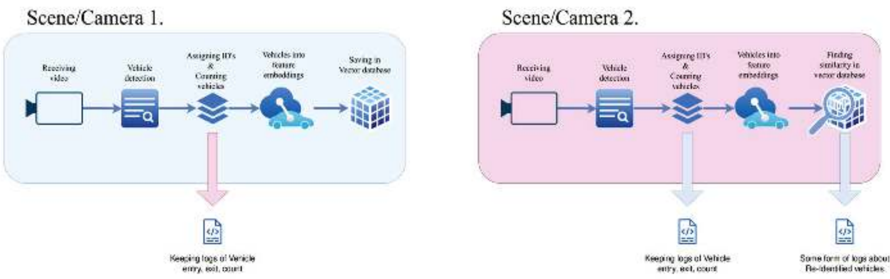

Scene/Camera 1.
Receiving video
Vehicle detection
Assigning ID's & Counting vehicles
Vehicles into feature embeddings
Saving in Vector database
Keeping logs of Vehicle entry, exit, count
Scene/Camera 2.
Receiving video
Vehicle detection
Assigning ID's & Counting vehicles
Vehicles into feature embeddings
Finding similarity in vector database
Keeping logs of Vehicle entry, exit, count
Some form of logs about Re-identified vehicles

Figure 11.3 The proposed structure of the re-identification pipeline. ⏎

During development, we split the pipeline into two parts – Vehicle counting and tracking and Vehicle Re-Identification.

11.3.1 Vehicle detection, tracking and counting

First step in the larger pipeline is object detection. We receive the videos from a network camera, detect the vehicles, get their bounding boxes and start tracking them. We use the YOLO v8 model for detection and the *ByteTrack* tracking package [23] to assign IDs to vehicles and track them through consecutive frames.

Further we establish counting criteria for incoming and outgoing cars (for example entries and exits in an intersection). This can be done with the built-in functions of *ByteTrack*, for example, *drawLine* - counting when a car drives past a drawn line as seen in Figure 11.4. The count, entry and exit times of cars should be continuously logged.

Before testing re-identification, verifying the accuracy of the vehicle counting step is essential. However, as this falls outside the paper's scope, we have intentionally excluded these sections.

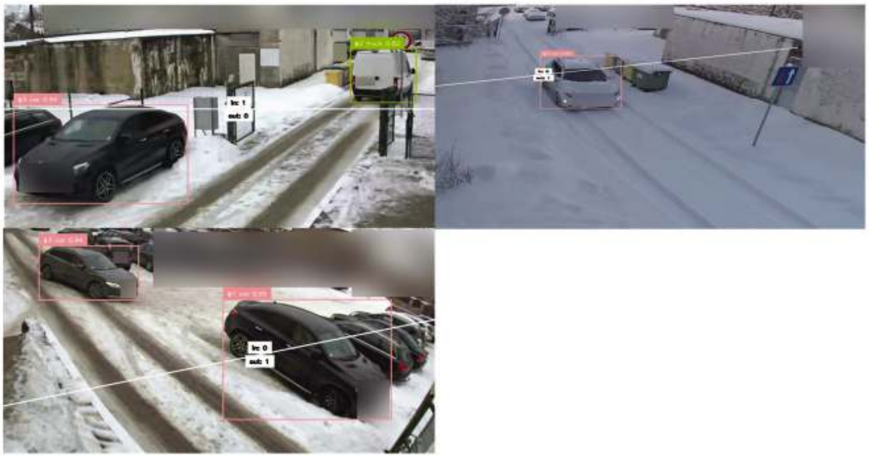


Figure 11.4 The implementation of the counting lines and ByteTrack in cameras (from the left, upper row) 1.,3. and (lower row) 2. ⏎

11.3.2 Vehicle feature extraction and storage

Consequently vehicles should be cropped out of the frame. Then the re-identification model will turn the object features into n-dimensional vectors that consist of natural, real or complex numbers, where one number represents a feature or a part of a feature [24].

We require a Vector database, so vectors can be stored and queried efficiently [25]. The database must contain cosine similarity search complemented by metadata filters. The cosine similarity function is widely used and requires an input of at least two unit-length normalized vector inputs to output a vector distance [24]. We aim to store data in the form of

$$\text{Key:Value} = \text{ObjectId:ObjectFeaturesVector}$$

while more fields should be easy to add.

For this we have chosen LanceDB – an opensource database for vector search, built for efficiency in handling vector data and integration with Python [26]. It is flexible in saving and querying data.

We also introduce the following points of action:

11.3.2.1 Datasets

We create our own "real-world" dataset for testing from footage we have gathered from our network cameras. In our custom dataset we aim to collect images of as many vehicles as the limited service-road traffic flow allows us to. We also aim to have a similar number of images per vehicle (i.e. 4-6 images, not more or less). These shots should be evenly distributed between far, medium and close distance and low, medium and high-resolution images respectively.

We will also use the three widely used and public datasets that we've discussed in 2.4, to see which fits best for our real-world data and then combine the best performing standalone dataset with the Vehicle X synthetic data.

11.3.2.2 Training hyper-parameters

The training parameters and their default values are found in [27], including Backbone, Learning rate, Warm epochs, Batch size, and Erasing probability. To further explore the model's performance, we conducted experiments by varying additional hyperparameters, specifically:

- Colour jitter: enabled or disabled;
- Size of the last linear layer: 256, 512, or 1024;
- Cosine learning rate: enabled or disabled;
- Stride: 1, 2, or 3.

These variations were designed to evaluate the potential impact of these hyperparameters on training outcomes.

11.3.3 Edge device considerations

While the pipeline has been tested on a desktop computer with an 8GB GPU, this setup provides a rough estimate of the computational load and performance we might expect on edge devices like the NVIDIA Jetson series [28]. Current work suggests that despite differences in power consumption and architecture, the constraints with desktop testing can offer insights for edge deployment that we wish to implement in the future.

11.4 Experimental settings

11.4.1 Receiving video from a Network camera

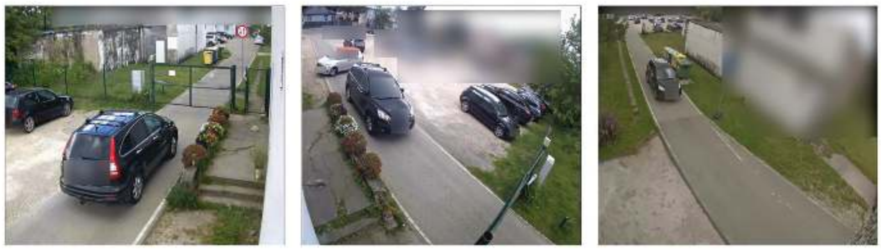


Figure 11.5 The same car visible in our network cameras (from the left) 1., 2. and 3., respectively. ⏎

To record "real-world" footage for the dataset we've envisioned in section 3.2.1, we will use 3 AXIS P1427-LE Network cameras [29] that record footage from the same service road and a parking lot. We are receiving the video in 1280×960 resolution with $\sim$ 2 fps. The view from the cameras illustrated in Figure 11.5. are summarized in Table 11.1.

When a vehicle is at the gates, camera 1 and 2 will see the car from the opposite sides (front and rear). Camera 3 is located deeper into the territory and generally sees the path of a vehicle driving down the service road with the gate in a far distance.

Table 11.1 The 3 network cameras used ⏎

#	Altitude (Above ground)	Optimal Focal zone (Position, distance from cam.)	Direction	Sides of car seen
1.	∼ 3m	centre of the frame, 4m	$\rightarrow$	Front, Back, Sides and roof (for lower cars)
2.	∼ 3m	further up the road from centre, 5-6 m	$\leftarrow$	More from the front and the back, but skewed sides and roof are visible
3.	6-8 m	Wider area around centre, 6-8 m	$\leftarrow$	front/back and roof of the car visible well, sides in poor quality

11.4.2 Vehicle re-identification

11.4.2.1 Testing and data annotation

We have chosen the following datasets to conduct our experiments on.

- **Benchmark datasets.** By using the three benchmark datasets we discussed in section 2.4, we can access reliable test data, standardize our test metrics and evaluate the re-identification model itself.
- **CityFlow test track video.** We test the whole re-identification part of the pipeline and simulate an intersection scenario where we are re-identifying vehicles with CityFlow test tracks. We will be using the scenario Nr. 1 (intersection S01) in this dataset, to re-identify vehicles from camera 1 to camera 4 [21]. There are around 2000 frames in each video and 91 unique vehicles seen. Both cameras point to the same intersection but from vastly different locations.
- **Custom test data.** We have recorded footage from 3 of our cameras. All of them cover overlapping sections of a service road inside a closed territory. The vehicles were cropped from these videos and saved into 3 folders, each for its own camera. This dataset contains 70-100 images from each camera, with ∼ 25 unique vehicle identities. We will use this data to, first and foremost, test the generalisation of our trained models to the actual data that we will use this pipeline on.

11.4.2.2 Saving the feature extractions

We will experiment with four methods (See their comparison in Table 11.2) for dealing with cropping vehicles from the frame and saving them into the database:

1. **Basic Frame-by-Frame Saving**: In this method, a feature extraction of a vehicle is saved in every frame a vehicle is detected. These vectors are saved separately under the same vehicle ID in the database. Hence, there are multiple feature embeddings for the same vehicle.
2. **Vector Summing**: Instead of storing every feature vector separately, we maintain a single vector per vehicle. Each new embedding for a vehicle is summed with the existing vector, and the result is divided by the total number of updates, averaging the embeddings over time. This process keeps track of how many times the vector has been updated by adding an additional field in the database.
3. **Zone-Based Saving**: The frame is divided into zones using a grid, and a vehicle's feature embedding is saved once per zone it passes through. The zones can be seen illustrated blue in Figure 11.6. Typically, this results in 4-6 saved vectors per vehicle, depending on its trajectory as opposed to many more vectors when saving in each frame. Each instance of feature extraction is stored as a separate vector under the same vehicle ID.
4. **Zone-Based with Vector Summing**: Similar to the previous method, but here we apply vector summing. The vehicle's vector is summed for every frame in which a vehicle has changed zones.

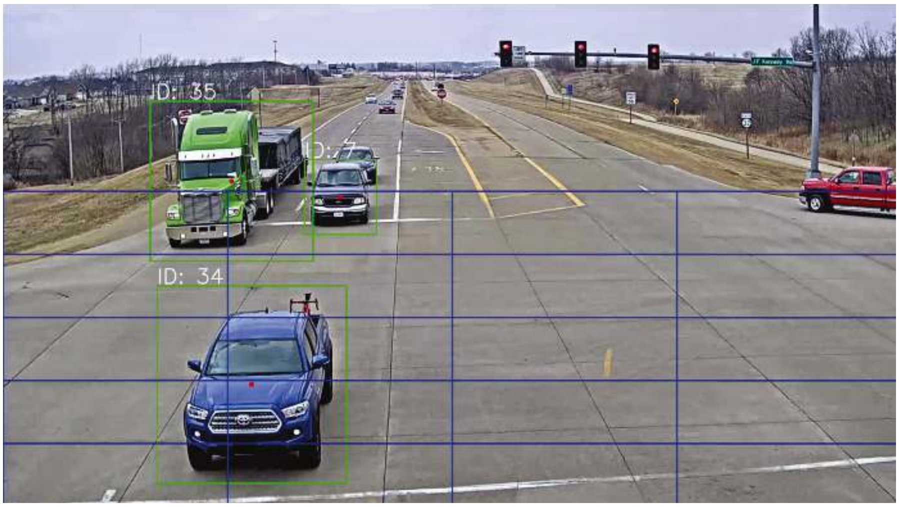


Figure 11.6 The implementation of the saving zones (drawn as blue rectangles) as seen on the CityFlow test track video. ⏎

Table 11.2 Vehicle Cropping and Saving Strategies ⏎

Method	Number of saved vectors	Saving strategy	Embedding management
Basic frame-by-frame saving	Multiple (all frames)	Per Frame	New vector for each detection
Vector Summing	1	Per Vehicle	Sum and average of all vectors
Zone-Based Saving	4-6 (based on zones)	Per Zone	New vector for each detection in new zone
Zone-Based with Vector Summing	1	Per Vehicle	Sum and average for each detection in new zone

11.5 Results

This section discusses the results of the experiments.

11.5.1 Performance metrics

- **Rank-1, Rank-X Accuracy**: Measures the proportion of examples for which the predicted label matches the single target label (Rank-1) or any of the top X predictions match the label (Rank-X) [30].
- **mAP (Mean Average Precision)**: Average precision (AP) is the average of accuracy values at all rankings where relevant objects are retrieved, and mAP - the average of all APs provided [31].
- **Micro Precision and Micro Recall**: Calculated by aggregating true positives, false positives, and false negatives across every query (instance) and doesn't consider possible class over/under representation [32].
- **Macro Precision and Macro Recall**: Precision and recall calculated separately for each class and then averaged, giving equal weight to all classes regardless of their size [32].
- **Validation Loss**: A measure of how well a model is learning during validation step of training. For training we use Cross Entropy Loss and Triplet Margin Loss together (summed).

11.5.2 Dataset generalization

We aim to find the benchmark dataset, that makes the baseline model generalize best on our own recorded data. For this we will use the Rank-1 accuracy metric as described in section 5.1, that has been averaged from 3 tests on our custom dataset.

Table 11.3 Testing on our custom dataset

Training dataset	Veri	Vehicle-ID	CityFlow
Averaged Rank-1 accuracy, %	**65.81**	44.85	65.44

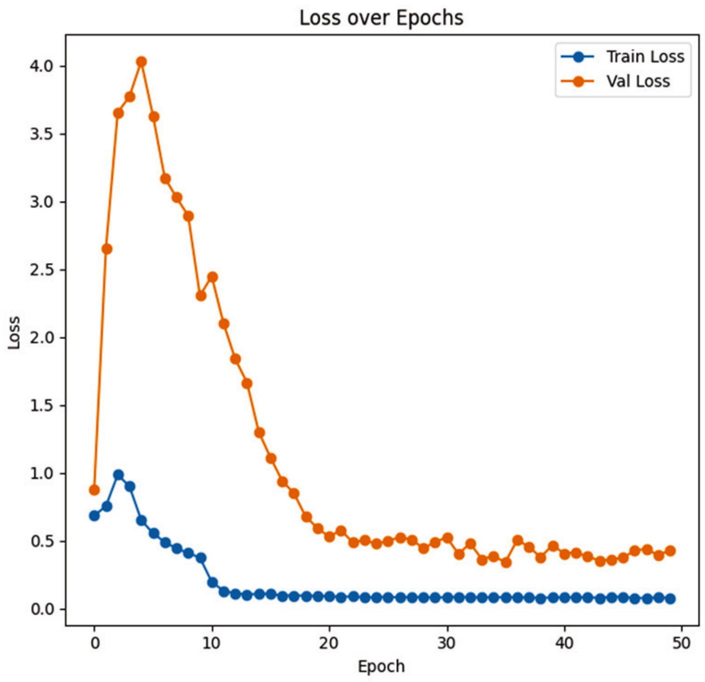


Figure 11.7 Model loss values during training on the VeRi dataset. We find values plateauing after 20 epochs. ⏎

We use the assumed default training parameters as seen in [27].

Table 11.1 indicates that training the baseline model on the Veri dataset provides the best result (we compare values at epoch 19. to reduce training times as depicted in Figure 11.7). Even if the difference between VeRi and Cityflow datasets is insignificant, visually the vehicles seen in the Veri dataset resembles our collected data more.

11.5.3 Hyper-parameters

We have trained the model on all combinations of the hyperparameters that we have outlined in section 3.2.2 over 15 epochs and recorded their loss

Table 11.4 Validation results during training

Nr.	Loss	Stride	Lin. Num.	Color jitter	Cosine learning rate	Epoch
1	0.0082	**1**	**256**	**1**	0	15.
2	0.0083	**1**	**256**	**1**	1	15.
3	0.0084	2	**256**	0	0	15.
4	0.0084	3	**256**	**1**	1	15.
5	0.0086	3	**256**	0	0	15.
6	0.009	**1**	**256**	0	0	13.
7	0.0092	**1**	**256**	0	0	15.
8	0.0092	**1**	**256**	**1**	1	13.
9	0.0092	**1**	512	**1**	0	11.
10	0.0093	**1**	**256**	**1**	0	11.

values. Training the model for 36 times in total, consisting of 540 epochs. The rest of the training parameters have been left with the default values. The 10 entries with the lowest validation loss value we showcase in Table 11.4.

The parameters that appear to have the greatest impact are Stride, Linear Layer Size, and Colour Jitter. Colour Jitter was enabled in 6 out of the 10 best results, Stride was set to 1 in 7 of the top results, and the Linear Layer Size was set to 256 in 9 out of the top 10 results. This highlights the significance of these parameters in achieving best performance.We will further train the model with a stride of 1, number of linear layers of 256 and color jitter enabled.

11.5.4 Performance on the VeRi-776 benchmark

We use the VeRi-776 dataset as a benchmark. We compare the results of our improvements to the unmodified baseline model.

In our experiments, we initially trained the baseline model using the VeRi dataset. As shown in Table 11.5 we used the default training hyper-parameters and evaluated the model's performance over 20 and 50 epochs. The model trained on 50 epochs outperforms the one with 20.

Training the model with the updated parameters of stride, linear number and color jitter (see Section 5.3) demonstrates further improvement on the benchmark.

Table 11.5 VeRi Trained Model comparisons on the VeRi-776 benchmark ⏎

Model	**Rank-1**	**Rank-5**	**Rank-10**	**mAP**
Baseline trained on Veri, 19$^{\text{th}}$ epoch	93.80	97.61	98.74	68.96
Baseline trained on Veri, 49$^{\text{th}}$ epoch	94.04	97.49	98.80	69.4
Model with updated parameters trained on Veri, 49$^{\text{th}}$ epoch	95.47	97.62	98.51	71.70
Model with updated parameters trained on Veri and VehicleX, 49$^{\text{th}}$ epoch	**95.83**	**98.09**	**98.87**	**73.64**

Lastly, we conducted additional training by incorporating the VehicleX synthetic data into the training process.

All together our efforts have increased mAP by 4.24% and Rank-1 by 1.79% from "Model trained on Veri, 49th epoch" to "Model trained on Veri and VehicleX, 49th epoch with updated parameters".

11.5.5 Re-identification testing on test data from our cameras

In this section we test the model performance on the test data gathered from our network cameras.

11.5.5.1 Camera to camera re-identification

Table 11.4 contains the Micro and Macro precision values and table 5.5 contains the Micro and Macro recall values (see the description of these metrics in section 5.1), when re-identifying from a query camera *x* (rows) to gallery camera *y* (columns).

$$\text{Micro Precision} = \frac{\sum \text{True Positives}}{\sum \text{True Positives} + \sum \text{False Positives}} \tag{11.1}$$

$$\text{Macro Precision} = \frac{1}{N} \mathop{}\limits_{i=1}^{N} \frac{\text{True}^{N}\,\text{Positives}_{i}}{\text{True Positives}_{i} + \text{False Positives}_{i}^{\text{Pa}}}. \tag{11.2}$$

$$\text{Micro Recall} = \frac{\sum \text{True Positives}}{\sum \text{True Positives} + \sum \text{False Negatives}} \tag{11.3}$$

$$\text{Macro Recall} = \frac{1}{N} \mathop{}\limits_{i=1}^{N} \frac{\text{True}^{\text{Positives}_{i}}}{\text{True Positives}_{i} + \text{False Negatives}_{i}^{\text{Nat}}} \tag{11.4}$$

Table 11.6 Precision when re-identifying from a query camera to a gallery camera

Micro pr.	Macro pr.	Camera 1		Camera 2		Camera 3	
Camera 1				64.37	65.09	78.08	77.29
Camera 2		66.13	57.58			100	100
Camera 3		85.71	78.68	73.58	76.98		

Table 11.7 Recall when re-identifying from a query camera to a gallery camera ⏎

Micro r.	Macro r.	Camera 1		Camera 2		Camera 3	
Camera 1				64.37	66.00	78.08	76.28
Camera 2		66.13	46.22			100	100
Camera 3		85.71	75.75	73.58	76.67		

Where: N is the total number of classes,
True positives$_i$ are the number of correct predictions for class i,
False negatives$_i$ are the incorrect predictions for class i.

Judging by the precision and recall values we can establish the following conclusions:

- The model generalizes reasonably well to our custom testing data
- The percentages and differences in micro/macro values suggest the model may be overfitting to certain well-represented vehicle identities in camera 2 and hints at difficulty in recognizing rare vehicles elsewhere.
- Overall, it seems the model can generalise camera 3. feature extractions the best against all scenes but has trouble generalising camera 1. and 2. feature extractions against each other.

Table 11.8 Macro, micro precision and recall when re-identifying from a query camera to a gallery of two cameras ⏎

Cameras	**Micro Precision, %**	**Macro Precision, %**	**Micro Recall, %**	**Macro Recall, %**
1 → 2,3	73.39	80.63	73.39	74.96
2 → 1,3	75.81	67.09	75.81	59.38
3 → 2,1	85.71	89.21	85.71	88.17

11.5.5.2 Sets of cameras

We can even out the results of the different cameras if a car has passed through at least two cameras, which has allowed us to gather more data in the database of any given car. Let's use a test, where we query vehicles of one camera from a gallery of two cameras. The cameras column specifies the camera x (query), who's images are queried in the camera y,z (gallery) images.

Table 11.8 shows that re-identification for vehicles that have passed through at least two cameras is more reliable. Camera 3 shows the highest performance with both micro and macro values exceeding 85%, indicating strong generalization across vehicle identities. In contrast, re-identifying from camera 2 to cameras 1 and 3 results in the lowest macro recall (59.38%), suggesting difficulty in retrieving true matches for less represented vehicle identities. Overall, the model performs well, but the noticeable drop in macro values for camera 2 queries points to possible need to ease class imbalance in the future.

In practice the results may indicate that cameras that have a slightly zoomed out or aerial view of the roads or intersections (Camera 3) can produce feature extractions that generalize better than cameras that see the vehicles up close. It can also be observed that camera 2 had a 100% re-identification precision when querying against camera 3, which could be explained by the fact that both cameras 2 and 3 had similar angles with only height varying. What is harder to conclude is why the same was not true for camera 3 querying against camera 2. Similarly, it can be observed that cameras 1 and 2 have opposite viewpoints, so that even if the car went back and forth through both cameras, each camera would only have a flipped image of what the other camera has. This explains the poorer performance between camera 1 and 2.

11.5.6 Testing the whole re-identification part of the pipeline

We test our model on the CityFlow video tracks - re-identifying vehicles from (intersection) S01 traffic camera 4 to camera 1 To make the process

Table 11.9 Testing methods of the re-identification pipeline on CityFlow video tracks

Test scenario	Basic Frame-by-Frame Saving:	Vector Summing	Zone-Based Saving	**Zone-Based with Vector Summing:**
Rank-1 accuracy in %	65.52	62.78	72.90	**74.13**
Test duration in seconds	1107.80	1085.30	330.12	**328.90**

as close to real world as possible we will use the Rank-1 accuracy to measure the accuracy of the pipeline, since in a scenario like this we simply care for whether the vehicle has been re-identified correctly or not. As mentioned in section 4.2.2 , we test the 4 approaches of capturing the feature embeddings and saving them into a database.

Overall, the tests on CityFlow video tracks show that capturing vehicles in distinct zones improves both accuracy and efficiency, as clearly illustrated in Table 11.7.

11.6 Future research

Although this paper proposes a complete pipeline for object re-identification on edge devices, multiple areas remain for future research. First, optimizing the pipeline for specific edge devices like the Nvidia Jetson by profiling performance and applying techniques such as model pruning, quantization, or TensorRT for efficient inference. Additionally, fine-tuning models trained on public datasets with our in-house data could reduce distribution shifts and improve real-world performance. Finally, a more in-depth analysis of feature vectors is needed to better understand which components are relevant for re-identification accuracy and which are redundant.

11.7 Conclusion

In this paper, we presented a multi-step pipeline for object re-identification, focusing on real-time applications using edge devices. The pipeline handles object detection, feature extraction, and matching through a vector database, demonstrating reliable vehicle re-identification across various scenes. Performance dynamics were evaluated by comparing different datasets, models, pipeline processes. The authors observed the position and angle of cameras significantly influencing the accuracy of vehicle re-identification, with higher or wider vantage points producing better feature extractions for generalization across scenes.

The findings demonstrate that while we successfully constructed an object re-identification pipeline by combining state-of-the-art methods, its effectiveness and constraints are highly dependent on the specific application scenario. Re-identification performance is influenced by factors such as the suitability of training data for real-world generalization, the model training approach, and the careful tuning of hyperparameters. For vehicle re-identification, it is crucial to consider where and when the vehicles are detected, the characteristics of the road or intersection, and the specific features of the camera recording the footage. From this, we can conclude that achieving high performance in vehicle re-identification requires not only advanced methodologies but also scenario-specific adaptations in data preparation, model optimization, and detection configuration. We also conclude that obtaining multiple embeddings of the same object in different poses and locations help re-identifying it later.

Acknowledgements

This work was supported by Chips Joint Undertaking EdgeAI project. The project EdgeAI "Edge AI Technologies for Optimised Performance Embedded Processing" is supported by the Chips Joint Undertaking and its members including top-up funding by Austria, Belgium, France, Greece, Italy, Latvia, Netherlands, and Norway under grant agreement No 101097300.

References

[1] S. J. Prince, *Computer vision: models, learning, and inference.* Cambridge University Press, 2012.

[2] D. Lu and Q. Weng, "A survey of image classification methods and techniques for improving classification performance," *International Journal of Remote Sensing*, vol. 28, no. 5, pp. 823–870, Mar. 2007, doi:https://doi.org/10.1080/01431160600746456.

[3] J. Du, "Understanding of Object Detection Based on CNN Family and YOLO," *Journal of Physics: Conference Series*, vol. 1004, p. 012029, Apr. 2018, doi:https://doi.org/10.1088/1742-6596/1004/1/012029.

[4] X. Li and Z. Zhou, "Object Re-Identification Based on Deep Learning," *IntechOpen eBooks*, Jul. 2019, doi:https://doi.org/10.5772/intechopen.86564.

[5] R. Kuma, E. Weill, P. Sriram, and F. Aghdasi, "Vehicle re-identification: an efficient baseline using triplet embedding," presented at the 2019 International Joint Conference on Neural Networks (IJCNN), IEEE, Jul. 2019, pp. 1–9.

[6] J. Barthélemy, N. Verstaevel, H. Forehead, and P. Perez, "Edge-Computing Video Analytics for Real-Time Traffic Monitoring in a Smart City," *Sensors*, vol. 19, no. 9, p. 2048, May 2019, doi: https://doi.org/10.3390/s19092048.

[7] C. Wang, Y. Yang, M. Qi, and H. Ma, "Efficient Cloud-edge Collaborative Inference for Object Re-identification," *arXiv preprint*, vol. arXiv:2401.02041, 2024.

[8] K. Cao, Y. Liu, G. Meng, and Q. Sun, "An Overview on Edge Computing Research," *IEEE Access*, vol. 8, no. 1, pp. 85714–85728, 2020, doi: https://doi.org/10.1109/access.2020.2991734.

[9] "YOLOv8 Ultralytics: State-of-the-Art YOLO Models," *learnopencv.com*, Jan. 10, 2023. https://learnopencv.com/ultralytics-yolov8/#YOLOv8-vs-YOLOv5 (accessed Sep. 23, 2024).

[10] A. O. Salau and S. Jain, "Feature Extraction: A Survey of the Types, Techniques, Applications," *IEEE Xplore*, Mar. 01, 2019. https://ieeexplore.ieee.org/abstract/document/8938371 (accessed Sep. 23, 2024).

[11] Z. Zheng, X. Yang, Z. Yu, L. Zheng, Y. Yang, and J. Kautz, "Joint Discriminative and Generative Learning for Person Re-identification." Accessed: Jan. 10, 2025. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2019/papers/Zheng_Joint_Discriminative_and_Generative_Learning_for_Person_Re-Identification_CVPR_2019_paper.pdf

[12] E. Almeida, B. Silva, and J. Batista, "Strength in Diversity: Multi-Branch Representation Learning for Vehicle Re-Identification," *arXiv (Cornell University)*, Jan. 2023, doi: https://doi.org/10.48550/arxiv.2310.01129.

[13] S. V. Huynh, N. H. Nguyen, N. T. Nguyen, Vinh Tq. Nguyen, C. Huynh, and C. Nguyen, "A Strong Baseline for Vehicle Re-Identification," *arXiv (Cornell University)*, Jun. 2021, doi: https://doi.org/10.1109/cvprw53098.2021.00468.

[14] Z. Zheng, T. Ruan, Y. Wei, and Y. Yang, "VehicleNet: Learning Robust Feature Representation for Vehicle Re-identification," *Computer Vision and Pattern Recognition*, pp. 1–4, Jan. 2019.

[15] Z. Zheng, L. Zheng, and Y. Yang, “A Discriminatively Learned CNN Embedding for Person Reidentification,” *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 14, no. 1, pp. 1–20, Dec. 2017, doi:https://doi.org/10.1145/3159171.

[16] Z. Zheng, T. Ruan, Y. Wei, Y. Yang, and T. Mei, “VehicleNet: Learning Robust Visual Representation for Vehicle Re-identification,” Apr. 2020.

[17] “Papers with Code - Vehicle Re-Identification,” *Paperswithcode.com*, 2022. https://paperswithcode.com/task/vehicle-re-identification#benchmarks (accessed Sept. 23, 2024).

[18] layumi, “GitHub - layumi/Person_reID_baseline_pytorch: Pytorch ReID: A tiny, friendly, strong pytorch implement of person re-id / vehicle re-id baseline.,” *GitHub*, Jul. 25, 2022. https://github.com/layumi/Person_reID_baseline_pytorch/ (accessed Sept. 24, 2024).

[19] H. Liu, Y. Tian, Y. Wang, L. Pang, and T. Huang, “Deep Relative Distance Learning: Tell the Difference between Similar Vehicles,” *IEEE Xplore*, Jun. 01, 2016. https://ieeexplore.ieee.org/document/7780607

[20] X. Liu, W. Liu, T. Mei, and H. Ma, “A Deep Learning-Based Approach to Progressive Vehicle Re-identification for Urban Surveillance,” *Computer Vision – ECCV 2016*, pp. 869–884, 2016, doi:https://doi.org/10.1007/978-3-319-46475-6_53.

[21] Z. Tang *et al.*, “CityFlow: A City-Scale Benchmark for Multi-Target Multi-Camera Vehicle Tracking and Re-Identification,” *arXiv (Cornell University)*, Jun. 2019, doi:https://doi.org/10.1109/cvpr.2019.00900.

[22] Y. Yao, L. Zheng, X. Yang, Milind Naphade, and T. Gedeon, “Simulating Content Consistent Vehicle Datasets with Attribute Descent,” *Lecture notes in computer science*, pp. 775–791, Jan. 2020, doi: https://doi.org/10.1007/978-3-030-58539-6_46.

[23] Y. Zhang *et al.*, “ByteTrack: Multi-object Tracking by Associating Every Detection Box,” pp. 1–21, Oct. 2021, doi:https://doi.org/10.1007/978-3-031-20047-2_1.

[24] R. J. Bayardo, Y. Ma, and Ramakrishnan Srikant, “Scaling up all pairs similarity search,” *The Web Conference*, May 2007, doi:https://doi.org/10.1145/1242572.1242591.

[25] T. Taipalus, “Vector database management systems: Fundamental concepts, use-cases, and current challenges,” *Cognitive Systems Research*, vol. 85, p. 101216, Jun. 2024, doi:https://doi.org/10.1016/j.cogsys.2024.101216.

[26] lancedb, “GitHub - lancedb/lancedb: Developer-friendly, serverless vector database for AI applications. Easily add long-term memory to your

LLM apps!," *GitHub*, Sep. 24, 2024. https://github.com/lancedb/lancedb (accessed Oct. 03, 2024).

[27] regob, "GitHub - regob/vehicle_reid: Vehicle Re-identification," GitHub, 2022. https://github.com/regob/vehicle_reid/tree/master(accessed Oct. 14, 2024).

[28] S. Valladares, M. Toscano, R. Tufiño, P. Morillo, and D. Vallejo-Huanga, "Performance Evaluation of the Nvidia Jetson Nano Through a Real-Time Machine Learning Application," *Advances in Intelligent Systems and Computing*, pp. 343–349, 2021, doi: https://doi.org/10.1007/978-3-030-68017-6_51.

[29] "AXIS P1427-LE Network Camera - Product support | Axis Communications," *Axis.com*, 2023. https://www.axis.com/products/axis-p1427-le/support (accessed: September 26, 2024).

[30] V. Shankar, R. Roelofs, H. Mania, A. Fang, B. Recht, and L. Schmidt, "Evaluating Machine Accuracy on ImageNet," *PMLR*, pp. 8634–8644, Nov. 2020, Accessed: Sept. 26, 2024. [Online]. Available: http://proceedings.mlr.press/v119/shankar20c.html?ref=https://githubhelp.com

[31] Mian Muhammad Talha, Hikmat Ullah Khan, S. Iqbal, M. Alghobiri, T. Iqbal, and M. Fayyaz, "Deep learning in news recommender systems: A comprehensive survey, challenges and future trends," *Neurocomputing*, vol. 562, pp. 126881–126881, Dec. 2023, doi: https://doi.org/10.1016/j.neucom.2023.126881.

[32] Jean-Charles Lamirel, Maha Ghribi, and Pascal Cuxac, "Unsupervised recall and precision measures: a step towards new efficient clustering quality indexes," Aug. 2010.

12

A TinyMLOps Framework for Real-world Applications

Mattia Antonini, Massimo Vecchio, and Fabio Antonelli

Fondazione Bruno Kessler, Italy

Abstract

As devices become smarter, embedding intelligence in microcontrollers and constrained environments is critical. Optimising machine learning models for these tiny devices requires balancing software efficiency, such as accuracy, with hardware constraints like memory and power. We introduce a TinyMLOps-based framework for optimising models across the cloud-to-device continuum. In our approach, cloud resources handle heavy tasks like data labelling and model training, while microcontrollers gather real-time metrics on efficiency and hardware utilisation. Then, some repositories manage models and metadata identified during the optimisation phase, including performance metrics collected directly from the target devices, thus ensuring an accurate exploration of the model space in real-world conditions. Using tools such as MicroPython and MLFlow, our framework enables seamless AI deployment on resource-constrained devices, providing a scalable solution for the future of edge AI.

Keywords: MLOps, edge AI, edge computing, model deployment, AI workflow, real-time metrics.

12.1 Introduction

Artificial intelligence (AI) is becoming an invaluable companion in everyday life, present in wearables, smartphones, cars, and homes. We use AI to

DOI: 10.1201/9788743808862-12

write, compose music, and edit pictures, making access almost effortless. However, this can obscure the real challenges of deploying AI models, especially on devices at the edge of the network [1]. These devices include single-board computers (e.g., Raspberry Pi, Coral Dev Kit, NVIDIA Jetson boards), which support full-fledged operating systems (OS) like Linux but are costly and energy-intensive. In contrast, microcontrollers (MCUs) offer lower computational power but are significantly cheaper and less demanding regarding energy and hardware resources. MCUs are ideal for specialised, real-time tasks in IoT environments but require adapted methodologies for their management [2]. The absence of a full-fledged OS and the constrained computational environment demand adjustments in orchestrating AI workflows on such devices.

In this context, the TinyMLOps methodology presented in [3] can be fundamental in designing, deploying, and monitoring AI capabilities on constrained devices. From an operational standpoint, US-based companies like Roboflow [4], Edge Impulse [5], and Neuton.ai [6] provide platforms for designing, optimising, deploying, and executing AI pipelines at the network edge. While Roboflow focuses on computer vision problems, primarily supporting single-board computers, Edge Impulse specialises in creating and deploying machine learning models on resource-constrained devices like MCUs, making it ideal for IoT applications. However, it requires hardcoding models into firmware, updated via Over-the-Air (OTA) updates, which reduces flexibility in real-time scenarios. Similarly, the Neuton TinyML platform, provided by Neuton.ai, leverages a no-code approach to generate ultra-compact neural networks, optimising resource use for constrained devices. Its patented framework incrementally builds neural networks neuron by neuron, resulting in models significantly smaller than those from other frameworks. It enables deployment on devices with as little as 8-bit capacity, making it highly suitable for diverse IoT applications.

This paper presents a TinyMLOps framework architecture designed to streamline AI workflows on MCUs, enabling seamless model deployment without embedding the model in the firmware. We describe the framework's components, their role in supporting the methodology, and the various actors involved, from data labelling to model design, optimisation, firmware engineering, and operations. Furthermore, we provide a set of candidate state-of-the-art technologies that can be adopted to implement the framework and demonstrate its real-world feasibility.

The rest of the paper is structured as follows. Section 12.2 gives an overview of the TinyMLOps methodology, then Section 12.3 presents the framework architecture by describing all the components. Section 12.4 provides a technological landscape for the framework. Finally, Section 12.5 concludes the paper.

12.2 TinyMLOps methodology

The TinyMLOps methodology [3] evolves the MLOps methodology [7] to bring AI workflows and models to the edge and far edge of the network.

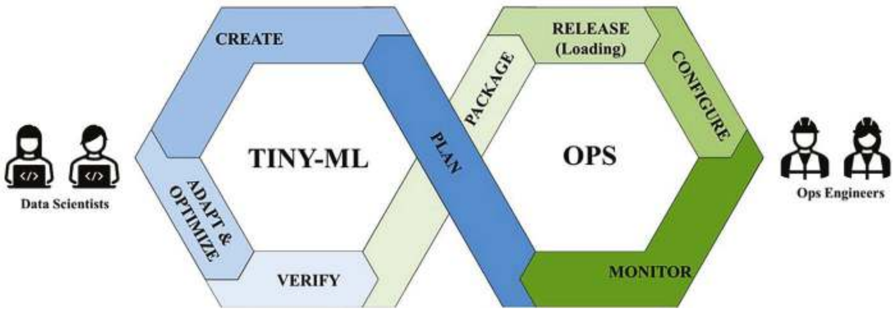


Figure 12.1 The TinyMLOps Loop [3]. ⏎

TinyMLOps particularly tackles the adaptation, deployment, and monitoring of models even on low-end MCUs (e.g., Raspberry Pi Pico, ESP32, Arduino, and STM32 families).

The TinyMLOps approach is illustrated as a simple loop, as depicted in Figure 12.1.

In more detail, the TinyMLOps loop starts with the **TinyML circle** (on the left side, highlighted in blue, Figure 12.1). It proceeds through a series of iterative steps involving collaboration between data scientists and operations engineers (on the right side highlighted in green in the Figure 12.1) to optimise and deploy AI models on edge devices. The steps are briefly introduced as follows.

- **Plan**: data exploration and feature engineering to prepare and optimise data for model development;
- **Create**: model training and feature adaptation tailored to the problem space;
- **Adapt & Optimize**: the trained model is adapted and optimised for the target computing platform, typically through techniques such as quantisation and pruning;

- **Verify**: ensures model compatibility and executability on the target platform, confirming that all operations are supported;
- **Package**: the model is compiled or converted into a deployable format (e.g., TFLite) suitable for execution on edge devices;
- **Release**: the model is deployed into the device memory, making it ready for inference;
- **Configure**: run-time parameters may be adjusted based on platform- and application-specific requirements (e.g., detection threshold);
- **Monitor**: ongoing real-time surveillance of the model's performance and health, ensuring its reliability post-deployment. If the model behaves unexpectedly, it triggers a new iteration of the entire loop.

This closed-loop process facilitates seamless adaptation, deployment, and monitoring of TinyML models, ensuring efficient execution on resource-constrained edge devices.

12.3 A TinyMLOps framework architecture

As discussed above, the TinyMLOps methodology encompasses multiple abstract phases to deliver an efficient pipeline for deploying and maintaining machine learning models on edge devices. To implement these phases in real-world applications, we need to translate them into concrete components and pipelines within a software framework architecture. Figure 12.2 illustrates this architecture, which is divided into two sections corresponding to the two circles of the TinyMLOps loop (Figure 12.1). The left side (in blue) represents the components required for handling the TinyML-specific tasks, while the right side (in green) focuses on the components involved in the methodology's operations phase.

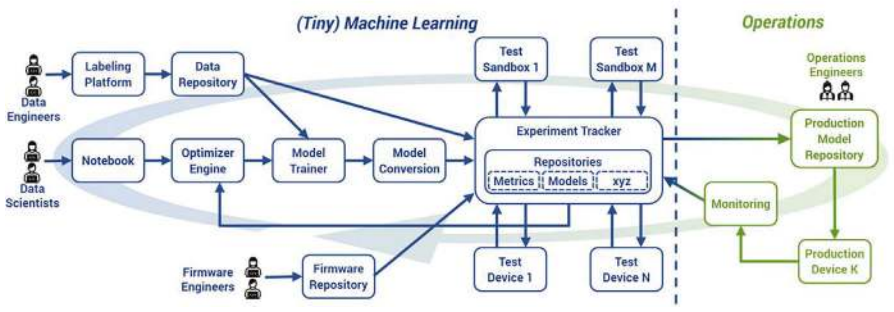


Figure 12.2 TinyMLOps Framework Architecture. ⏎

Starting from the TinyML part, we can distinguish three main pipelines: the firmware engineering pipeline, the data engineering pipeline, and the data science pipeline, which are introduced as follows:

- **firmware engineering pipeline**: managed by firmware engineers, produces the various firmware versions for the edge devices supported by our framework. These firmware binaries are organised and stored in the Firmware Repository, ready for deployment. For example, one firmware might be based on MicroPython[1], compiled for ESP32 and equipped with libraries tailored to the target application.
- **data engineering pipeline**: managed by data engineers, provides a UI through the Labelling Platform for labelling data and storing the information (e.g., labels, metadata) in the Data Repository. This repository plays a crucial role, as it is accessed during model training and validation.
- **data science pipeline**: managed by data scientists, which offers the possibility to design, develop, and test models, as detailed below.

Models are initially created using a Notebook environment (e.g., Jupyter Notebooks, VSCode) where scientists can define the computing pipeline, the model architecture, and the relevant hyperparameters ranges based on the target metrics they aim to optimise (e.g., model size vs. accuracy). Additionally, they can specify the desired hardware platform for deployment, ensuring the model aligns with the constraints and capabilities of the chosen edge device.

At this stage, the Optimization Engine generates and manages various parameter combinations to explore and optimise the search space. In our case, this process is handled through a metaheuristic approach, ensuring effective exploration and solution space optimisation. This includes not only the hyperparameters of the models but also factors such as hardware configurations, allowing for an optimal balance between performance metrics (e.g., model size, accuracy, latency) and resource constraints.

The combination of parameters and the model definition (e.g., code) is passed to the Model Trainer component, which trains the model using data fetched from the Data Repository. However, the trained model may not yet be fully compatible with the target hardware platform. To address this, the model is further processed by the Model Conversion component, which applies various techniques such as quantisation, pruning, and other optimisations to ensure the model is adapted for efficient execution on the specific hardware platform.

[1] https://micropython.org/

At this stage, it is essential to evaluate the model's performance. To efficiently manage the artefacts (i.e., model binary files), along with associated parameters (e.g., data used to train and model hyperparameters) and metrics, we rely on the Experiment Tracker to handle the storage of data and metadata in dedicated internal repositories, accessible via APIs.

Based on the metrics we aim to measure and the target hardware platform, the model can be deployed in the appropriate computing environment: a cloud-based Test Sandbox for software metrics (e.g., accuracy) and physical Test Devices for hardware-specific metrics (e.g., latency, memory usage). This approach ensures a clear understanding of the model's performance in real-world environments, as testing is carried out on actual computing units. This provides insights into software and hardware efficiency, allowing for more accurate metrics assessments.

The collected metrics are then returned to the Optimizer Engine, where they are used to further refine the hyperparameters, model configuration, and hardware selection by triggering new iterations.

This continuous feedback loop allows the system to iteratively improve the model, ensuring optimal performance and deployment on the target edge devices. All tested models, along with their associated parameters and metrics, are stored in the Experiment Tracker. This comprehensive tracking provides valuable insights into how models can be further optimised for the specific application and platform. This is beneficial as it allows for easy access, tracking, and comparison of different model iterations, ensuring that the most effective model can be quickly identified and deployed. Additionally, this centralised storage facilitates collaboration between teams, improves reproducibility, and supports ongoing optimisation efforts by making historical data readily available for analysis.

Once the best model is identified, fully optimised, and validated, it is promoted from the Model Repository inside the Experiment Tracker to the Production Model Repository, making it ready for deployment on production devices. This transition marks the shift from development and testing to the operational phase, managed by Operations Engineers. From the Production Model Repository, the model is deployed to Production Devices, typically (far-)edge devices operating in a live environment. Operations Engineers oversee this deployment, ensuring the model performs reliably and efficiently in real-world conditions.

Throughout the Operations phase, continuous monitoring is crucial. Operations Engineers may continue to observe the model's performance, gathering insights such as real-time metrics, latency, and memory usage using the Monitoring component. If any issues arise or further optimisations are needed, the model can be updated or refined through subsequent iterations, with performance feedback potentially being routed back into the development process to trigger new rounds of optimisation and improvements via the Experiment Tracker. This ongoing cycle ensures that the model remains efficient and effective as it operates on resource-constrained edge devices, seamlessly fulfilling the principles of the TinyMLOps methodology.

12.4 Technology Overview for TinyMLOps Adoption

Adopting the TinyMLOps methodology into real-world scenarios requires integrating a wide range of cutting-edge technologies that allow us to tackle the different challenges of deploying machine learning models on resource-constrained devices in the edge and far-edge of the network. The current landscape, driven by open-source innovation, encompasses platforms and tools that ease data management and versioning, model design, development, optimisation, and monitoring in environments where computational power and storage are limited.

Starting with the **firmware engineering pipeline**, several tools are available to support the early-stage development and production deployment of machine learning models on hardware-constrained devices. On the hardware side, embedded platforms such as Arduino, ESP32, Raspberry Pi Pico, ARM, and STM32 are widely adopted. These platforms are typically paired with development environments like Mbed OS, PlatformIO, ESP-IDF, and STM32Cube, which aid in firmware development, sensor integration, and more. For software, frameworks like TensorFlow Lite for Microcontrollers [8] and pure C/C++ code (e.g., code generated by STM32Cube.AI), along with a Real-Time Operating System (e.g., FreeRTOS), simplify model deployment on embedded devices. However, since the model is embedded within the device's firmware, updating it typically requires a complete flash. On the same side, MicroPython offers an alternative solution by enabling the creation of firmware that exposes a Python interpreter running on the constrained device, effectively functioning as a Hardware Abstraction Layer (HAL)[9]. This approach decouples the machine learning model from the underlying hardware and dependency libraries, allowing for the deployment of only a Python script and associated binary files without the need for a full firmware flash. The device can initiate updates through a simple HTTP request, significantly streamlining the process and making it more flexible and efficient.

For the **data engineering pipeline**, tools like Label Studio, Universal Data Tool, or Computer Vision Annotation Tool (CVAT) are critical for labelling and annotating complex datasets, ranging from images to time-series sensor data. These labelled datasets are managed using tools like Pachyderm, which combines scalable data storage with version control, making it ideal for tracking large volumes of data throughout different iterations of the machine-learning pipeline. Object storage solutions such as MinIO or AWS S3 are often employed to store these datasets, ensuring scalability and accessibility during model development with a simple and standardised API. Alternatively, it is possible to integrate, via APIs, existing platforms like Edge Impulse or Roboflow to exploit their data labelling functionalities. Based on cost constraints, privacy, and use-case requirements, it is possible to choose one solution or the other.

Lastly, the **data science pipeline** relies massively on cloud-based tools to design, develop, evaluate, and optimise machine learning models. Kubeflow, a powerful open-source framework, is crucial in orchestrating machine learning workflows, from data preprocessing to model training. By leveraging Kubernetes for scalable and efficient workload management, Kubeflow standardises the MLOps process, streamlining the entire model development lifecycle. It natively integrates with platforms such as Jupyter Notebooks and VSCode, providing collaborative environments for interactive model development and pipeline processing. A core feature of Kubeflow is its Pipelines component, which automates end-to-end workflows by structuring them as Directed Acyclic Graphs (DAGs). Each pipeline component is encapsulated in a Docker container, ensuring modularity and enabling integration with various tools and libraries. These pipeline steps can include tasks such as data preprocessing, model training, and evaluation.

A key step often incorporated is hyperparameter optimisation, which determines the best hyperparameter configuration for a given model. Katib, a component of Kubeflow, streamlines this process by supporting both standard and advanced search techniques, ensuring optimal performance for the model and hardware platform. This approach allows for co-design between software and hardware, optimising both simultaneously to target the deployment environment. This stage also often involves conversion tools to export models in TensorFlow Lite and ONNX (Open Neural Network Exchange) formats to enable optimised evaluations for edge devices. Such models can be stored in an Experiment Tracker, like the MLFlow Tracking, along with their metadata to study and supervise the optimisation process properly.

Before the deployment in production environments, models are evaluated in Test Sandboxes using Docker-based technologies like Seldon Core, KServe, or BentoML. For tests on Test Devices, models can be deployed using OTA updates by replacing the firmware or using MLFlow Tracking APIs by downloading models into device flash memory. Performance metrics are logged in the Experiment Tracker (MLFlow Tracking), enabling a feedback loop for subsequent iterations of the model optimisation. MLFlow also offers the Model Registry, a repository for models that have reached maturity for production deployment. Using some internal labels, it is possible to tag models for the target device or deployment scenario. Then, the model can be deployed via OTA updates or via the MLFlow APIs.

Finally, tools like Prometheus and Grafana enable real-time monitoring of the production devices, providing detailed insights into hardware (latency, memory usage, and CPU load) and software performance metrics (errors, crashes, anomalies). Prometheus collects and stores time-series data from the devices, while Grafana visualises this information through customisable dashboards by also enabling webhooks to trigger new optimisation iterations automatically.

This integration allows operations teams to proactively track the health of models and devices, ensuring that any performance issues can be addressed promptly to maintain optimal efficiency and reliability in the deployed system.

12.5 Conclusions

The ubiquitous presence of AI in everyday aspects of our lives is deeply transforming how we approach the lifecycle of models. From the initial model design to the deployment on production devices, multiple steps must be completed, many of which are often taken for granted. Every phase (e.g., data collection, data labelling, model training, model optimisation) involves complex processes that need to be carefully orchestrated to ensure optimal model performance in real-world scenarios.

In these settings, the TinyMLOps methodology clearly defines the high-level steps that must be tackled. Building on this foundation, we propose a software framework architecture streamlining these processes along the cloud-to-thing continuum. Our framework enables the design, development, and deployment of machine learning models while managing hardware constraints effectively. By integrating real-world tools like MicroPython for lightweight and simple HAL and MLFlow for experiment tracking and deployment, the framework addresses key challenges in scaling AI applications to tiny devices, such as resource management and deployment complexity. Additionally, it enhances adaptability and scalability across diverse platforms, ensuring that models perform efficiently even in the most constrained environments.

Acknowledgments

This paper was partially supported by the Interconnected Nord-Est Innovation Ecosystem (iNEST) and received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR)—MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.5—D.D. 1058 23/06/2022, ECS00000043). Moreover, it was partially supported by the Horizon Europe "NEUROKIT2E" project (Grant Agreement 101112268) and by the Horizon Europe "Arrowhead fPVN" project (Grant Agreement 101111977).

References

[1] T. Meuser et al., "Revisiting Edge AI: Opportunities and Challenges," IEEE Internet Comput., vol. 28, no. 4, pp. 49–59, Jul. 2024, https://doi.org/10.1109/MIC.2024.3383758.

[2] M. Antonini, M. Pincheira, M. Vecchio, and F. Antonelli, "An Adaptable and Unsupervised TinyML Anomaly Detection System for Extreme Industrial Environments," Sensors, vol. 23, no. 4, p. 2344, Feb. 2023, https://doi.org/10.3390/s23042344.

[3] M. Antonini, M. Pincheira, M. Vecchio, and F. Antonelli, "Tiny-MLOps: a framework for orchestrating ML applications at the far edge of IoT systems," in 2022 IEEE International Conference on Evolving and Adaptive Intelligent Systems (EAIS), Larnaca, Cyprus: IEEE, May 2022, pp. 1–8. https://doi.org/10.1109/EAIS51927.2022.9787703.

[4] Dwyer, B., Nelson, J., Hansen, T., and et. al., Roboflow (Version 1.0). (2024).

[5] S. Hymel et al., "Edge Impulse: An MLOps Platform for Tiny Machine Learning," 2022, arXiv. https://doi.org/10.48550/ARXIV.2212.03332.

[6] "Neuton.AI - No-code artificial intelligence for all." Accessed: Jan. 10, 2025. [Online]. Available: https://neuton.ai/

[7] M. M. John, H. H. Olsson, and J. Bosch, "Towards MLOps: A Framework and Maturity Model," in 2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Palermo, Italy: IEEE, Sep. 2021, pp. 1–8. https://doi.org/10.1109/SEAA53835.2021.00050.

[8] R. David et al., "TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems," 2020, arXiv. https://doi.org/10.48550/ARXIV.2010.08678.

[9] M. Antonini, M. Pincheira, M. Vecchio, and F. Antonelli, "A TinyML approach to non-repudiable anomaly detection in extreme industrial environments," in 2022 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT), Trento, Italy: IEEE, Jun. 2022, pp. 397–402. https://doi.org/10.1109/MetroInd4.0IoT54413.2022.9831517.

13

Transfer and Self-learning in Probabilistic Models

Fetze Pijlman[1,2]

[1]Signify, The Netherlands
[2]Eindhoven University of Technology, The Netherlands

Abstract

Transfer learning and self-learning are well known techniques that can separately improve probabilistic models. In this article it is investigated how to combine both methods in a single approach enabling transfer learning over different environments in which self-learning models are deployed. It is found that such learnings can be enabled through prior optimisation.

Keywords: probabilistic models, Bayesian, online learning, self-learning, prior optimisation, transfer learning.

13.1 Motivation

Probabilistic models are of interest for constructing classifiers or estimating parameters (see e.g. Murphy [1]). Examples of such applications are image classification in cameras and noise level estimation in radar sensors. These sensors are often deployed in diverse environments. This means that the underlying model could benefit from self-learning. Baye's rule offers such a method

$$P(\theta \mid X) = \frac{P(X|\theta)P(\theta)}{P(X)}, \tag{13.1}$$

where θ are the model parameters, X are the observations, and P(θ) is the prior distribution. Note that the observations X may also contain feedback

DOI: 10.1201/9788743808862-13

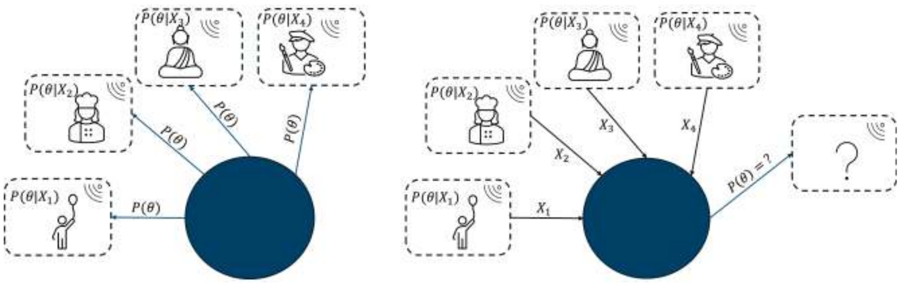


Figure 13.1 On the left-hand side a prior was sent to various diverse environments where in each environment a posterior was estimated. On the right-hand side the question is how to choose the prior for a new unknown environment when sensor events from other environments are known. ⏎

and interactions from users. A self-learning example is a motion sensor that self-learns its false positive rate.

The prior distribution is our initial expectation on the parameters prior to the observation of new data. When no historical data is available, an expert choice should be made for the prior. However, in some situations data may be available from other diverse environments. The question here is how to choose a prior for a new unseen environment when data is available from other diverse environments. This problem is illustrated in Figure 13.1.

In our motion sensor example, it may be believed that sensors in some applications have a false positive rate of once per week while in other applications the false positive rate is once per month. Note that for a new sensor deployment one cannot simply take an average of the posteriors from different environments as each posterior represents a different deployment: some posteriors may be very concentrated around a particular value which may slow down adaptation to a new environment and posteriors for different environments may have seen different amount of data (how to average that?). Moreover, averaging different posteriors is fundamentally wrong; similarly, one cannot take the average of an apple and an orange.

To solve this problem Xuan [2] discusses a graphical model containing common and custom nodes. A challenge with such an approach is that the graph itself will also change as more data becomes available. Similarly, Suder [3] splits up the parameter space into common and custom parameters. A challenge with this approach is how to decide which parameters are common or which ones are not. Pautrat [4] discusses the grouping of similar environments, each having their own prior. A challenge with such an approach is how to decide on the number of groups and when to decide on forming a new group when a new environment is substantially deviating.

Although the posteriors are fundamentally different for different environments, knowledge from different environments is still useful for determining an optimal prior for a new environment. Although the environments may be substantially different there is one aspect that they share in common which is the prior from which the probabilistic model could have started. This commonality offers an opportunity for transfer learning and it will be studied in this article.

13.2 Prior Optimisation

A choice for prior is similar to a choice for model. When having a set of models $\mathcal{M}_i$ then the probability for a model to hold given the data is given by (see e.g. Murphy [1])

$$P\left(\mathcal{M}_i|X\right) = \tfrac{P(X|\mathcal{M}_i)\ P(\mathcal{M}_i)}{P(X)}. \tag{13.2}$$

If a prior is parametrized through a hyperparameter μ then the best hyperparameter can be selected through

$$\mathrm{argmax}_{\mu} P\left(X|\mu\right)\ P\left(\mu\right). \tag{13.3}$$

In some cases, there may be no direct expectations for $P\left(\mu\right)$ as given in Equation 1.3 but there may be prior expectation for parameter θ. In those cases, an effective prior on μ can be derived. Prior to the knowledge on the prior $P\left(\theta\right)$ we take $P\left(\mu\right)$ to be uniform (first line equation below right-hand-side). The probabilities for observing a distribution $P\left(\theta\right)$ from μ is simply the product of $P\left(\theta_i \mid \mu\right)$ weighted by the amount $P\left(\theta_i\right)\Delta\theta$ (second line involves a product integral)

$$\begin{aligned} P(\mu \mid P[\theta]) &\propto P(P[\theta] \mid \mu) \\ &= \lim_{\Delta\theta \to 0} \quad {}_{i} \; P\left(\theta_i \mid \mu\right)^{P(\theta_i)\Delta\theta} \\ &= \exp \int d\theta P(\theta) \log P(\theta \mid \mu), \end{aligned} \tag{13.4}$$

where in the last step the Geometric integral relation from Volterra was used. Note that the same result can be obtained by using variational Bayesian approximation. A variational approximation of a posterior is given by

$$P(\mu \mid X) \approx \operatorname{argmax}_{q(\mu)} \int d\mu \, q(\mu) \left(\log P(X \mid \mu) - \log \tfrac{q(\mu)}{P(\mu)} \right). \tag{13.5}$$

Using the prior $P(\theta)$ for updating our knowledge on $P(\mu)$ gives

$$\begin{aligned} P(\mu \mid P[\theta]) &\approx \underset{q(\mu)}{\operatorname{argmax}} \int d\mu q(\mu) \quad \int d\theta P(\theta) \log P(\theta \mid \mu) - \log \frac{q(\mu)}{P(\mu)} \\ &\propto \exp \int d\theta P(\theta) \log P(\theta \mid \mu), \end{aligned} \tag{13.6}$$

where in the last line we assumed the initial prior $P(\mu)$ to be flat. Equation 1.4 gives for the effective prior $P(\mu)$

$$P(\mu \mid P[\theta]) = \frac{\exp \int d\theta P(\theta) \log P(\theta \mid \mu)}{\int d\mu' \exp \int d\theta P(\theta) \log P(\theta \mid \mu')} \tag{13.7}$$

In Equation 1.3, X contains all of the available data. It will be assumed that captured diversity in environments is representative for the underlying population of environments. The known environments are allowed to have different amounts of observations. As the known environments are assumed to be representative, an optimal prior for known environments will be an optimal prior for a new unknown environment.

Taking the logarithm of Equation 1.3 and labelling data from environment i by X_i one finds

$$\mu = \operatorname{argmax}_{\mu} \left[\textstyle\sum_i \log P(X_i \mid \mu) + \log P(\mu) \right]. \tag{13.8}$$

Using

$$\log P(X) = \int d\theta \, q(\theta) \ \log P(X \mid \theta) - \log \tfrac{P(\theta \mid X)}{P(\theta)} \quad , \tag{13.9}$$

with

$$\int d\theta \, q(\theta) = 1, \tag{13.10}$$

one finds

$$\begin{aligned} \mu \approx \underset{\mu}{\operatorname{argmax}} \quad {}_i & \int d\theta_i P(\theta_i \mid X_i, \mu) \left\{ \log P(X_i \mid \theta_i) \right. \\ & \left. - \log \frac{P(\theta_i \mid X_i, \mu)}{P(\theta_i \mid \mu)} \right\} + \log P(\mu) \quad . \end{aligned} \tag{13.11}$$

In case the posterior for an environment cannot be exactly determined one can approximate by (see Attias [5], Morey [6], and Tran [7])

$$\mu \approx \underset{\mu}{\mathrm{argmax}} \max_{q(\theta_i)} \quad {}_{i} \quad d\theta_i q(\theta_i) \quad \log P(X_i \mid \theta_i) - \log \frac{q(\theta_i)}{P(\theta_i \mid \mu)}$$

$$+ \log P(\mu) \tag{13.12}$$

Note that $q(\theta_i)$ approximates the posterior for each environment i in terms of the reversed KL-divergence. The reversed KL-divergence may lead to modes in $q(\theta_i)$ introducing errors. For this reason the posterior $q(\theta_i)$ should have sufficient freedom to describe all typical groups of customers.

It is of interest to study the optimal prior in case of a single deployment (or in the limit that all deployments have the same environment) and for $P(\mu)$ being constant. If one has conjugate priors then one can iteratively solve Equation 1.11 by choosing at each iteration the prior $P(\theta_i|\mu)$ to be equal to the posterior from the previous iteration $P(\theta_i|X_i, \mu)$. In the limit of convergence, the logarithm containing ratio of posterior over prior will vanish and the posterior $P(\theta_i|X, \mu)$ will strongly peak around $\mu = \mathrm{argmax}_{\theta_i} P(X|\theta_i)$. In other words, in absence of any prior belief then the best prior for a new environment that is the same as the existing environment is a strongly peaked function around the most likely parameters (as expected).

Note that strongly concentrated priors delay learnings from new data. Peaking of priors is reduced when a belief for $P(\mu)$ is included and when known environments are diverse.

13.3 Example Categorical Distribution

For categorical distributions the conjugate prior is a Dirichlet distribution of which its hyperparameter is often denoted by $\boldsymbol{\alpha}$ instead of μ that was used in this paper upto now. Given applications i some observed categories j and assuming a flat prior such that the last term $\log P(\mu)$ in Equation 1.11 disappears one obtains

$$\boldsymbol{\alpha} = \mathrm{argmax}_{\boldsymbol{\alpha}} \sum_i \log \frac{\Gamma\left(\sum_j \alpha_j\right) \quad {}_j\, \Gamma(\alpha_j + n_{ij})}{\Gamma\left(\sum_j \alpha_j + n_{ij}\right) \quad {}_j\, \Gamma(\alpha_j)}. \tag{13.13}$$

Consider the following example. In a first application one encounters 1000 items of category 0 and 2 items of category 1 while in a second application one encounters 10 items of category 0 and 2 items of category 1. Using the equation above one finds the optimal prior to be a Dirichlet distribution with $\alpha = (5.8, 0.41)$. If the previous two applications are random draws from an underlying population of applications, then the obtained Dirichlet distribution would be optimal for a new application. The ratio of the hyperparameters indicates the expected ratio of classes, the sum of the hyperparameters is an indicator for the knowledge strength.

Consider now another example in which one encounters in the first application 50 items of category 0 and 3 items of category 1, and in the second application 5 items of category 0 and 20 items of category 1. In this example the optimal hyperparameters for the Dirichlet distribution are $\alpha = (0.87, 0.60)$. The low numbers indicate a lack of knowledge which is expected given the large differences in observations between the two applications.

13.4 Conclusions and Discussion

Self-learning probabilistic models are useful for learning context and thereby self-learning increases performance in diverse environments. Similarly, transfer learning can help to use learned knowledge from one environment for another environment. In this article these two aspects are integrated into a single approach by optimising the prior.

It was found that an optimal prior can be obtained by maximising a sum of evidences over environments. The amount of data per environment is allowed to vary. In order to optimise the prior, observations from different environments need to be shared. For cameras this means sharing of images, for radar sensors it means sharing of radar signals. Sharing of sensor events may involve privacy aspects.

Although the approach leads to the best (or most optimal) prior one does need to realize that this approach carries the risk of overfitting. The situation is similar to variational Bayesian methods that yield the "best approximated" posterior which in reality may be far away from the true posterior.

Acknowledgements

EdgeAI “Edge AI Technologies for Optimised Performance Embedded Processing” project has received funding from Key Digital Technologies Joint Undertaking (KDT JU) under grant agreement No 101097300. The KDT

JU receives support from the European Union's Horizon Europe research and innovation program and Austria, Belgium, France, Greece, Italy, Latvia, Luxembourg, Netherlands, Norway.

References

[1] K.P. Murphy, "Machine learning - a probabilistic perspective", The MIT press (2012)

[2] J. Xuan, J. Lu, G. Zhang, "Bayesian Transfer Learning: An Overview of Probabilistic Graphical Models for Transfer Learning". arXiv:2109.13233 (2021)

[3] P.M. Suder, J. Xu, and D.B. Dunson, "Bayesian Transfer Learning". arXiv:2312.13484 (2023)

[4] R. Pautrat, K. Chatzilygeroudis and J.B. Mouret, "Bayesian Optimization with Automatic Prior Selection for Data-Efficient Direct Policy Search". arXiv 1709.06919 (2017)

[5] H. Attias. "A variational bayesian framework for graphical models". *neurIPS* (1999).

[6] R. D. Morey, J.W. Romeijn, J.N. Rouder, The philosophy of Bayes factors and the quantification of statistical evidence, Journal of Mathematical Psychology (2016)

[7] M.N. Tran, Trong-Nghia Nguyen, Viet-Hung Dao, A practical tutorial on Variational Bayes, arXiv: 2103.01327 (2021)

14

A Novel Hierarchical Approach to Perform On-device Energy Efficient Fault Classification

Devesh Vashishth[1], Julio Wissing[2], and Marco Wagner[3]

[1]University of Applied Sciences, Heilbronn, Germany
[2]Fraunhofer IIS, Fraunhofer Institute for Integrated Circuits, Germany
[3]University of Applied Sciences, Heilbronn, Germany

Abstract

Using convolutional neural networks (CNN) to discover sensor data patterns help predict upcoming failures in industrial machines. Traditionally, this pipeline is deployed on the cloud as deep neural networks have high computational requirements. Alternatively, an on-device deployment to make decisions locally for this pipeline could lower the energy requirements by not sending the sensor data back and forth to the cloud. However, this is challenging from an edge deployment perspective due to the limited computational resources and energy budget. To approach this issue, we propose a hierarchical architecture that leverages multiple smaller networks dividing the bigger problem into smaller sub-problems in a divide and conquer approach. With this architecture, we achieve a 9x reduction in energy consumption going from 0.045J per inference for a non-hierarchical CNN classifier to 0.005J. At the same time, our approach delivers low latency with comparable accuracy to the baseline, while running completely at the edge.

Keywords: convolutional neural networks (CNNs), convolutional neural networks (HiCNNs), Case Western Reserve University (CWRU), multiplication addition computations (MAC), neural network architecture search (NAS).

DOI: 10.1201/9788743808862-14

14.1 Introduction and Background

With the advancement of Industry 4.0, leveraging data-driven methodologies like machine learning to discover faults in industrial machine sensor data has gained rapid momentum. In this regard, fault classification using machine sensor data can be interpreted as a multifaceted task including feature extraction, detection, and classification of the specific fault type. The fundamental structure of these approaches typically involves acquiring signals in the form of sensor data from the machinery under monitoring followed by data exploration, which means that the acquired data is analysed to identify potential anomalies or deviations concerning data conformity.

This genre of manual feature extraction involves mathematical techniques like Fourier transform to extract relevant information from the sensor data for effective fault detection [3]. These methods compose features by exploiting randomness and dynamic changes in time-series signals, capturing information related to the frequency domain [8]. Later the machine learning classifier is selected and trained on this feature set to distinguish between different fault types finally. Conventionally, this entire pipeline would be deployed on energy-hungry cloud services, and the inference results are received back at sensor nodes [10]. Traditional rolling-bearing fault classification methods rely not only on manual feature extraction but are also time-consuming due to the sending and receiving of data. Deploying machine learning models directly on sensor devices enhances security by restricting the need for data transfer. Unlike the power-intensive cloud alternative (Figure 14.1), it reduces inference overhead and contributes to model efficiency [7].

Model efficiency bridges the gap between machine learning research and practical applications. This is particularly important for ultra-low-power wireless sensors, where maximizing battery life is critical to deploy them in remote areas without direct access to power. However, the challenge lies in developing energy-efficient algorithms that maintain accuracy. Increasing the energy efficiency of an algorithm, in general, impacts the performance aspect of the algorithm due to a trade-off between performance and energy efficiency [9].

In this work, we contribute a novel meta-structure to CNNs we call Hierarchical CNN (HiCNN) that splits the classification into multiple subdecisions. This process aims to reduce computational requirements per decision while achieving on-par classification performance in comparison to standard non-hierarchical classifier (flat). HiCNN utilizes training of CNN architectures to reduce feedforward computations during on-device inference and solve classification tasks. Feeding extracted features to subsequent classifiers helps reduce error propagation from higher to lower class hierarchies and removes the need for more convolutional and dense layers in lower classifiers. The primary goal of HiCNN is to capitalize on the real-time accurate results offered by CNNs while simultaneously reducing storage requirements and computations while running inference on-edge devices. Due to HiCNN's low computational requirements, it holds potential utility for deployment in remote environments constrained by limited power resources (e.g., undeveloped regions, forest locations, or geographically isolated industrial machinery).

14.2 State of the Art

In the following, we discuss related fields, starting with introducing the dataset used in this work, followed by further approaches to solving the classification problem behind it. We then discuss other approaches to reducing energy consumption and different hierarchical approaches.

14.2.1 Experimental setup

Our target application area is condition monitoring, so we are using the well-studied Case Western Reserve University (CWRU) Bearing dataset [8]. The bearing dataset is acquired by the electrical engineering laboratory of CWRU and recorded using a 2HP motor connected to a dynamometer via a torque transducer/encoder. It has 4 operating modes of machines based on different operating powers, 4 fault classes, and 3 severity classes. The bearings used in this test support the motor shaft and have been artificially damaged at different locations with fault depths ranging from 0.007" to 0.021". The vibration data included in the CWRU dataset is collected using single-axis accelerometers with a sampling rate of 12kS/s (fan-end).

14.2.2 Related work

From an industrial condition monitoring perspective, [8] highlights a key limitation: mode partitions designed for multi-mode processes often exhibit high specificity to a particular process and dataset. Therefore, achieving model generalization becomes imperative. In a similar attempt [1] used Autoencoders to generate compressed feature representations from raw data utilizing the encoder-decoder arrangement and eliminating manual feature engineering

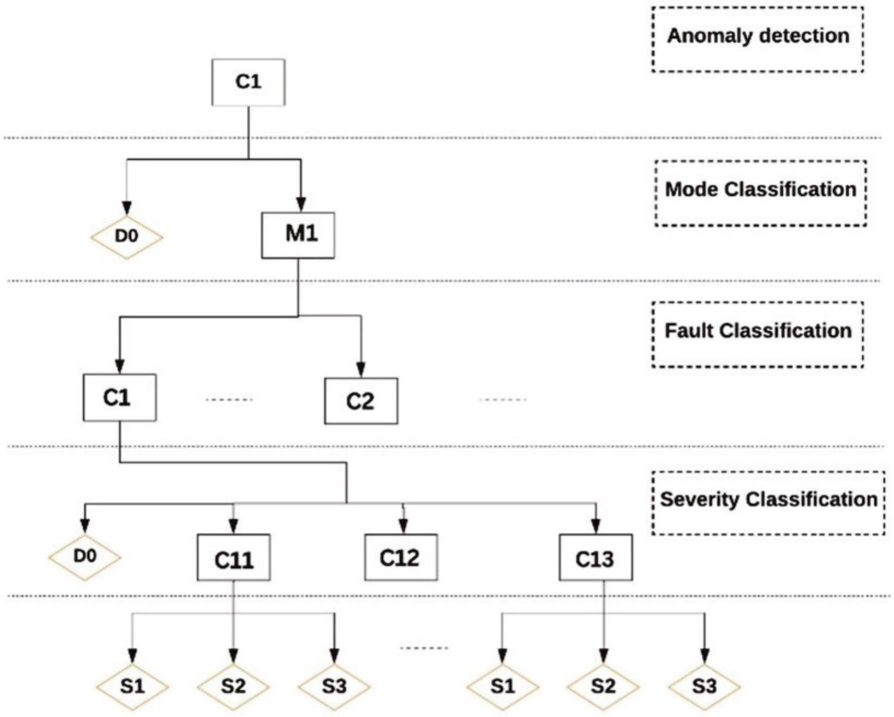


Figure 14.1 HiCNN distributed architecture with a divide and conquer approach of solving the classification task, with D0 node refering to no-fault class and S1,S2 and S3 nodes refers to the end of classification. ⏎

to classify fault classes in the CWRU dataset. However, this approach's computational demands present a significant challenge for energy-constrained edge device deployment. This is because architectures of autoencoders are potentially like CNNs including many stacked conv layers [4]. Similarly, in [11], the authors devised a 3-step classification system using three Deep neural network stacks of Auto Encoders to feed relevant features to the classes.

The first DNN stack facilitates mode partition for the different types of modes in the CWRU (Case Western Reserve) dataset. In contrast, the second DNN stack pursues feature extraction, and the third stack is proposed for severity classification. However, all these hierarchical approaches focus only on getting a good prediction accuracy of the fault classification results and lack an optimized energy-efficient approach.

Decreasing energy consumption of feedforward neural networks during inference is an active research direction, and key techniques to achieve this include pruning the architecture, Network Architecture Search (NAS), and quantization. Pruning aims to reduce energy consumption by removing unnecessary neurons before deployment. This is done by assigning scores to each neuron removing the ones with the lowest scores and subsequently fine-tuning the architecture. Depending on the type of scores, different pruning methods are realized ranging from e.g. relevance-based to unstructured. Expanding on the concept of reducing computations caused by redundancy in neural network architecture [5] performed redundant kernel removal in filters of CNN layers using unsupervised clustering by leveraging the mean instead of individual weights of created centroids to decrease total MACs. However, it is important to remember that different clustering algorithms lead to different assumptions when creating clusters [7].

Alternatively, NAS a vital field in TinyML ([8]) research, offers diverse approaches for selecting classifiers. In general, NAS searches for an optimal network architecture. For this certain metrics are used based on metrics like relevance, accuracy, or confidence scores, but mostly it is just the accuracy score. For efficient architectures multi-objective optimization is used, optimizing for accuracy and energy consumption simultaneously. In this case, surrogate metrics like inference time, memory consumption, or model size are used. NAS also utilizes evolutionary algorithms and reinforcement learning to explore and identify efficient neural architectures.

Meanwhile, employing quantization to reduce neural network weight and activation representations from 32-bit floating-point to 8-bit integers (or lower) yields a notable decrease in model size and increased computational efficiency. ([2]) utilized entropy constraints calculated for clusters of parameters by balancing the compression rate and resulting model performance in a relatively smooth and error-tolerant landscape by quantizing weights based on their distance from clusters. However, this technique requires a careful induction of heuristics to balance the performance of the model and the magnitude of quantization in the architecture.

[10] proposed a hierarchical taxonomy of mixed classifiers to address the limitations of single-classifier systems. The approach dynamically selects classical machine-learning classifiers based on performance metrics such as accuracy and energy efficiency. However, this method exhibits potential drawbacks: 1) Reliance on Feature Engineering: The effectiveness of the methodology is dependent on feature engineering tailored to specific classifier types. 2) Feature Set Selection which is meticulous and often requires

domain expertise and empirical evaluation. Therefore, we are interested in CNN-based architecture that can deliver good accuracy with less energy consumption.

14.3 Hicnn Approach

HiCNN framework introduces training in a staged architecture featuring anomaly detection as the parent classification task, instilling a top-down classification approach. The training process leverages a hierarchical approach, where each deeper hierarchy classifier is trained on the features forwarded from the mode partition classifier. The methodology is summarized and divided into three parts in the following sub-sections.

14.3.1 HiCNN training

Notably, despite potentially having more parameters than the benchmarked baseline CNN approach, HiCNN achieves significant energy savings during inference on edge devices by selectively activating only the relevant CNN networks based on the detected anomalies and fault type. This allows for further optimization of overall architectures for task-specific computational complexity. This happens because task-specific training of the CNN networks enables us to deactivate redundant parameters/parts of the network that are not required to achieve the desired inference result. Generally, the depth of CNN strengthens the CNN architectures ([6]), making them capable of finding intricated patterns in input data. At the same time, the width aspect of a layer would increase the capability to detect and extract features from input data. This increases the computational demand as the size of the architecture increases. By introducing HiCNN, we introduced a distributed approach to solving the task without the direct need to eliminate the parameters. The trade-off between the size of architecture and energy efficiency is tackled using a one-classifier per node approach. This helps scale down the network size without eliminating parameters just by training the network. As part of the black-box approach, HiCNN is trained so that its inherent structure only activates the necessary sub-classifier.

In hierarchical sensor data, obtained from different operating modes of industrial machines errors from early classification stages can propagate and be amplified through the network, resulting in unreliable overall accuracy. Hence, we devised an automatic feature forwarding to overcome this problem which eventually also contributes to reducing energy consumption. HiCNN improves adaptability to varying machine conditions and eliminates reliance on manual feature engineering. This is achieved by defining and training three individual CNN architectures node-wise within a hierarchical tree structure. During the training, learned features from the classifiers in the initial hierarchy are forwarded to the lower classifiers. This approach ensures that the relevant feature vector is forwarded to the appropriate fault mode, thus preserving overall classification accuracy/performance. The CNN architectures of HiCNN are trained on raw data, and the deeper architectures are trained on features produced by CNN architectures lying in the earlier part of the hierarchy (Figure 14.1). As part of our empirical findings, a filter size of 5 in the initial conv layer of classifier C2 (Figure 14.2) ensures the availability of enough samples to train the classifiers located lower in the hierarchy. This type of feature for- warding of HiCNN helps us lower the total computations required by HiCNN during classification as we eliminate the need for more conv layers as classification progresses to a deeper hierarchy.

14.3.2 Feature forwarding

As part of the HiCNN approach, anomaly detection functions as an event trigger within the HiCNN framework, prioritizing energy efficiency throughout its hierarchical structure by initiating classifications solely upon successful fault identification. The anomaly node architecture consists of four layers: one Conv1d (Convolutional 1-dimensional) layer, one max pooling layer, one batch normalization layer, and one full-connected layer. However, due to simpler task complexity, a classifier with one Conv1D layer is sufficient to perform the classification. This Conv1d layer consists of only one filter and a kernel size of 80. HiCNN architecture prioritizes anomaly detection by placing it as the parent classifier at the top of the hierarchical structure. This prioritization stems from data-dependent task complexity. [11]. Consequently, the HiCNN architecture employs a simple, single-layer CNN classifier for anomaly detection. This classifier acts as a gatekeeper, triggering further classifications only when an anomaly is identified. This approach is crucial for optimizing computational efficiency. By identifying anomalies early on, the HiCNN architecture avoids redundant computations in the lower levels of the hierarchy. In contrast, the baseline classifier continuously performs computations even in the state of no anomaly. This approach leads to unnecessary calculations and inefficient use of resources. This helps us activate only a leg of the HiCNN architecture (Figure 14.2) ultimately helping to avoid using all the weights of trained architecture during inference as is the case with baseline CNN architecture.

14.3.3 Baseline CNN and Hierarchical CNN

This subsection differentiates Baseline and HiCNN algorithms. A flat classification approach is represented by a single complex multilabel classifier responsible for classifying all classes at once. We constructed a baseline convolutional neural network (CNN) with a flat classification structure. This model is iteratively fine-tuned to achieve robust performance, yielding approximately 97 % accuracy on the test dataset. Subsequently, this baseline classifier serves as a benchmark for comparison against a hierarchical classification approach.

The baseline model (Figure 14.2) requires a computationally intensive architecture to accommodate the one-hot encoded representation of 28 classes. This single architecture incorporates six convolutional layers and five dense layers, independently managing the classification task while demonstrating noteworthy performance. However, the baseline approach does not optimize for computational efficiency.

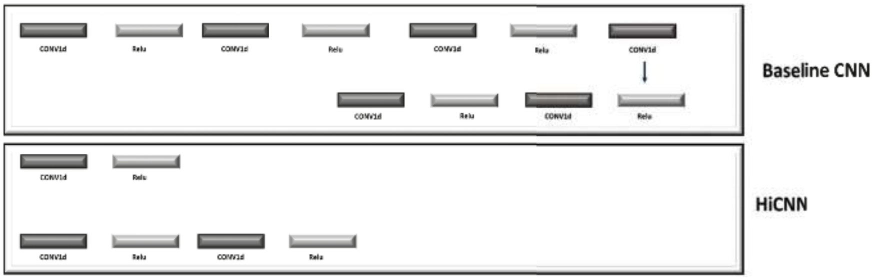


Figure 14.2 Baseline Architecture vs HiCNN architecture, indicating towards flexible architectures and number of layers used. The HiCNN architectures are used with different filter size and filter numbers. ⏎

On the other hand, as illustrated in Figure 14.1, if the initial mode corresponds to M1, only classifier M1 would be activated. Subsequent severity class determinations follow the same principle. This inherent flexibility enables the hierarchical model to manage complex classification problems by using specialized classifiers at each node. The training process within the HiCNN follows a node-wise approach, beginning with the Anomaly node (Figure 14.1). This hierarchical strategy involves the fine-tuning of classifiers at each level of the HiCNN architecture after each training epoch. Employing specialized architectures for each classification task, every component is individually trained and fine-tuned. The initial architecture focuses on fault detection, followed by mode partitioning, fault classification, and severity classification. HiCNN promotes energy efficiency while preserving flexibility by adjusting layer- specific hyper-parameters, such as filter count and dimension.

By controlling these parameters, we can regulate the volume of data propagated to the severity node, thus influencing the overall computational complexity of the HiCNN model. [13] To propagate relevant features to deeper nodes (i.e., severity classification), the mode partition CNN classifier at hierarchy level two is designed with two outputs. During training, a custom loss function outputting zero is employed for the first convolutional layer of mode classifiers (Figure 14.1) responsible for feature output. The overall accuracy of the HiCNN model is heavily influenced

by the quantity of data samples provided to the Mode partition node. For training, individual architectures are connected in a distributed, tree-like structure (Figure 14.1) using conditional logic (if-else) and then used to perform final inference on the test dataset. Beginning with the first instance of the test dataset, HiCNN dynamically selects the subsequent classifier based on predictions from the preceding stage. This process continues until a leaf node is reached, signaling the termination of the classification process. Edge devices offer limited memory space and the size of trained tensorflow models is large enough not to be able to be deployed on them. Tensorflow-Lite (TF-Lite) is a library offered by tensorflow to downscale the model's optimum to be deployed on edge devices. Therefore, to perform inference on edge using HiCNN, initial thirty-one classifiers were converted to TF-Lite format to downscale. Conditional logic (if-else) is again utilized to run hierarchical inference on the edge device.

14.4 Evaluation

In this section, contents are divided into two following sub-sections to discuss experimental setup and measurement.

14.4.1 Experimental setup

As part of the data pre-processing pipeline, a custom data generator class is created to partition the dataset into 60:20:20 ratios for training, validation, and testing. Before evaluation, input features were scaled to have zero mean and unit variance. A 512-sample window for training the HiCNN is determined to be optimal for balancing classification accuracy and latency, an important hyper-parameter for balancing energy efficiency and performance within the HiCNN architecture.

Table 14.1 Comparison between Baseline and HiCNN ⏎

Algorithm	Accuracy	Current drawn(mA)
Baseline	98%	368.8
HiCNN	95%	50.8

The window size used to train the HiCNN is critical for model performance optimization, as an inappropriate window size could adversely affect the generalization capabilities of classifiers in HiCNN due to the smaller number of data samples available for model learning leading to underfitting. Notably, while reducing the window size to 256 samples does impact HiCNN's classification accuracy (approximately 70 % performance), its energy efficiency remains the same. The smaller size of input samples to the initial architecture means the input fed to the following architectures would be scaled down proportionally due to the smaller kernel size of feature forwarding conv output layers, leading to a proportional reduction in performance due to underfitting in deeper models of hierarchy but no change in energy consumption while running inference.

14.4.2 Measurement

All the classifiers of HiCNN are trained utilizing TensorFlow 2 libraries. Subsequently, the trained models are deployed to a Raspberry Pi using the Tensor-Flow lite library [12] for inference. A dedicated hardware setup is employed for precise energy consumption measurements, incorporating a data-logging multimeter. This configuration allowed for the ongoing measurement of electrical current consumption during the classification process recorded at 100 millisecond intervals. To account for the influence of background processes and sensors, the baseline power consumption of Raspberry Pi-2b is established during idle operation. This baseline power is then subtracted from the power consumption recorded during inference, yielding energy in Joules per inference, expressed in Joules. The Raspberry Pi is powered by a 5V DC power supply, with the multimeter interfaced for data acquisition (Figure 14.3). LabVIEW orchestrated the triggering of

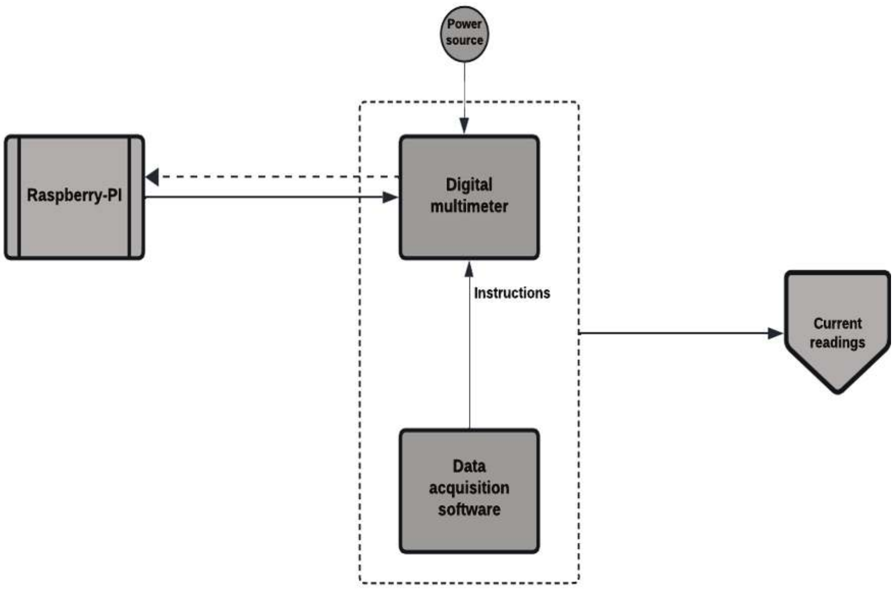


Figure 14.3 Energy measurements setup using Raspberry pi connected to multimeter. ⏎

classifications while simultaneously capturing current readings (in amperes) and corresponding timestamps (in milliseconds). The hyper-parameter window size 512 in HiCNN further guarantees HiCNN's energy efficiency regardless of the model's generalization capability. This could be beneficial for transfer learning applications. As a result, energy consumption stays constant with these variations. This occurs because the baseline algorithm performs unnecessary computations for non-anomaly cases, unlike HiCNN.

The HiCNN architecture achieves a test accuracy of approximately 95%, slightly lower than the baseline model's 98 %. However, HiCNN exhibits significant efficiency gains. TensorFlow Lite conversion and evaluation on desktop and Raspberry Pi reveal considerably faster inference times for HiCNN (380 microseconds vs. 1980 microseconds). Additionally, HiCNN demonstrates lower current draw (0.38 amperes vs. 0.46 amperes), (Figure 14.4 and Table 14.1) and reduced inference time (12 seconds vs. 86 seconds) for the classification task, consuming 197.8 Joules with baseline algorithm vs 22.8 Joules for HiCNN. This, along with lower energy per decision (0.002 joules vs. 0.0455 joules), translates to lower current draw and latency for HiCNN during every inference.

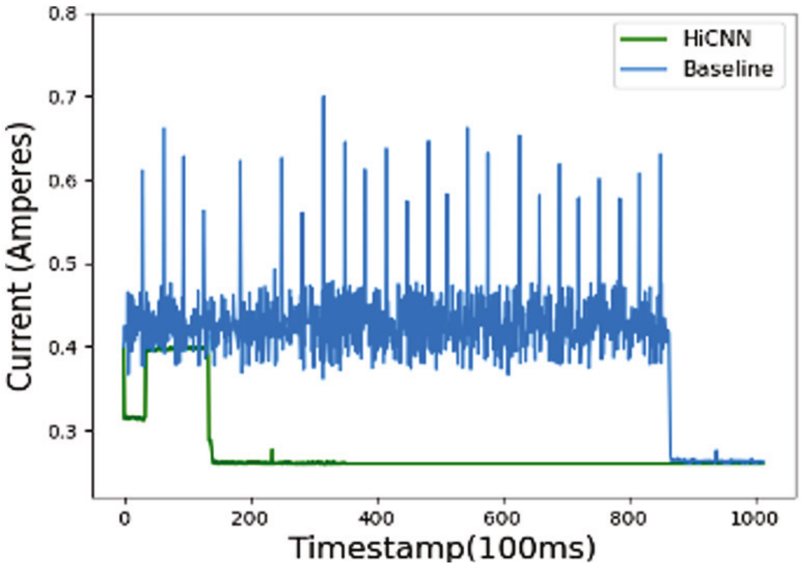


Figure 14.4 Current consumption during inference for Baseline algorithm vs Hierarchical algorithm indicating towards low latency inference by HiCNN. ⏎

14.5 Conclusion and Future work

The HiCNN framework demonstrates promising results in attaining energy efficiency. It achieves up to 80% reduction in computational overhead compared to the baseline classifier according to per-decision energy consumption. This efficiency gain is attributed to controlled computations, ensuring decreased energy consumption with architectures devised per task complexity.

These results emphasize the potential of algorithmically optimized approaches for embedded systems, enabling near-real-time monitoring with improvement in response latency. HiCNN is more suited to hierarchical datasets and even though HiCNN requires more memory, the reduced matrix operations could be accredited to the selective portions of the HiCNN network active during inference. A potential limitation arises when low data volume is propagated to lower hierarchical levels. This can lead to insufficient training data for severity classification, increasing the risk of model overfitting or underfitting in subsequent models due to inherent model complexity. Whereas HiCNN energy efficiency is completely independent of sample size, the energy consumption of the baseline model is independent of the window size till a threshold. The Baseline algorithm's performance remains relatively stable despite changes in the number of input neurons unless it drops significantly below a threshold, such as 64 neurons. Future research directions should include a comparison with manual feature engineering. Investigating the performance of HiCNN against methods employing manual feature extraction could provide insights into error propagation trade-offs. Another strategy could be hybrid cloud/edge deployment, which could help explore the potential benefits of partial algorithm/architecture deployment across cloud and on-device environments.

References

[1] "SIGCOMM Comput. Commun. Rev.," vol. 13–14, no. 5–1, 1984.

[2] D. Becking, "Compressed neural networks," 2022.

[3] E. Denton, "Networks for Efficient Evaluation L1-Norm Batch Normalization for Efficient Training of Deep Neural Networks," n.d.

[4] X. Jie, "Predictive Exit: Prediction of Fine-Grained Early Exits for Computation- and Energy-Efficient Inference," 2021.

[5] I. Jungla, "Computation reduction in CNN inference by exploiting clustering and sparsity," 2022.

[6] J. Lin, "On-Device Training Under 256KB Memory," 2022.

[7] C. T. Liu, "Computation-performance optimization of convolutional networks with redundant kernel removal," 2018.

[8] S. Neupane, "Bearing Fault Detection and Diagnosis Using Case Western Reserve University Dataset With Deep Learning Approaches: A Review," 2018.

[9] T. Priyardishani, "Conditional Deep Learning for Energy-Efficient and Enhanced Pattern Recognition," 2016.

[10] J. Wißing, "HiML edge: An energy-aware optimization framework for hierarchical machine learning in wireless sensor systems at the edge," 2022.

[11] X. Song, J. Yao, Y. Gu, and J. Cao, "A Denoising Autoencoder-Based Bearing Fault Diagnosis System for Time-Domain Vibration Signals," 2021.

[12] R. Piramuthu, V. Jagadeesh, Z. Yan, and H. Zhang, "HDCNN: Hierarchical Deep Convolutional Neural Networks for Large Scale Visual Recognition," 2015.

[13] F. Zhou, "A Novel Multimode Fault Classification Method Based on Deep Learning," 2022.

15

Discovering and Classifying Digital and Wooden Industries Products' Defects at the Edge by a Yolo/ResNet-based Approach and Beyond

Robin Faro[1], Alessandro Strano[1], and Francesco Cancelliere[2]

[1]Deepsensing SRL, Italy
[2]University of Catania, Italy

Abstract

The paper aims to present an AI-based Automated Optical Inspection (AOI) software for both digital and wooden industries developed within the EdgeAI project. Current approaches rely on centralized solutions, where the computation is performed inside the inspection machine itself. Instead, we present algorithms that work at the edge to give rise to competitive solutions to existing ones. In particular, we experiment with two different tasks of defect identification: detecting the defect position within a wooden panel by using YOLO, in which a 96% accuracy is reached. Secondly, concerning the digital industry, we perform a two-step classification between defective and non-defective microchips and then between four possible defect classes in their surface exploiting a ResNet network and obtaining a 97% accuracy. We also exploit explainability tools to understand which parts of the images caused the model's decision. After developing the AI models we port them to two less power-consuming edge devices, Nvidia Orin Nano, and Nvidia Orin AGX, observing unchanged performance.

DOI: 10.1201/9788743808862-15

Keywords: automated optical inspection, edge computing, key performance indicators, deep learning, computer vision, PCBA defect detection, wooden product defect detection, NVIDIA Jetson ORIN.

15.1 Introduction

The paper aims to present an AI-based AOI software developed within the EdgeAI project for defect detection of the PCBAs used in the digital industry to be implemented at the edge with the main expected outcome of being a viable and cost-effective solution for the inspection of many industrial products, not only digital boards. In particular, these algorithms are at the core of the AOI solution shown in Figure 15.1 where visual testing is done at the edge and learning in the cloud. This solution, as illustrated in [1], can reduce purchase and power consumption costs without increasing latency. Indeed, in this architecture, learning is done on the cloud but several tests suitable for highlighting groups of defects can be performed in parallel by competing boards thus decreasing latency.

A solution similar to the one adopted in the EdgeAI project has been proposed by Advantech where the NVIDIA board is replaced by the MIC-72 but without discussing the algorithms used in practice for defect detection. On the contrary, several algorithms have been proposed in the literature to manage the abovementioned problems using GPUs. From the literature, we found that these are mainly optimized versions of the DL-powered YOLO algorithm [2]. The mAP (mean Average Precision) of the latest proposed algorithms goes beyond 99%, i.e. 99.17% in [3], 99.5% in [4] , and 99.71%

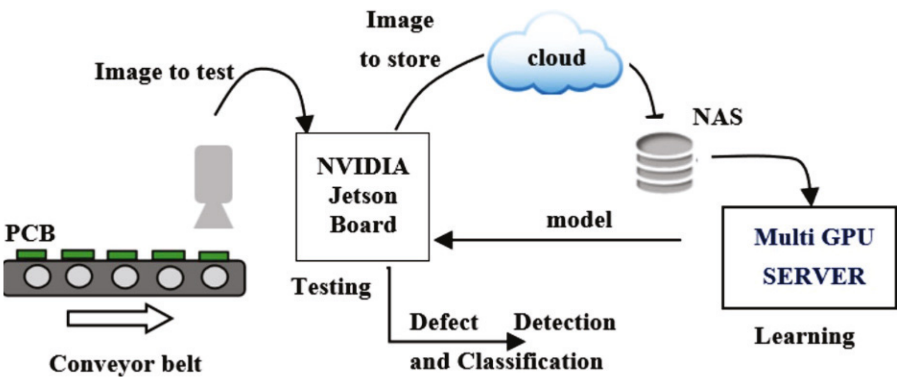


Figure 15.1 An AOI solution consisting of an edge board for testing and a GPU server for learning. ⏎

in [5]. However, this comparison is only indicative since the mentioned precision values were not achieved using the same data set. Also, it is not shown the performance degradation passing from a solution implemented on GPUs to real testing done using edge devices.

The feasibility of implementing YOLO-based algorithms for edge tests has been recently shown in literature, for example, in the YOLO implementation on NVIDIA Jetson TX2 illustrated in [6] which is characterized by satisfactory precision performance, that is, mAP0.5 = 98%. This is a relevant starting point for our implementations aiming at solving two open problems:

- To what extent very accurate algorithms can be implemented for edge tests using more performant NVIDIA boards, such as JETSON ORIN, for the quality control of PCBAs to be used in applications where the constraint of near-zero defects is required
- How to implement such algorithms on less expensive boards such as STM32

We decided to conduct experiments on wood and digital industries products which are also cases of study in the context of EdgeAI project. We start by performing wood defect detection on a publicly available dataset. Secondly, considering the absence of a significant dataset on defective PCBAs (which are the main focus of the project), we conducted preliminary experiments on advanced packaging microchips. Finally, we export the models into edge devices comparing the performance of the ported models in terms of latency, power consumption, and accuracy.

15.2 Related Works

Wood Defect Detection

The idea of an AI-based detection of wood superficial imperfections was introduced several years ago. For example, [7] tried to catch the presence of a defect in Pinus lumber. The dataset was small and composed of 400 training images and 100 test images, two different learning approaches were used: Neural Networks and Support Vector Machines. Even if these algorithms are pretty simple, interesting results were achieved, with a 97% accuracy. In [8] this problem was faced by using a cascade of Adaboost classifiers. First, he tried to detect the presence of blue stain, then of decay, and finally of cracks. The experiment was conducted on a limited collection of about 100 images, which presented 300 examples of defects and the best result obtained was a 12% error rate. Subsequently, [9] decided to exploit a VGG16 model for this task, obtaining really high performance on a 6 common defects dataset. They also performed data augmentation on the original 1200 available pictures and trained a Mix-FCN model that achieved a 91% pixel accuracy.

A direct application of the above-mentioned YOLO architecture was instead done by [10], who aimed to distinguish among 4 different typologies of defects, obtaining an 88% mAP. Furthermore, [11] proposed a modified version of YOLOv7 for wood floor small defect detection which reaches a 94% mAP. Similarly, [12] showed that a backbone-modified version of YOLOv7 reached an mAP of 81% on our same public dataset.

In summary, wood defect detection is a big challenge considering the various shapes, positions, and possible combinations of the existing industrial defects, which could worsen the performance of a model trained to catch only a part of them. Hence our goal is to exploit state-of-the-art error detection methods in order to increase the amount of defects that our model is able to spot, without penalizing reliability and above all inference time, which is a crucial parameter in industrial world applications.

Chip Surface Defect Detection

The detection of surface defects in chips is crucial for ensuring high-quality production in the semiconductor industry. Historically, methods for detecting these defects relied on image processing techniques. For example,

[13] used a combination of grayscale transformation, mathematical morphology, and pattern recognition to detect defects on printed circuit boards, achieving high detection speed. Similarly, [14] employed filtering methods with Support Vector Machines to classify defects. These methods are fast but often struggle with generalization across different defect types due to their reliance on manually set parameters.

Deep learning has become increasingly popular in recent years and to address this limitation its advancement has introduced more robust methods for defect detection. Most modern techniques are divided into three categories: classification, segmentation, and object detection. For instance, [15] utilized an improved Spatial Pyramid Pooling Network (SPPNet) for defect classification, while [16] applied a 3D convolutional neural network (CNN) to classify defects on wafers. Object detection networks, such as Faster R-CNN and YOLO, are increasingly favored due to their speed and accuracy. For example, [17] used Faster R-CNN to detect multiple types of defects in steel and concrete structures, while [18] enhanced YOLOv3 with a Group Pyramid Pooling module for rapid detection of PCB surface defects.

Moreover, other deep learning-based techniques such as the SSD (Single Shot MultiBox Detector) and its variants have been explored for defect detection. [19] proposed an SSD-based method that utilized shallow features to detect smaller defects, although it faced challenges in detecting finer details. Later improvements such as DSSD [20] (Deconvolutional Single Shot Detector) achieved better detection accuracy by incorporating deconvolution layers to add more contextual information, although at the cost of increased processing time.

To tackle the problem of small object detection, [21] proposed using YOLOv3 with multi-scale feature maps, but it showed limited effectiveness for small defects due to insufficient deep feature extraction. An improvement came with the introduction of YOLOv4, which balanced speed and accuracy better than previous models. However, small object detection remained a challenge. In response, [22] developed SO-YOLO, a modified version of YOLOv4, aimed at detecting small-scale chip defects by improving shallow feature fusion. This approach achieved superior performance with an 86% mAP, surpassing YOLOv4 and even YOLOv5.

The continuous evolution of deep learning methods has also seen the integration of attention mechanisms to improve defect detection. For example, [23] proposed a weakly supervised detection framework to predict the location and probability of defects using a small dataset, achieving 99.5% accuracy.

These approaches underscore the ongoing challenges in defect detection, particularly for small objects, and highlight the importance of balancing accuracy with inference speed in real-world industrial environments.

Deployment of AI Models at the Edge

Deploying AI models at the edge has become an increasingly popular solution due to its ability to bring computation closer to data sources, reducing latency and reliance on centralized infrastructure.

In edge computing, AI models are deployed on local, resource-constrained devices, such as edge boards or embedded systems like Nvidia Orin Nano or Orin AGX. These devices are capable of executing complex deep learning models directly at the point of data collection, enabling faster response times and reducing the load on cloud services. This approach is especially beneficial in industrial settings where real-time defect detection is crucial, such as quality control in manufacturing, where any delay in detecting defects can result in significant costs.

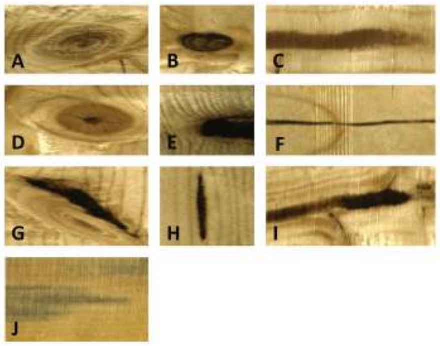
One of the key challenges in deploying AI models at the edge is the limitation of computational power and energy resources. Models need to be optimized to balance inference accuracy with speed while keeping resource consumption in check. Techniques such as model quantization, pruning, and using more efficient architectures like YOLO for object detection and ResNet for classification have proven effective in ensuring that edge devices maintain high performance without compromising on precision. The deployment of these models allows for effective parallel testing of multiple samples, leading to decreased latency compared to centralized processing, where communication delays and network reliability can introduce bottlenecks. Moreover, edge deployment supports data privacy and security by processing data locally, which is often a requirement in industrial settings.

15.3 Spotting Defects in Wood Industry Products

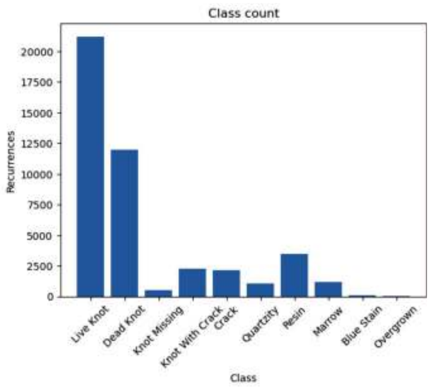
15.3.1 Defect Detection Dataset

First, we try to identify the exact position of a defect within an image of a wooden panel. To do so, we exploit a publically available dataset containing a total of 20275 images: 1992 images of sawn timbers without any defects and 18283 timber images with one or more surface defects. On average, there are 2.2 defects per image, while only 6.7% of images contain more than three defects. The highest occurrence of defects is 16 defects per image. The dataset includes the following wood defects:

1. **Live Knot:** A portion of a tree branch incorporated into the trunk, appearing as a circular or oval area of darker wood. Live knots are solid and firmly attached but may cause irregular grain patterns.
2. **Dead Knot:** Similar to a live knot, but from a non-living tree. Dead knots result from a branch that died and fell off, leaving a void filled with resin or bark, often differing in color or texture from the surrounding wood.
3. **Knot Missing:** The absence of a knot where one would typically be expected, leads to a more uniform wood appearance.
4. **Knot With Crack:** A knot that contains a crack or split within it.
5. **Crack:** A separation or break in the fiber structure of the wood.
6. **Quartzity:** The presence of quartz or silica deposits in the wood, appearing as small, translucent or whitish mineral inclusions.
7. **Resin:** The presence of sticky or resinous substances, occurring naturally or due to injury or stress, often appearing as pockets or streaks of amber-colored substance.



(a) Examples of wood defects



(b) Distribution of defects in the dataset

Figure 15.2 Visual representation of wood defects and their distribution. ⏎

8. **Marrow:** Soft, spongy tissue found in the central portion of the tree trunk, lighter in color and softer than the surrounding wood.
9. **Blue Stain:** A bluish discoloration caused by fungal or bacterial growth, leaving pigments that produce a blue or grayish tint in the wood.
10. **Overgrown:** Abnormal growth of wood fibers, resulting in irregular or distorted patterns, often due to hormonal imbalances or stress on the tree.

Some examples of the defects are shown in Figure 15.2 a while their distribution can be observed in Figure 15.2 b. It's important to underline the considerable unbalance between the classes, where the first two are dominant with respect to the other eight. Considering that our main concern is to spot defective elements rather than understanding their nature and that this class imbalance, together with the innate similarity between defects, makes it hard to distinguish among all of them, we decide to combine all of them under a general 'Defect' class.

15.3.2 Experiments and Results

The object detection model we decided to exploit is YOLO[24]. This model, whose name stands for You Only Look Once, was revolutionary in the field of object detection since it was able (starting from its first release) to output both ROIs (Regions of Interests) and their classification after just one forward pass of the input image through the network. In our case, we finetuned one of the last available releases of YOLO, which is YOLOv8, in its medium version

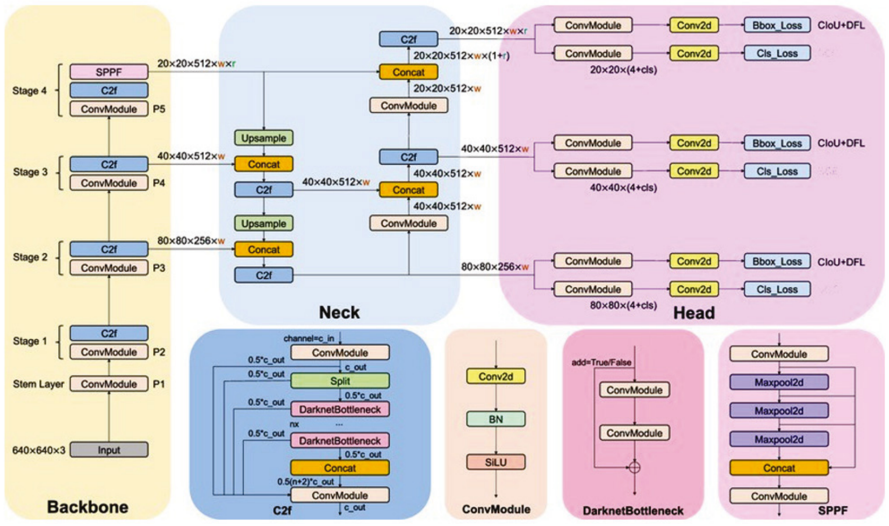


Figure 15.3 YOLOv8 architecture. ⏎

(around 26 million parameters). In general YOLO architecture (fully shown in Figure 15.3)is made up of three main blocks:

- Backbone → This is a CNN whose main role is feature extraction. It is composed of several stages which progressively reduce the spatial resolution while increasing the number of channels. Each of them relies on the so-called C2f block, which is in turn based on the Darknet Bottleneck, made up of 2 convolution blocks and a skip connection.
- Neck → This block aggregates information from different stages in order to improve the capacity of the model to recognize objects of different sizes. It is also based on the C2f block.
- Head → This component processes the aggregated features coming from the neck passing them through convolution blocks responsible for the final predictions of both bounding box and classes

We trained the model for 100 epochs and in terms of results, a 95% precision and a 94% recall are obtained, while the mAP_{50} reached a 96% value. Some of the predictions on the validation set are shown in Figure 15.4 while the progress of the abovementioned metrics during the training, as well as the loss functions on train and validation, is shown in Figure 15.5

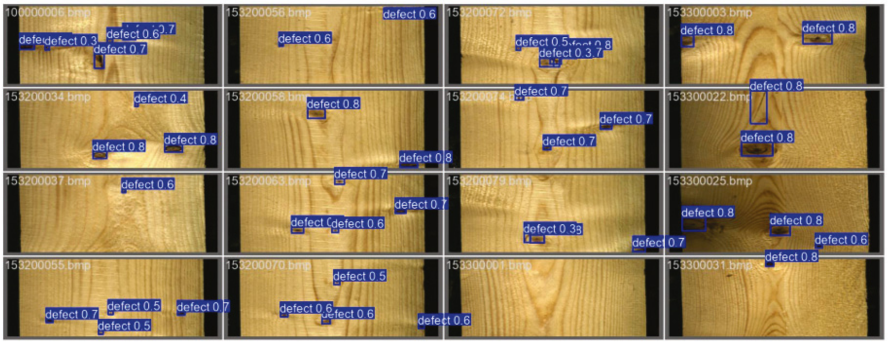
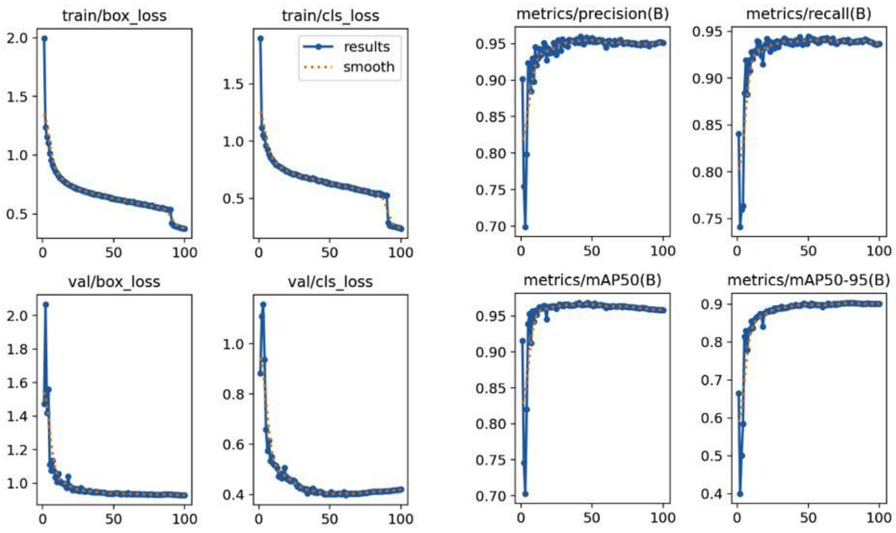


Figure 15.4 Example of prediction on validation set. ⏎



(a) Losses on train and validation sets (b) Precision, Recall and Mean Average Precison

Figure 15.5 Training metrics of YOLOv8 model. ⏎

15.4 Spotting Defects in Digital Industry Products

In this section, we focus on the identification and classification of chip surface defects. Given the challenges associated with acquiring a comprehensive dataset, we employed the one introduced by Wang et al. [25], which, despite its modest size, provides valuable insights into the defect detection process.

15.4.1 Defect Detection and Classification Dataset

The selected dataset includes a total of 2763 images of chip surfaces, where 2000 images do not contain any defects, and 763 images present one among four possible defects. We implemented a two-step classification approach, aimed to streamline the industrial process and enhance the efficiency of quality control. The initial phase involves a binary classifier designed to quickly and accurately filter out defect-free chips. This step serves as a critical screening process, ensuring that only chips identified as non-defective continue through the subsequent stages of the industrial process. By doing so, we effectively reduce the computational load and focus further inspection efforts only on potentially problematic chips. This approach is particularly valuable in high-throughput manufacturing environments, where minimizing delays and optimizing resource allocation are crucial. For the chips flagged as defective in the first step, a more granular analysis is then conducted in the second step. This phase involves classifying the specific type of defect present on the chip surface. The detailed classification not only aids in determining the exact nature of the defect but also provides essential insights that can be used for root cause analysis. In particular, the dataset contains the following defect classes:

- NO_DIE (6b) represents a unique defect scenario where the actual error lies in the absence of the soldered chip: the chip is missing on the substrate. The absence of a chip can be attributed to manufacturing faults, such as misalignment during the chip placement process or incorrect soldering. These errors can occur due to equipment malfunction, human error, or process inconsistencies.
- DIE_INK (6c) includes chips with internal ink stains: these defects can arise from ink deposition errors or contamination during the manufacturing process. Detecting and classifying these internal defects is essential for ensuring the integrity and reliability of the chip's functionality.
- DIE_BROKEN (6d) involves chips with visible breaks or fractures along their edges. These breaks can occur during the manufacturing process or as a result of external factors. Detecting and accurately localizing these broken edges is crucial for quality control and identifying potential manufacturing issues.

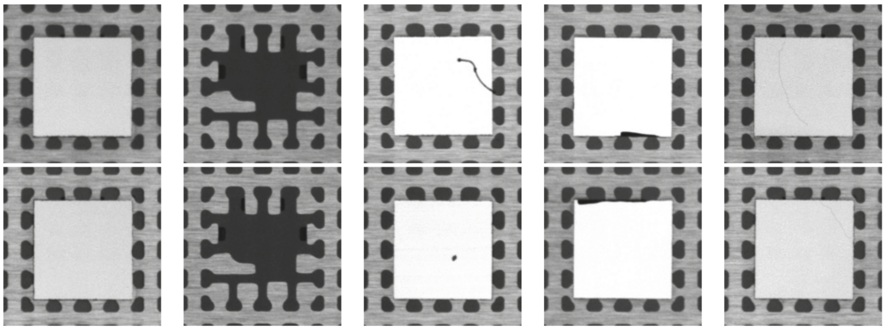

(a) DEFECT_FREE (b) NO_DIE (c) DIE_INK (d) DIE_BROKEN (e) DIE_CRACK

Figure 15.6 Defect-free chip and four common chip surface defects

Table 15.1 Distribution of the dataset

DEFECT_FREE	NO_DIE	DIE_INK	DIE_BROKEN	DIE_CRACK
2000	100	135	42	486

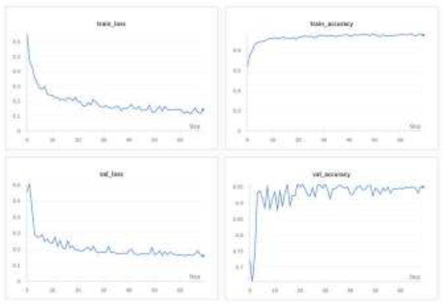
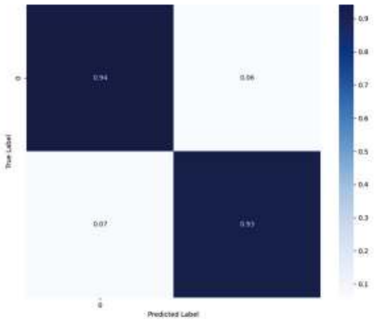
- DIE_CRACK (6e) represents chips with cracks occurring internally within the chip structure. These cracks can stem from stress during fabrication, handling, or environmental factors. Detecting and characterizing these cracks aids in identifying structural weaknesses and preventing potential chip failures.

The distribution of the different classes in the dataset is the following (1):

15.4.2 Experiments and Results

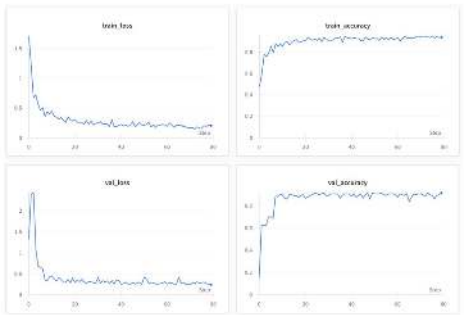
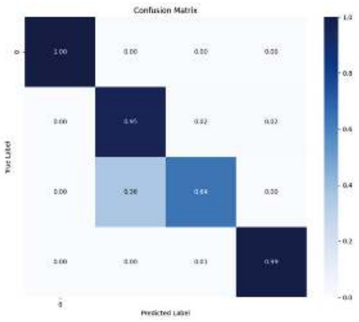
The model we used to perform classification is ResNet [26], in particular the version 18 layers deep (namely ResNet-18). It introduced the innovative concept of “skip connection", which connects the activations of a given layer to further layers by skipping some intermediate layer, thus alleviating the issue of vanishing gradient. The binary classifier was trained for 70 epochs and obtained a 94.5% accuracy and 94% recall, precision, and F1 score. The progress of accuracy and loss function on train and validation is shown in Figure 15.7 a

As regards the second phase classifier, it was trained for 80 epochs obtaining 97% accuracy, 87% recall, and 89% F1 score, as shown in Figure 15.8 a

(a) Loss and accuracy on train and validation sets (b) Confusion Matrix: 0 Defect Free - 1 Defective

Figure 15.7 Metrics' trends during training and test result. ⏎

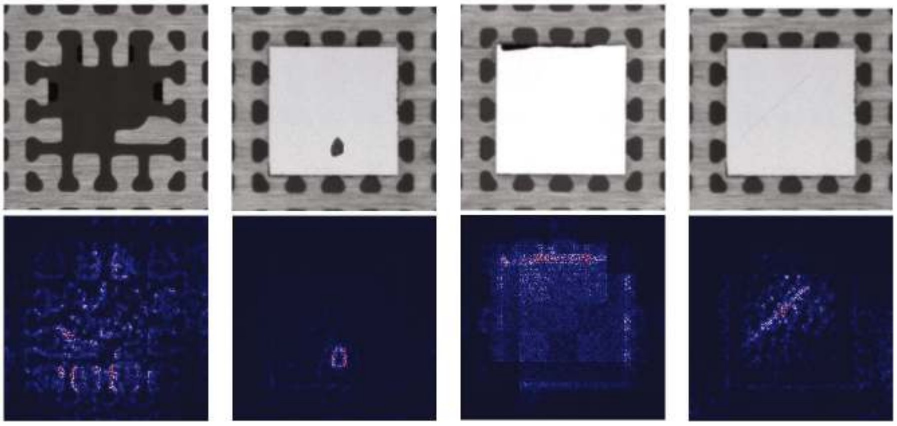
(a) Loss and accuracy on train and validation sets (b) Confusion Matrix: 0 NO_DIE - 1 DIE_INK - 2 DIE_BROKEN - 3 DIE_CRACK

Figure 15.8 Metrics' trends during training and test result ⏎

15.4.3 XAI Analysis: insigths into ResNet-18 using Grad-CAM

Explainable AI methods, often referred as XAI, are essential for enhancing the transparency and interpretability of deep learning models by providing insights into their decision-making processes. Grad-CAM (Gradient-weighted Class Activation Mapping) [27] visualizes the regions of an input image that most influence the model's predictions by highlighting the gradients flowing into the convolutional layers, thus allowing us to identify the specific areas of the chip surface on which the model focuses. Below, we provide examples of the Grad-CAM results for each defect class:

- NO_DIE: The activation maps effectively highlight the missing chip region, providing a clear indication of the defect (Figure 15.9 a).



(a) NO_DIE (b) DIE_INK (c) DIE_BROKEN (d) DIE_CRACK

Figure 15.9 Grad-CAM activation maps for different chip surface defect classes. ⏎

- DIE_INK: The maps emphasize the areas surrounding ink stains, accurately identifying the relevant regions for defect classification (Figure 15.9 b).
- DIE_BROKEN: It's the least represented class and in fact the maps show some uncertainty in pinpointing the exact broken regions, reflecting challenges observed in the confusion matrix and indicating potential areas for model improvement (Figure 15.9 c).
- DIE_CRACK: The activation maps clearly outline the crack, demonstrating the model's proficiency in localizing and detecting this defect (Figure 15.9 d).

15.5 Porting of the Models on Edge Devices

Porting AI models to edge devices is a critical step in realizing efficient, real-time solutions, especially in industrial settings. The edge devices used in this study include Nvidia Orin Nano and Nvidia Orin AGX, which are specifically chosen for their balance between computational power and energy efficiency. The process of porting models, such as YOLO for defect detection and ResNet for classification, involved converting the models to ONNX (Open Neural Network Exchange) format to ensure they could run efficiently on these resource-constrained devices while maintaining high accuracy. ONNX provides a unified format that allows models to be optimized for different hardware platforms, making them more portable and reducing dependency on specific frameworks. By converting to ONNX, we were able to leverage hardware acceleration features available on our tested edge devices, in which we observed improved inference speed and reduced latency.

The deployment tests demonstrated that both devices could perform real-time inference, with the Orin AGX showing a higher throughput due to its superior GPU capabilities. However, the Orin Nano, being more cost-effective and energy-efficient, also provided satisfactory performance for applications where slightly lower throughput was acceptable. The deployment experiments showed that the models could achieve near-identical accuracy compared to their performance in a centralized server environment, proving the viability of using edge devices for industrial defect detection.

In addition, an important aspect of edge deployment is ensuring low latency and robustness under varying conditions. The edge devices managed to maintain high accuracy with inference times well within the acceptable range for real-time operations, demonstrating their suitability for on-site, autonomous quality inspection. By processing data locally, the system also ensures data privacy, which is crucial in industries dealing with sensitive information, such as PCB industries that deal with pre-production boards.

Overall, porting the models to edge devices proved successful, providing a viable solution for decentralized, real-time defect detection in industrial applications. The performance metrics, as shown in Tables 15.2 and 15.3, highlight the trade-off between inference speed and power consumption for each device. The Orin AGX, with its higher power consumption, offers better inference times, while the Orin Nano provides a more energy-efficient alternative suitable for less demanding applications. The use of powerful yet efficient edge devices like the Orin Nano and Orin AGX resulted in a system capable of high performance under the constraints typical of edge computing environments.

Table 15.2 Performance of Nvidia Orin Nano for model deployment ⏎

Model	**Inference Time (ms)**	**Power Consumption (W)**
Chip Defect/No Defect	25.1	11.2
Chip Defect Classification	64.5	13.4
Wood Defect Classification	72.3	14.2

Table 15.3 Performance of Nvidia Orin AGX for model deployment ⏎

Model	**Inference Time (ms)**	**Power Consumption (W)**
Chip Defect/No Defect	22.0	15.6
Chip Defect Classification	44.4	21.5
Wood Defect Classification	50.4	22.5

15.6 Conclusions and Future Works

In this paper, we presented an AI-based Automated Optical Inspection solution designed for both the digital and wood industries, focusing on defect detection at the edge. Through a series of experiments, we explored the application of YOLOv8 for wood defect detection and ResNet for chip surface defect classification. The results demonstrated that our proposed approach achieved high accuracy, precision, and recall, proving the effectiveness of deep learning models in industrial defect detection tasks. We also presented an explainability analysis of the ResNet prediction, which can help in understanding the region of the image that led to the model's decision.

Moreover, by porting the models to edge devices such as the Nvidia Orin Nano and Orin AGX, we successfully validated the feasibility of running these complex models in a resource-constrained environment without compromising performance, enabling real-time defect detection. We showed that the edge deployment maintains unvaried the accuracy of the models while reducing latency and power consumption, which are critical aspects in the industrial setup.

Future works will focus on several aspects to enhance our solution, for example:

- The next step involves integrating our solution into a real-time industrial setup. This will involve testing the models in environments where objects move on conveyor belts, and high-speed cameras capture images of products for defect detection. Such real-world trials will help assess the models' ability to handle real-time constraints, such as varying lighting conditions, motion blur, and differences in defect types and positions. By doing so, we aim to further optimize the system for seamless operation in an actual production line.
- While our models achieved high accuracy, future efforts will focus on enhancing the quality of training data. This involves not only increasing the quantity of data but also ensuring that it represents a wide range of defects across different types of products and environments. High-quality, diverse data is critical for improving the generalization capabilities of the models, reducing false positives and negatives, and adapting to unseen defect types. In particular, augmenting the dataset with hard-to-detect defects and edge cases will be crucial for improving the system's robustness.
- Facing directly the PCB issue by generating a dataset suitable for detecting defects in these boards, either real or synthetic. Successively,

testing both the one-stage and the two-stage solution presented in the paper. In particular, in the second case, we imagine a first model able to detect each PCB internal component (i.e. resistors, capacitors...) and then several “expert" classifiers that will be able to decide whether a component is correctly mounted or not.

- We aim to explore multi-task learning approaches where the models can detect and classify defects from multiple industries simultaneously, improving the efficiency and flexibility of the AOI system.

Acknowledgements

This work was supported by Chips Joint Undertaking EdgeAI project. The project EdgeAI “Edge AI Technologies for Optimised Performance Embedded Processing” is supported by the Chips Joint Undertaking and its members including top-up funding by Austria, Belgium, France, Greece, Italy, Latvia, Netherlands, and Norway under grant agreement No. 101097300.

References

[1] AAEON. *AI@Edge: AI Vision in Automated Optical Inspection.* Available at: http://www.aaeon.com/ru/ai/boxer-6841m-aoi-app-story.2023. http://www.aaeon.com/ru/ai/boxer-6841m-aoi-app-story.

[2] Juan Terven, Diana-Margarita Córdova-Esparza **and** Julio-Alejandro Romero-González. “A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS”. **in** *Machine Learning and Knowledge Extraction*: 5.4 (**november** 2023), 1680–1716. issn: 2504-4990. 10.3390/make5040083. http://dx.doi.org/10.3390/make5040083.

[3] JiaYou Lim, JunYi Lim, Vishnu Monn Baskaran **and** Xin Wang. “A deep context learning based PCB defect detection model with anomalous trend alarming system”. **in** *Results in Engineering*: 17 (2023), **page** 100968. issn: 2590-1230. https://doi.org/10.1016/j.rineng.2023.100968. https://www.sciencedirect.com/science/article/pii/S2590123023000956.

[4] Yinchao Du, Jiangpeng Chen, Han Zhou, Xiaoling Yang, Zhongqi Wang, Jie Zhang, Yuechun Shi, Xiangfei Chen **and** Xuezhe Zheng. “An automated optical inspection (AOI) platform for three-dimensional (3D) defects detection on glass micro-optical components (GMOC)”.

in *Optics Communications*: 545, 129736 (**october** 2023), **page** 129736. 10.1016/j.optcom.2023.129736.

[5] Hongjin Zhu, Lina Xing, Honghui Fan **and** Tao Wu. *New PCB Defect Identification and Classification Method Combining MobileNet Algorithm and Improved YOLOv4 Model*. **april** 2022. 10.21203/rs.3.rs-154 4671/v1.

[6] Shaojun Song, Junfeng Jing, Yanqing Huang **and** Mingyang Shi. "EfficientDet for fabric defect detection based on edge computing". **in** *Journal of Engineered Fibers and Fabrics*: 16 (**april** 2021), **page** 155892502110083. 10.1177/15589250211008346.

[7] P. Cavalin, L. S. Oliveira, A. L. Koerich **and** A. S. Britto. "Wood Defect Detection using Grayscale Images and an Optimized Feature Set". **in** *IECON 2006 - 32nd Annual Conference on IEEE Industrial Electronics*: 2006, **pages** 3408–3412. 10.1109/IECON.2006.347618.

[8] Selman Jabo. "Machine vision for wood defect detection and classification". **in** *M.S Thesis, Chalmers University of Technology*: (2011).

[9] Ting He, Ying Liu, Chengyi Xu, Xiaolin Zhou, Zhongkang Hu **and** Jianan Fan. "A Fully Convolutional Neural Network for Wood Defect Location and Identification". **in** *IEEE Access*: 7 (2019), **pages** 123453–123462. 10.1109/ACCESS.2019.2937461.

[10] Wei-Han Lim, Mohammad Babrdel Bonab **and** Kein Huat Chua. "An Optimized Lightweight Model for Real-Time Wood Defects Detection based on YOLOv4-Tiny". **in** *2022 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS)*: 2022, **pages** 186–191. 10.1109/I2CACIS54679.2022.9815274.

[11] Wenqi Cui, Zhenye Li, Anning Duanmu, Sheng Xue, Yiren Guo, Chao Ni, Tingting Zhu **and** Yajun Zhang. "CCG-YOLOv7: A Wood Defect Detection Model for Small Targets Using Improved YOLOv7". **in** *IEEE Access*: 12 (2024), **pages** 10575–10585. 10.1109/ACCESS.2024.3352 445.

[12] Rijun Wang, Yesheng Chen, Fulong Liang, Bo Wang, Xiangwei Mou **and** Guanghao Zhang. "BPN- YOLO: A Novel Method for Wood Defect Detection Based on YOLOv7". **in** *Forests*: 15.7 (2024). issn: 1999-4907. 10.3390/f15071096. https://www.mdpi.com/1999-4907/15/7/1096.

[13] Zhichao Liu **and** Baida Qu. "Machine vision based online detection of PCB defect". **in** *Microprocessors and Microsystems*: 82 (2021), **page** 103807. issn: 0141-9331. https://doi.org/10.1016/j.micpro.2020.1038 07. https://www.sciencedirect.com/science/article/pii/S0141933120309 522.

[14] Cong Li, Hui-Qing Lan, Ya-Nan Sun **and** Jun-Qiang Wang. "Detection algorithm of defects on polyethylene gas pipe using image recognition". **in** *International Journal of Pressure Vessels and Piping*: 191 (2021), **page** 104381. issn: 0308-0161. https://doi.org/10.1016/j.ijpvp.2021.104381. https://www.sciencedirect.com/science/article/pii/S030801612100079X.

[15] Yufeng Shu, Bin Li **and** Hui Lin. "Quality safety monitoring of LED chips using deep learning- based vision inspection methods". **in** *Measurement*: 168 (2021), **page** 108123. issn: 0263-2241. https://doi.org/10.1016/j.measurement.2020.108123. https://www.sciencedirect.com/science/article/pii/S0263224120306618.

[16] Nikolaos Dimitriou, Lampros Leontaris, Thanasis Vafeiadis, Dimosthenis Ioannidis, Tracy Wotherspoon, Gregory Tinker **and** Dimitrios Tzovaras. "A Deep Learning framework for simulation and defect prediction applied in microelectronics". **in** *Simulation Modelling Practice and Theory*: 100 (2020), **page** 102063. issn: 1569-190X. https://doi.org/10.1016/j.simpat.2019.102063. https://www.sciencedirect.com/science/article/pii/S1569190X19301947.

[17] Young-Jin Cha, Wooram Choi, Gahyun Suh, Sadegh Mahmoudkhani **and** Oral Büyüköztürk. "Autonomous structural visual inspection using region-based deep learning for detecting multiple damage types". **in** *Computer-Aided Civil and Infrastructure Engineering*: 33.9 (2018), **pages** 731–747.

[18] Sanli Tang, Fan He, Xiaolin Huang **and** Jie Yang. "Online PCB defect detector on a new PCB defect dataset". **in** *arXiv preprint arXiv:1902.06197* : (2019).

[19] Yiting Li, Haisong Huang, Qingsheng Xie, Liguo Yao **and** Qipeng Chen. "Research on a surface defect detection algorithm based on MobileNet-SSD". **in** *Applied Sciences*: 8.9 (2018), **page** 1678.

[20] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi **and** Alexander C Berg. "Dssd: Deconvolutional single shot detector". **in** *arXiv preprint arXiv:1701.06659* : (2017).

[21] Mingjie Liu, Xianhao Wang, Anjian Zhou, Xiuyuan Fu, Yiwei Ma **and** Changhao Piao. "Uav-yolo: Small object detection on unmanned aerial vehicle perspective". **in** *Sensors*: 20.8 (2020), **page** 2238.

[22] Haixin Huang, Xueduo Tang, Feng Wen **and** Xin Jin. "Small object detection method with shallow feature fusion network for chip surface defect detection". **in** *Scientific reports*: 12.1 (2022), **page** 3914.

[23] Liang Xu, Shuai Lv, Yong Deng **and** Xiuxi Li. “A weakly supervised surface defect detection based on convolutional neural network”. **in** *IEEE Access*: 8 (2020), **pages** 42285–42296.

[24] Joseph Redmon, Santosh Divvala, Ross Girshick **and** Ali Farhadi. *You Only Look Once: Unified, Real- Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV]. https://arxiv.org/abs/1506.02640.

[25] Shuo Wang, Hongyu Wang, Fan Yang, Fei Liu **and** Long Zeng. “Attention-based deep learning for chip-surface-defect detection”. **in** *The International Journal of Advanced Manufacturing Technology*:121.3 (2022), **pages** 1957–1971. issn: 1433-3015. 10.100 7/s00170-022-09425-4. https://doi.org/10.1007/s00170-022-09425-4.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren **and** Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. https://arxiv.org/abs/1512.03385.

[27] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh **and** Dhruv Batra. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization”. **in** *International Journal of Computer Vision*: 128.2 (2020), **pages** 336–359. issn: 1573-1405. https://doi.org/10.1007/s11263-019-012 28-7.

16

Conscious Agents Interaction Framework for Industrial Automation

Polina Ovsiannikova[1] and Valeriy Vyatkin[1,2]

[1] Aalto University, Finland
[2] Luleå Tekniska Universitet, Sweden

Abstract

This work-in-progress paper explores the integration of human cognition and interaction models into industrial automation systems. We begin by examining how human cognitive patterns can be applied to the development of conscious agents within these systems. From an interaction standpoint, we analyse the dual role of humans as both users and workers within automated environments. The convergence of these perspectives enables the creation of intelligent, multi-agent systems where humans function as equal agents. Such systems are characterised by true flexibility, as each component can independently assess its capabilities and collaboratively plan actions to achieve both collective (external) and individual (internal) goals. We present two case studies to illustrate these concepts: The first case study examines a distributed vertical farm, where modules must coordinate their energy consumption, demonstrating steps towards cognitive reasoning in machines. The second involves the automation of a cruise ship's HVAC system, where agents (cabin units) negotiate a temperature setpoint based on human behaviour (user-focused scenario).

Keywords: multiagent systems, human-computer interaction, computational rationality, reconfigurable manufacturing systems.

DOI: 10.1201/9788743808862-16

16.1 Introduction

Flexibility and reconfigurability are essential for driving automation processes based on demand. These characteristics are applicable across a range of domains, including building automation (e.g., HVAC systems, lighting control), pharmaceutical production, smart factories, and other manufacturing systems.

Reconfigurability is also critical in situations such as changing energy consumption targets, emergency scenarios, equipment upgrades, or the introduction of new workers who need time to reach full capacity while production targets remain unchanged. Similarly, adapting to external factors, such as increased HVAC demands—ensures that system goals are met without compromising productivity, even when resources are limited.

The dynamic requirements in these contexts arise from both internal and external factors. The internal ones are related to business processes and goals, liveness, and safety properties of the systems concerning with respect to their momentary conditions. External, in turn, are often influenced by general economic and environmental conditions. These requirements can be introduced during runtime or can be pre-defined.

Internal and external requirements can conflict, even if initially they appear aligned. For example, a predefined internal goal of meeting the minimal production output might contradict a dynamic external demand to reduce energy consumption. One solution would be to set priorities among requirements, while another approach involves optimising the performance of the system to balance competing needs.

Pre-programming for such optimisation is possible, but as systems grow in complexity, maintaining them becomes increasingly effort-intensive. A more scalable solution involves creating multi-agent systems (MAS), where each agent “knows” its capabilities, can “perceive” and adapt to changing goals.

In this paper, we propose a MAS framework based on self-conscious intelligent agents motivated by human cognition and capable of perceiving humans as equals.

16.2 Related Research

The concept of multi-agent systems (MAS) is not new, nor is the idea of conscious machines (both can be found in research and sci-fi novels). MAS is a system that involves multiple intelligent agents that interact with each other and solve a set of problems. In this paper we consider the industrial application of MAS, meaning that our agents are control programs for processes and devices in an industrial automation setting. Thus, for example, [1] discusses the use of agent technology in smart grid automation using the standards IEC 61499 and IEC 61850, [2] develops the idea of intelligent mechatronic components, and [3] uses an agent-based approach to model the dynamics of liquid goods logistics and energy efficient sensors usage.

While we refer the reader to [4] for the introduction of MAS, the works on the second concept we discuss here in more detail.

One way to bring consciousness to machines is to implement a belief-desire- intention (BDI) model [5]. In the BDI model, intentions are primarily perceived as elements of partial plans of action (although it is argued that BDI models do not include a built-in capacity for "lookahead" type of planning [6]). It reflects how humans reason and form intentions based on their goals and the information they have. In this model, belief represents the agent's perception of the environment, desire refers to the goals the agent wants to achieve (which can be organised hierarchically, although this is not always necessary), and intention is the agent's commitment to a specific course of action based on a plan.

The BDI model has its limitations, and a series of works tackles them. For example,

[7] complements the reasoning of BDI agents about time with reasoning in time, making the model suitable for dynamic environments. Then, the BDI model does not describe the communication between agents within the context of MAS, which is partially dealt with in [8] where LORA (the logic of Rational Agents) allows reasoning about interaction and incorporates action logic. An important extension of the BDI model is [9]. The author integrates a decision-theoretic planning framework into BDI and proposes the concept of bounded rationality. In this approach, the agent's decision-making is based on utility maximisation while also considering cognitive and environmental constraints, making it a more realistic model of rational behaviour under limitations.

Computational rationality [10] is based on the core ideas of bounded rationality. In addition to the mentioned constraints (cognitive, environmental, etc.), it considers the computational cost of the decision-making process. A solution to a computation rationality problem is optimal with respect to the trade-offs between the quality of decisions and the cost of computation. In [10], the authors highlight that in many cognitive science frameworks, decision-making is often modelled with two distinct levels, i.e., the higher rational level which defines goals and strategies, and the lower mechanistic level which focuses on the implementation or on how the agent uses its cognitive mechanisms (e.g., perception, memory) to approximate or achieve these goals in practice. Computational rationality, in turn, treats information-processing limitations (finite memory, time, or computational resources) as integral parts of the model of rational behaviour. Unlike traditional models of rationality, which assume agents have unlimited cognitive capacity, computational rationality integrates computational and information-processing constraints directly into the definition of rational behaviour.

The framework offers a structured way to explore how behaviours arise from cognitive mechanisms that are tailored not just to the environment, but also to the cognitive structure of the mind. Within this system, optimal behaviour is understood as the result of executing the best possible program suited to a given environment. This behaviour represents the machine's upper limit of performance under the given conditions.

The primary focus of [10] is how individual agents make decisions, so, although in the context of MAS, other agents can be viewed as the environment of individual agents they perceive, there is no specific focus on the interaction between agents, including scenarios where a human is perceived as an agent. In [12], the authors, in turn, lay the concept of computational rationality as a foundation for a theory of interaction within the domain of human-computer interaction (HCI).

While modelling the interaction between a computer and a human (e.g., through a graphical user interface or other means), the modeller must guess human reactions and encode them into a set of rules. This is a nontrivial task as users often deviate from the standard behaviour, which requires machines to be flexible in their replies.

In the industrial automation world, the problem is especially acute as the mental states of both human workers and human users are subject to change. Depending on the length of the collaborative production scenario, it can further alter and require a rapid, safe, and performance-optimal response from the machine. While [12] does not tackle industrial automation scenarios per se, it argues that computationally rational agents can successfully interact with humans.

The core idea is that interactive behaviour results from a control policy that is optimally adapted to users' preferences and constraints. These constraints arise both from the internal environment (cognitive limits) and the external environment (the interaction context). Interaction, therefore, is seen as the rational outcome of these bounded choices. By utilising approximate optimization techniques like reinforcement learning (RL), hypotheses about user goals and processing limits can be generated, allowing for adaptable strategies. The theory also allows the machines to generate explanations of human actions, i.e., answer "why?" and "what if?" questions, which allows them to be more adaptive. The need for adaptiveness is also motivated by the fact that individuals often distribute cognitive processing across their environment to optimise the use of their mental and cognitive resources, this is known as the adaptive distribution of cognition [13].

The works discussed provide a foundation for exploring computational rationality as a framework for agent interaction. Notably, we do not differentiate between human and machine agents in our approach. This leads us to our central question: "If we develop intelligent agents that approximate human cognition, can we integrate humans into the system as equal agents?" Our research objective is to create a multi-agent system (MAS) interaction framework that allows agents to be dynamically added or removed while pursuing both individual and collective system goals. This framework should have adaptable communication strategies to prevent using inefficient methods of problem-solving (e.g., becoming trapped in local optima [13]) when evidence suggests a better approach exists.

The central point in the agent architecture we envision is a self-awareness layer (or if we follow terminology from [12], the internal environment). Suppose we aim to avoid hard-coding production recipes or human-machine interaction. In that case, the control program should be able to "reflect" on itself and "decide" how its capabilities and limitations fit into the current set of system goals. For instance, consider an agent which is a program that controls a robotic arm. It can make the arm move and perform grip physically, but if it lacks awareness of these capabilities and is asked to retrieve a book from a shelf, the agent might respond that it cannot perform the task. One might argue that a self-awareness layer can be easily generated since an agent is a control program with the code often known.

Surely, if the agent is a white box and the control program is deterministic, one can very well predict its future behaviour knowing, for example, its execution traces. However, this luck is rare, and agents can be represented by a neural network or other machine learning models.

The next point is that to achieve the full coverage of the environment relevant to the agent, it needs to learn not only its own capabilities but also how they fit into the whole system and, hence, what other agents do. Thus,

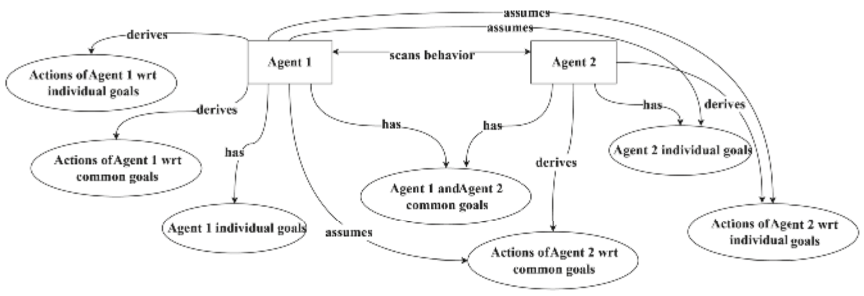

assumes
assumes
derives
Agent 1
scans behavior
Agent 2
Actions of Agent 1 wrt individual goals
derives
has
has
has
derives
Actions of Agent 1 wrt common goals
has
Agent 2 individual goals
derives
Agent 1 and Agent 2 common goals
Agent 1 individual goals
assumes
Actions of Agent 2 wrt individual goals
Actions of Agent 2 wrt common goals

Figure 16.1 Ontology showing the interaction of two agents. Assume-Derive phases are shown for Agent 1. ⏎

they need to observe the behaviour of other agents, assume what they could do with respect to the current goals, and derive their behaviour based on this knowledge and their goals. The assume-derive model is shown in Figure 16.1.

16.3 Interaction Framework

In this section, we outline the framework within which we can achieve the integration of human cognition principles into the MAS system in a way that a human is treated as an equal agent. It consists of four layers: automation, self- awareness, high-level agent communication, and the highest level where the user specifies the system goals. Each layer at any time must ensure safe operation and pass formal checks when necessary. For each of the layers, in addition to their description, we also name the major problems to be solved.

16.3.1 Layer 1: Automation

We work with industrial automation systems; thus, our first layer is the layer of pure automation control programs. We can operate out of the idea that our low-level control programs are smart and can rewrite their sequence of actions, and such work is being carried out. However, at this stage, we assume that they operate with a set of atomic operations that can be activated in any order by a scheduler from the higher level (e.g., using OPC UA). An atomic operation has a common definition and different specifics depending on the implementation of the automation system. Commonly, an atomic operation is a single action performed by a machine that starts in the safe state of the system and leaves the system in a safe state ready for next allowed actions.

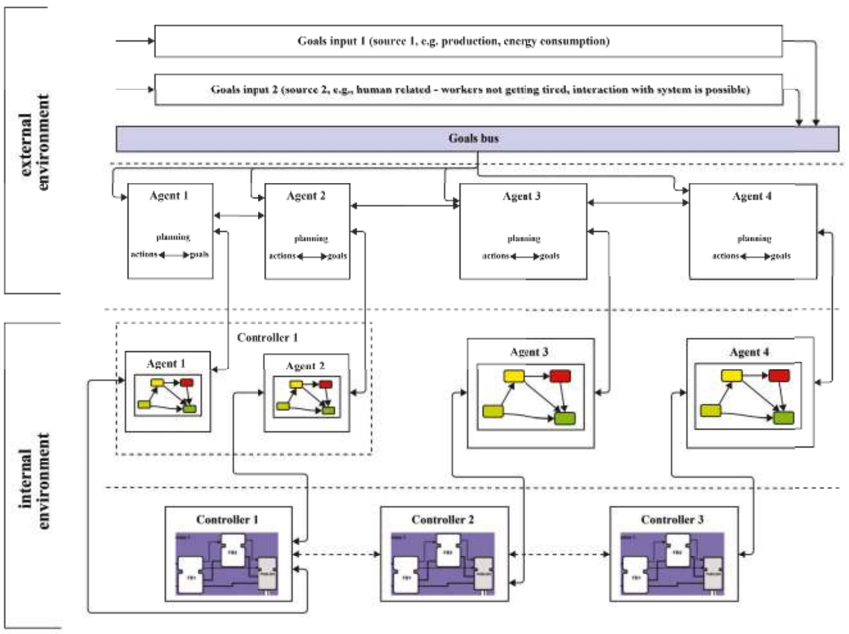

external environment
internal environment
Goals input 1 (source 1, e.g. production, energy consumption)
Goals input 2 (source 2, e.g., human related - workers not getting tired, interaction with system is possible)
Goals bus
Agent 1
Agent 2
Agent 3
Agent 4
planning
actions
goals
Controller 1
Controller 2
Controller 3

Figure 16.2 Interaction framework. ⏎

Consider having a drilling machine with a rotating table. Atomic operation "perform drilling" in addition to lowering the drill machine and turning it on for a defined amount of time, will also include checking if there is a workpiece underneath it. The same suffices with any other process constraints; for example, if there is a constraint on the maximum water temperature in the tank, the operation "heat" will not turn on the heater if the temperature has reached its maximum level.

An automation system can be represented as PLC control code with a human-machine interface (HMI) and in this case, we must interface the automation program directly, e.g., by sending data and events to function blocks, in case it is implemented in IEC 61499. Another option is to send the sequences of actions to the manufacturing execution system (MES), which has the set of possible actions defined. Thus, this level directly executes the commands from the upper layer and reports on the results of the operations performed back to the top. The PLC code can also be organised in a way that supports the MAS paradigm on this level to ease the debugging process.

16.3.2 Layer 2: Self-awareness

Self-awareness is one of the core layers of the framework, where the concept of an agent is introduced. As highlighted in Figure 16.2, the automation layer only implements control and equipment-specific safety constraints. The self-awareness layer, in turn, defines which automation operations can be grouped into a wholesome entity together with environmental observations specific to the operations. The grouping can be obvious, e.g., a conveyor belt or a robot arm can be self-sufficient agents. On the other hand, if the robot arm is installed on AGV, a composite agent can represent the whole assembly. Another way to define agents is by following the control loops of the system. Thus, having a vertical farming module with lighting and watering systems, the lighting agent would be capable of adjusting the light according to the lighting schedule and, for example, the PAR (Photosynthetically Active Radiation) sensor. The watering agent then is responsible for the regulation of the water flow depending on the watering schedule and a water flow sensor that detects clogs in the pipes.

The self-awareness layer can be developed through RL methods. For example, using data from factory processes, an offline RL method can learn the system's dynamics and generate an optimal control policy based on past experiences. In this approach, agents are trained to become aware of their capacities through iterative learning and feedback [14].

If a digital twin or a simulation model of the process is available, such development can be supported by a closed-loop system, comprising two main components: a modelling toolbox and a controlling toolbox. In [15], the modelling toolbox allows the creation of a plant model that simulates various system components or the entire system, which, in our case, would transform into a simulation of the controller actions and environmental response. Meanwhile, the controlling toolbox, originally, is a collection of autonomous agents that interact with the plant model using reinforcement learning algorithms, which would be replaced by the "consciousness" of the agents being trained.

The main function of this layer is to provide specific to the request available abilities and skills of the agent to the upper layer with the data needed for scheduling and planning and to translate these results into a sequence of atomic operations. The definition of appropriate reward functions for training the self-awareness layer to choose suitable capabilities is then necessary to keep the agents from "overthinking".

One interesting challenge in self-awareness is the possibility of discovering skills that were not demonstrated in the self-learning process. Assume, we have a conveyor belt, which is controlled by the motors and set to move workpieces with some fixed speed. Can the speed be changed to achieve production or energy consumption system goals? Whether the agent can discover new skills safely during the runtime is one of the research questions related to this layer.

Another challenge, stemming from the fact that the self-awareness of our agents is remote from their physical abilities, is how the self-awareness layer can adjust to the change of equipment executing approximately the same skills and actions. The advantage of our framework is that the upper layers do not have to be aware of such changes, while the self-awareness layer will have to perform self-discovery.

16.3.3 Layer 3: High-level communication and coordination

In this layer, agents communicate and coordinate the completion of tasks to achieve the goals passed down from the upper layer, thus we outline the following main functions of the level:

Goal selection. On the top level, the goals can be determined for the whole system (e.g., total energy consumption must be X) or for the subsystems (e.g., the human in the packaging unit should be protected from burning out). Agents must be able to select which goals are theirs to complete and which are for the other agents. Goals can also be adopted from other agents when they determine that assistance is needed.

Agents' collaboration. Agents should be able to group (or "team up") to collaboratively achieve the goals if needed. For example, for a human not to get a burn-out in the packaging unit, probably, a conveyor belt should work slower after lunch and an AGV should pick the packed products faster to keep the workspace of the human clean. For this, agents (1) must assume the goals of other agents, as precisely as they can, (2) assume their common and individual goals, and (3) derive their action based on the requirement inferred using this information. We call these two phases the "Assume" phase and the "Derive" phase (Figure 16.1). Reasoning can help with the Assume phase, where an agent can analyse the trace of decisions of other agents and find their motivation (for example, in [16]). This will allow agents to plan their actions in the context of plans of other agents.

Dynamic optimal policy determination. Some agents might have several goals to satisfy, which, in turn, can be contradictory and belong to different groups of agents. An agent must be able to find an optimal course of action. A crucial part of this decision-making process is evaluating utility [12]. The utility of an action is not limited to the immediate reward it brings but also depends on the future rewards that can be expected when the same policy is followed over time. By considering both short-term and long-term outcomes, agents can make decisions that optimise their overall performance in achieving individual and system-wide objectives.

External environment update handling. Agents can be sprouted and destroyed dynamically, depending on the momentary requirements, which they should be able to communicate and, thus, perceive when they are notified about the same events from the other agents. The reconfiguration in this case must happen automatically with as little downtime or human intervention as possible.

Communication with humans. Agents that interact with humans should have the mechanisms to assert the same aspects of the human internal environment as they do when communicating with nonhuman agents.

In the implementation of the framework for this layer, we also consider a time aspect of the decision-making. If finding an optimal action plan takes a significant number of resources (time, computational power, etc.), the heuristics are to be used. Also, here, we solve the problem of the cost of changing the plan if a new better option is found. For instance, if midway through the plan execution, the resources are already spent, this layer estimates whether replanning is beneficial.

16.3.4 Layer 4: System goals

Mainly the goals sprout from the requirements for the system, about which we talked in Section I. Different formulation languages are possible, for example, various temporal logics, diagrams, or natural language, which is becoming more possible with the development of LLMs. This layer is also the place for enlarging the vocabulary of the system, for example, if the goal is "to keep power consumption at 80% from maximum consumption during the nearest month", the system should at least know what power consumption is. LLMs and users play a key role in vocabulary and semantics enrichment.

The requirements the system should satisfy may have a time interval, for example, different production scenarios of the island-based factory might follow different production recipes, however, the general liveness and safety requirements for the production may remain. One important note is that we do not differentiate here goals for machines and goals for humans within the production. Human workers or users participate in goal distribution on the same level as nonhuman agents do. The difference shows only in the way the goals are communicated to a human as special interfaces are required for the purpose.

16.4 Case Study

As a leitmotif for the framework trials, we chose the topic of energy consumption, since, in large productions, being able to reduce it even for a small fraction of the total leads to a significant impact on sustainability and production costs. Our case studies explore different parts of the framework implementation starting from communication between the control loops of the assembly, where each of them can straightaway decide on energy consumption to indirect interaction with humans and agents that influence the energy consumption of the main consumers but not decide on it straight away.

16.4.1 Vertical farming module

Our first case study is on the optimisation of the energy consumption of a vertical farming module. Figure 16.3 shows the prototype module implemented at the Aalto Factory of the Future and its overall architecture. Our automation code runs on two M251 PLCs to which both digital and analogue sensors and actuators connect. This layer connects to the business logic via OPC UA. The

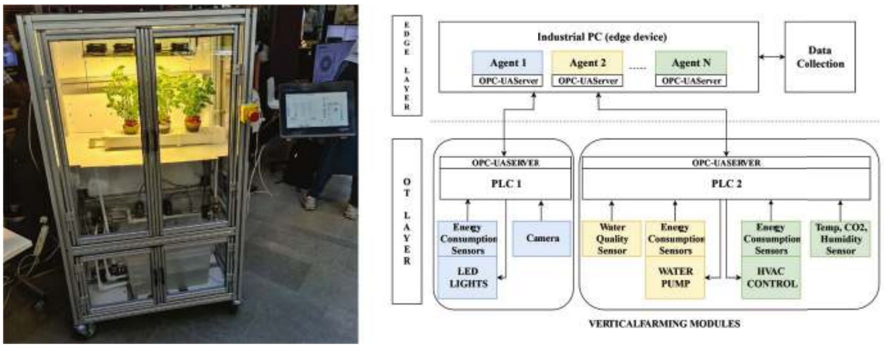


Figure 16.3 Vertical farming module and its controller architecture. ⏎

business logic layer is placed on the industrial PC, while it is also possible to develop it in the cloud.

Our high-level logic employs the MAS concept. Each agent here is responsible for calculating the consumption of a single control loop, where a control loop is composed of an actuator and the environmental response on the action of the actuator, for example, pump–water level. The whole module itself can also be an agent in case there is more than one. In this case, it becomes a composed agent, incorporating a set of agents—control loops. Agents communicate with each other to converge on the best individual power consumption values, given the total target power consumption of the system.

In this example, we explicitly specified the self-conscious layer by inputting the characteristics of the actuators controlled during the experiments, namely LED lights of the vertical farming module. We have also experimented with setting up communication between different agents through distributed optimisation methods (for details, see [16]).

16.4.2 HVAC system of a cruise ship

Another case study we have in progress is the energy consumption of a cruise ship. Here, several cabins connect to one Air Handling Unit (AHU) and several AHUs connect to a diesel engine. In HVAC, the parts that consume the most energy are the fans that drive the air inside the main ducts and the chiller that cools it down to a set temperature. The only way we can influence the energy consumption decision is by adjusting the temperature set points in the cabins. Thus, lowering the cabin temperature would mean that the corresponding valve for fresh air intake should open more, which decreases the pressure in the main air duct, making the main fan increase power consumption to stabilise the pressure, which also makes the chillers cool down a larger air volume.

Since the solution should be flexible (extendable to various actuator types) and scalable (more cabins can be connected to air ducts/more AHUs, diesel engines in the system), we define each cabin, each AHU, and each diesel as an agent. This relieves us from the need to explicitly encode the physics of the process (unlike in the first case study) and AHU "learns" its dynamics of power consumption based on the traces of the simulation model using RL methods. One of the research questions here is how to dynamically generate not only the agents but also the correct way of communication between them.

Another angle of this study is incorporating users' patience into the system. The energy reserves are restricted on the cruise ship. Suppose that a lot of people set the desired temperature below some minimum. In that case, there is a chance that the diesel engines will not be able (within safety boundaries) to provide enough energy for the ship infrastructure and the HVAC system. This means that the temperature set point should be adjusted automatically; however, this works up to a point where people do not massively complain.

16.5 Discussion and Conclusion

While extensive research has been conducted on isolated issues in MAS and HCI, little attention has been paid to the design of intelligent systems as a whole, especially, from the industrial automation perspective. This paper outlines the foundational problems and research questions aimed at shaping machine consciousness, enabling agents to understand one another as well as communicate effectively with humans.

We hypothesise that the implementation of such an architecture leads to emergent behaviours akin to needs and motivations for the system itself. As individual agents become more aware and coordinated, the factory's operations could form a collective consciousness, transforming the production process into something that responds naturally to the environment rather than following pre-set instructions. This represents the basis of a truly reconfigurable and flexible factory—one where manufacturing as a service becomes an inherent response to the factory's needs, rather than an artificially imposed command structure.

Future work will involve a deeper analysis of the solutions proposed for each layer of the system, with special emphasis on the communication aspects, both within the machine agents and between agents and human workers, which involves exploring how intelligent agents can effectively communicate and collaborate with humans.

Furthermore, experiments will be conducted in environments with static and dynamic actuators, including mobile workstations, AGVs, and reconfigurable factories. This framework will also have applications in energy communities, where negotiation and flexibility with human input is vital, especially for systems without plug-and-play capabilities, like HVAC.

Acknowledgements

This work was supported, in part, by projects I-SWARM X (p/n 411061), Multi- Energy VPP (p/n 13348415), and the project funded by the Academy of Finland, CAFE (p/n 13363691).

References

[1] G. Zhabelova, V. Vyatkin, and V. N. Dubinin, "Toward Industrially Usable Agent Technology for Smart Grid Automation," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 4, pp. 2629–2641, Apr. 2015,

[2] A. Kalachev, G. Zhabelova, V. Vyatkin, D. Jarvis and C. Pang, "Intelligent Mechatronic System with Decentralised Control and Multi-Agent Planning," *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*, Washington, DC, USA, 2018, pp. 3126-3133,

[3] J. Kaiser, M. P. Hernández, V. Kaupe, P. Kurrek, and D. McFarlane, "An agent-based approach for energy-efficient sensor networks in logistics," *Engineering Applications of Artificial Intelligence*, vol. 127, p. 107198, Jan. 2024,

[4] M. J. Wooldridge, "An introduction to multiagent systems". Chichester: John Wiley, 2012,

[5] M. E. Bratman, D. J. Israel, and M. E. Pollack, "Plans and resource-bounded practical reasoning," *Computational Intelligence*, vol. 4, no. 3, pp. 349– 355, Sep. 1988,

[6] S. Sardiña, L. de Silva, and L. Padgham, "Hierarchical planning in BDI agent programming languages," *5th international joint conference on Autonomous agents and multiagent systems (AAMAS '06)*, Association for Computing Machinery, New York, NY, USA, pp. 1001–1008, May 2006,

[7] F. Alzetta, P. Giorgini, M. Marinoni, and D. Calvaresi, "RT-BDI: A Real-Time BDI Model," *Advances in Practical Applications of Agents, Multi-Agent Systems, and Trustworthiness*, The PAAMS Collection, pp. 16–29, 2020,

[8] M. Wooldridge, "Reasoning about Rational Agents." *MIT Press*, 2003,

[9] G. Boella, "Decision theoretic planning and the bounded rationality of BDI agents," *In Proceedings of GTDT 2002 Workshop*, Technical report WS- 02, vol. 6, pp. 1-10, 2002,

[10] R. L. Lewis, A. Howes, and S. Singh, "Computational Rationality: Linking Mechanism and Behavior Through Bounded Utility Maximization," *Topics in Cognitive Science*, vol. 6, no. 2, pp. 279–311, Mar. 2014,

[11] A. Oulasvirta, J. Jokinen, and A. Howes, "Computational Rationality as a Theory of Interaction," *CHI Conference on Human Factors in Computing Systems (CHI '22),* Association for Computing Machinery, New York, NY, USA, Article 359, 1–14, 2022,

[12] S. J. Payne, A. Howes, and W. R. Reader, "Adaptively distributing cognition: A decision-making perspective on human - computer interaction," *Behaviour & Information Technology*, vol. 20, no. 5, pp. 339– 346, Jan. 2001,

[13] J. Deng, S. Sierla, J. Sun, and V. Vyatkin, "Offline reinforcement learning for industrial process control: A case study from steel industry," *Information Sciences*, vol. 632, pp. 221–231, Jun. 2023,

[14] Y. Berezovskaya, C.-W. Yang, and V. Vyatkin, "Reinforcement learning approach to implementation of individual controllers in data centre control system," *IEEE 20th International Conference on Industrial Informatics (INDIN)*, pp. 41–46, Jul. 2022,

[15] P. Ovsiannikova, I. Buzhinsky, A. Pakonen, and V. Vyatkin, "Oeritte: User- Friendly Counterexample Explanation for Model Checking," *IEEE Access*, vol. 9, pp. 61383–61397, 2021,

[16] K. Zhukovskii et al., "Energy Consumption Optimisation for Horticultural Facilities", *IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2024.

17

Neuromorphic IoT Architecture for Efficient Water Management

Mugdim Bublin[1], Heimo Hirner[1], Antoine-Martin Lanners[1], and Radu Grosu[2]

[1]University of Applied Science FH Campus Wien, Austria
[2]Technische Universität Wien, Austria

Abstract

The exponential growth of IoT networks necessitates a paradigm shift towards architectures that offer high flexibility and learning capabilities while maintaining low energy consumption, minimal communication overhead, and low latency. Traditional IoT systems, particularly when integrated with machine learning approaches, often suffer from high communication overhead and significant energy consumption.

This work addresses these challenges by proposing a neuromorphic architecture inspired by biological systems. To illustrate the practical application of our proposed architecture, we present a case study focusing on water management in the Carinthian community of Neuhaus. Preliminary results regarding water consumption prediction and anomaly detection in this community are presented. We also introduce a novel neuromorphic IoT architecture that integrates biological principles into the design of IoT systems. This architecture is specifically tailored for edge computing scenarios, where low power and high efficiency are crucial. Our approach leverages the inherent advantages of neuromorphic computing, such as asynchronous processing and event-driven communication, to create an IoT framework that is both energy-efficient and responsive. Moreover, we show that not only is the architecture neuromorphically inspired, but it can also be partially realized, especially at the edge, by neuromorphic hardware. This case study demonstrates how the neuromorphic IoT architecture can be deployed in a real-world scenario, highlighting its benefits in terms of

DOI: 10.1201/9788743808862-17

energy savings, reduced communication overhead, and improved system responsiveness.

Keywords: IoT, edge computing, neuromorphic computing, deep learning, machine learning.

17.1 Introduction and Background

The exponential growth of the Internet of Things (IoT) networks necessitates a paradigm shift in how these systems are designed and deployed. Traditional IoT systems, especially those integrated with machine learning techniques, often face significant challenges, including high communication overhead, increased energy consumption, and latency issues [1]. As IoT devices become more prevalent, particularly in edge computing scenarios, the need for architectures that balance flexibility, learning capability, and energy efficiency becomes critical. These issues are exacerbated when machine learning models are employed, as they typically require substantial computational resources. The challenge lies in creating an IoT architecture that can efficiently process data at the edge, with minimal energy consumption and latency, while still providing the flexibility and learning capabilities necessary for complex tasks such as anomaly detection and prediction.

This paper addresses these challenges by proposing a neuromorphic IoT architecture inspired by biological systems, specifically tailored for scenarios where low power consumption and high efficiency are essential. Neuromorphic computing is an innovative approach in computer engineering that designs computational systems inspired by the architecture and functionality of the human brain and nervous system [2–4]. This brain-inspired computing method offers significant advantages, including [5], [6]:

- **Energy Efficiency:** Traditional deep learning models may require up to 20 MW of power, while the human brain operates on just about 20 W, highlighting the potential for substantial energy savings.
- **Latency:** Neuromorphic systems excel in parallel processing, allowing for faster computations and reduced latency.
- **Safety & Security:** These systems enhance reliability by using redundant and analog components, mimicking the robustness of biological neural networks.
- **Reduced Costs and Waste:** By leveraging materials that mimic biological processes, neuromorphic computing can lower production costs and minimize environmental impact.

These benefits point towards a new computing paradigm that could revolutionize how we approach computational tasks. The primary objective of this study is to propose a neuromorphic IoT architecture that integrates principles from biological systems into the design of IoT systems. This architecture is intended to offer significant improvements in energy efficiency, communication overhead, and system responsiveness. Additionally, we aim to validate the effectiveness of this architecture through a case study on water management in the small city Neuhaus, Austria, demonstrating its practical application and potential benefits.

17.2 Neuromorphic IoT Architecture

17.2.1 Design principles

The proposed neuromorphic IoT architecture is inspired by the efficiency of biological systems, particularly human nervous system, which have evolved to process information with minimal energy consumption and high responsiveness [6]. The key features of this architecture include: distributed control and learning, prediction instead of commands and reservoir computing.

17.2.2 Hierarchical distributed control and learning

Control and learning in an IoT network are distributed across various nodes, allowing the system to leverage locally available information for decision-making where it is most effective. This architecture is tailored for edge computing environments, prioritizing low power consumption and high efficiency. By processing data at the edge, the system can make real-time decisions without the need for constant communication with centralized servers.

In practice, different machine learning models are deployed across various layers of the IoT architecture: the device, or edge layer, fog layer, and cloud layer. At the edge layer, rapid decisions are made, such as automatically shutting off a water pipe in case of damage. The fog layer allows for more sophisticated decisions by integrating data from nearby devices to improve local water management. At the cloud layer, data from IoT devices and the internet is aggregated to manage overall water resources and address disaster scenarios.

We propose that higher architectural levels use predictions instead of commands, similar to the human visual [7] and motor systems [8]. In fog and cloud, models of water predictions on regional and global scales are built.

The model predictions are sent to lower levels, while error corrections are sent from lower to upper layers. Unlike federated learning [9], which relies on a single global model for all devices, this approach utilizes local machine learning models to make decisions based on locally available information. This strategy reduces communication overhead, latency, and energy consumption.

Neuromorphic Reservoir Computing is particularly well-suited for edge implementation due to its low computational overhead, which results in reduced energy consumption, and its ability to harness underlying physical processes for computation. Neuromorphic reservoir computing is a computational framework inspired by the brain's neural networks, particularly suited for processing time-series data and performing complex pattern recognition tasks [10–12]. This approach is based on the concept of a "reservoir," which is a recurrent neural network with fixed, randomly initialized connections. The key idea is that the reservoir can transform temporal input data into a higher-dimensional space, where the information is easier to analyse and predict. The advantage of this method lies in its simplicity: only the output layer is trained, while the reservoir itself remains unchanged. This reduces the computational burden associated with training deep neural networks.

In Figure 17.1 the analogy between human nervous system and the proposed IoT architecture is depicted.

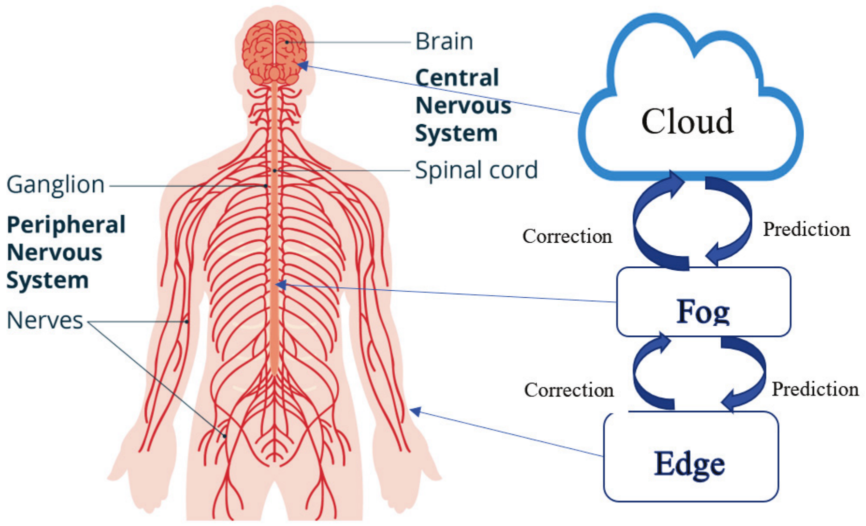


Figure 17.1 Analogies between human nervous system and the proposed IoT architecture. ⏎

Figure 17.2 depicts the black-box view of the overall system. The IoT network receives inputs from the environment (such as sensor signals like water consumption measurements, weather data, and other internet-sourced data) and executes commands on the environment (e.g., opening/closing pipes, issuing alarms, etc.).

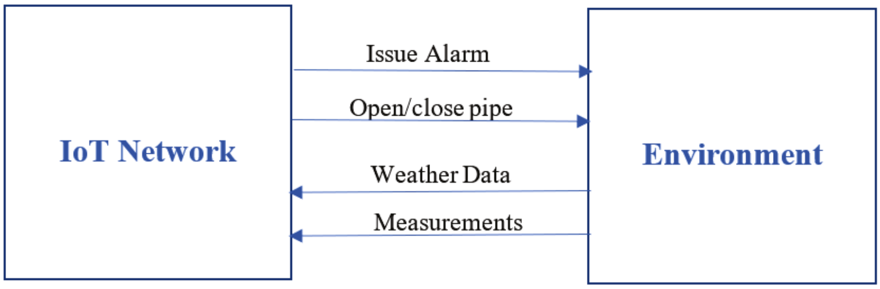


Figure 17.2 Black-Box view of the IoT network and the environment. ⏎

Using hierarchical distributed control and learning with prediction/correction feedback loops, we can minimize communication overhead and energy consumption. Information is primarily processed where it is available; only predictions and corrections are communicated, not commands. A mathematical framework for this approach is provided by Friston's active inference and free energy principle [13], [14].

17.3 Free Energy Principle

Friston's free energy principle [13], [14] is a theoretical framework from neuroscience, which posits that the brain minimizes a quantity called free energy to maintain a stable internal state and make sense of the world. This principle is derived from thermodynamics and statistical mechanics, and it explains how biological systems (like the brain) resist disorder (entropy) by maintaining an internal model of the environment. In our case, each level of the IoT hierarchy has its own world model that is updated based on inputs from the next lower level. Active inference has also been applied to IoT in [15].

The free energy principle can also be broken down into terms involving surprise and approximation errors.

$$F = DKL(q(s) \quad p(s \mid o)) - log\, p(o) \tag{17.1}$$

Where:

$DKL(q(s) \ \ p(s|o))$ is the Kullback-Leibler (KL) divergence between the recognition density $q(s)$ and the posterior distribution $p(s|o)$, which measures how different the brain's internal model is from the real posterior probability of hidden states s given observations o. The hidden state, s, is a vector representing the state of the environment in each IoT layer. The observations o are vectors of measurements (over time) from the environment and signals from the lower layers.

$log\ p(o)$ is the log-evidence or surprise associated with the sensory input o. The brain aims to minimize surprise (unexpected sensory states).

To reduce free energy, the brain does two things:

Perception: Adjusting its internal model (recognition density $q(s)$) to match sensory input better. This helps reduce the Kullback-Leibler divergence (KLD), which improves the accuracy of its beliefs about hidden states.

Action: Taking actions to influence the environment in a way that makes sensory input more predictable, thus reducing surprise $log\ p(o)$.

In our setup, each hierarchy level of an IoT network creates its own internal model of the environment, updates the model according to sensor inputs from lower layers, and performs actions on the lower layers to reduce discrepancies between the model and the environment.

The free energy principle enables **more autonomy** for each layer, as it places fewer constraints on the internal model of each layer compared to, for example, reinforcement learning (RL), which assumes that agents maximize expected rewards. Furthermore, it also requires **less communication overhead**, since information is signaled to other layers only when the differences between predictions and actual measurements exceed a certain threshold.

17.4 Asynchronous Processing and Event-driven Communication

Unlike traditional computing systems, which rely on a global clock, the proposed architecture processes information asynchronously. This allows for more efficient use of resources, as computation only occurs when necessary.

In biological systems, communication between neurons occurs only when a certain threshold is reached, triggering a spike of activity. This principle is applied in the proposed architecture to reduce unnecessary communication, thereby lowering energy consumption and latency.

17.5 The Role of Thresholds in Hierarchical IoT Model

In hierarchical models, each layer generates predictions based on the lower layer's output, with higher layers handling more abstract representations. Thresholds, in this context, can refer to parameters that decide:

- **Uncertainty bounds:** Determining when prediction errors (discrepancies between predicted and observed states) should trigger adjustments in the model or action.
- **Signal passing thresholds:** Deciding when sufficient confidence is achieved in a layer to propagate the information upward or downward.

In active inference framework agents (IoT layers) continuously minimize their **variational free energy** (a measure of surprise or uncertainty about sensory inputs) by updating beliefs or performing actions. Key to this process is minimizing **prediction errors** through updating the generative model (beliefs about the world) or by acting to bring sensory inputs in line with predictions.

- **Perception:** Update internal states to minimize prediction error.
- **Action:** Perform actions to reduce the discrepancy between expected and actual sensory inputs.

17.5.1 Setting adaptive thresholds using prediction errors

In active inference, **prediction error** (the difference between the predicted input and actual sensory input) drives updates to beliefs or actions. To adaptively set thresholds at different hierarchical layers, you can allow thresholds to be modulated by the **magnitude of prediction error** and the **precision (inverse uncertainty)** associated with each layer's predictions.

1. **Prediction Error Calculation:** At each layer L$_i$, prediction error is computed as:
$$\epsilon_i = input_i - prediction_i \tag{17.2}$$
where ϵ_i is the prediction error at layer *i*, and $input_i$ is the input to the layer (could be sensory data for the bottom layer or output from the previous layer).
2. **Precision-Weighted Prediction Error:** To adaptively set thresholds, weight the prediction error based on the layer's **precision** Πi which represents the inverse of uncertainty:
$$\widetilde{\epsilon_i} = \Pi_i \cdot \epsilon_i \tag{17.3}$$

Here, ϵ_i is the precision-weighted prediction error.

3. **Adaptive Threshold Adjustment:** Let the threshold at layer L_i denoted τ_i be adjusted based on the running average or variance of prediction errors at that layer. For example, using an exponentially weighted moving average (EWMA):
$$\tau_i(t+1) = \alpha \cdot \tau_i(t) + (1-\alpha) \cdot |\epsilon_i| \tag{17.4}$$
where α is a smoothing factor, controlling how quickly the threshold adapts to changes in prediction error magnitude.
4. **Hierarchical Precision Tuning:** Active inference frameworks often adjust precision at each layer based on uncertainty in the environment. If a lower-level layer exhibits high variability (uncertainty), precision at higher levels may be reduced (lower confidence in predictions), increasing tolerance for prediction errors.
We can compute adaptive precision Π_i at each layer based on past errors:
$$\Pi_i = \frac{1}{\sigma_i^2 + \beta} \tag{17.5}$$
where σ_i^2 is the variance of prediction errors at layer L_i, and β is a small constant to avoid division by zero.
5. **Threshold Propagation Through Layers:** In a hierarchical model, the adaptive thresholds τ_i can also be influenced by errors at neighboring layers. For instance:
$$\tau_i = f(\epsilon_{i-1}, \epsilon_{i+1}) \tag{17.6}$$
where τ_i is set adaptively based on errors in both the layer above and the layer below, helping each layer balance local and global model adjustments.

17.5.2 Incorporating actions into threshold setting

Active inference involves both **belief updates** and **action selection**. Thresholds can be adapted based on both the **perceptual** prediction errors (used to update internal beliefs) and the **action** outcomes (used to reduce discrepancy between predicted and actual sensory inputs).

For example:

- If actions reduce prediction error, thresholds may decrease, indicating higher precision in predictions.
- If actions increase prediction error, thresholds may increase to allow more flexibility in updating the generative model.

17.5.3 Optimizing thresholds using free energy minimization

In active inference, the agent seeks to minimize **free energy**, which combines **prediction error** and **model uncertainty**. Thus, thresholds τ_i can be adaptively set by minimizing the total free energy at each hierarchical level.

The free energy at each layer i can be expressed as:

$$F_i = \frac{1}{2}\epsilon_i x^2 + H(\Pi_i) \tag{17.7}$$

where $H(\Pi_i)$ represents the entropy or uncertainty in the precision at that layer. Minimizing F_i can help set optimal thresholds at each layer by balancing prediction accuracy and uncertainty.

17.5.4 Threshold setting summary

To adaptively set thresholds in hierarchical layers using active inference:

1. **Monitor prediction errors** at each layer.
2. **Adjust thresholds** based on the precision-weighted errors.
3. **Tune precision** and thresholds across layers by minimizing free energy and propagating uncertainty estimates.
4. **Use feedback from action** to refine thresholds.

This approach aligns with the active inference principle of reducing prediction errors while accounting for uncertainty in a dynamic environment.

17.6 Implementation

To implement the proposed architecture, we developed a neuromorphic IoT framework that integrates neuromorphic devices for data processing. These devices are deployed on edge devices, which are responsible for collecting data, processing it locally, and making decisions in real-time. The framework is designed to be modular, allowing for easy integration with existing IoT systems and flexibility in terms of the types of sensors and devices used.

Auto Regressive Integrated Moving Average (ARIMA) is a classical model used for time-series forecasting, which combines autoregression, differencing, and moving average components. Both moving average with threshold comparison, as required in each layer of our IoT architecture, and ARIMA can be realized using neuromorphic computing. Moving average and threshold comparison can be implemented in following way [16], [17] (see Figure 17.3):

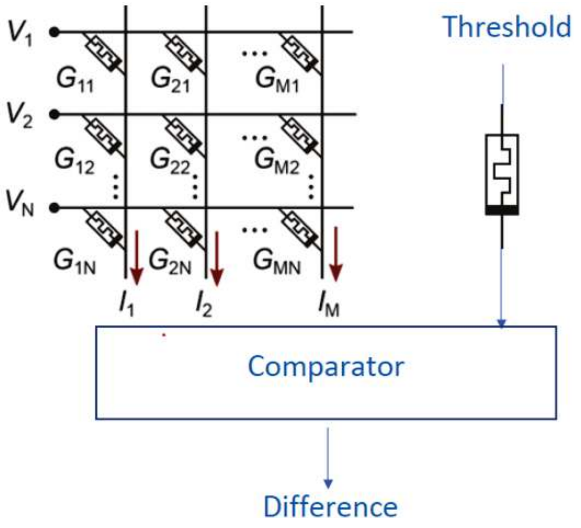


Figure 17.3 Memristor implementation of moving average and threshold comparison. ⏎

The implementation consists of the following components:

1. **Memristor Array for Moving Average:**
   - The inputs are weighted by the conductance of each memristor.
   - By applying voltages (input signals) across memristors with different conductances, a weighted sum of the inputs can be computed. Since the conductance (inverse of resistance) of the memristor changes based on the applied voltage, it represents the weight of each input in the moving average. The total output current represents the weighted sum (i.e., the moving average).
2. **Threshold Comparator:**
   - A reference memristor holds the threshold value.
   - The output current from the memristor array (moving average) is compared to the current through the threshold memristor.
3. **Output Decision:**
   The difference in the currents between the input memristor and the threshold memristor can be calculated using a current subtractor circuit or a differential amplifier. This will output a current (or voltage) proportional to the difference between the input signal and the threshold.

17.7 Case Study: Smart Village Water Management

17.7.1 Context and objectives

The municipality of Neuhaus in south-eastern Carinthia, Austria, on the Slovenian border with 1015 inhabitants (as of 1 January 2024, https://www.statistik.at) and an area of 36.34 kmš offers a unique opportunity to apply the proposed neuromorphic IoT architecture in a real-life scenario. Like other similarly structured micro-communities, Neuhaus is facing the typical problems of rural areas in Central Europe: declining population figures combined with shrinking financial and human resources and new challenges due to climate change. Water management is of crucial importance in this region, as efficient utilization of water resources is essential for both environmental sustainability and economic viability. In the last two years, Neuhaus was confronted with a period of drought in the summer of 2023 and enormous amounts of rain, flooding, and landslides in August 2024. Both have a negative impact on financial resources and the quality of life in the region.

In order to overcome these challenges and reduce the administrative expenses, the municipality decided back in 2020 to replace all 377 water meters in the municipality with smart meters. The water meters have to be replaced every 4 years anyway for calibration reasons, so the one-off additional costs were limited. A municipal LoRaWAN radio network was set up for data transmission. LoRaWAN fulfils the required criteria for long range in rural areas, low energy consumption of battery-operated smart meters, and low installation and operating costs. The first automatic meter reading took place in September 2021. In addition to the water meters, the reservoirs of the three separate water supply systems, in which the spring water is collected, were equipped with solar-powered level and flow meters. A total of around 560 LoRaWAN sensors are currently installed in the municipal area. In addition to the water meters, these include weather stations, road temperature sensors, snow depth gauges, and indoor CO_2 sensors, for. In July 2022, the research cooperation between the municipality of Neuhaus and the University of Applied Sciences FH Campus Wien was launched. The aim of this cooperation is to do research on IoT solutions for small municipalities. This gives us access to all measurement data of the LoRaWAN sensor network. Instead of simulated laboratory data, we can work with real data, coming from a lossy IoT network. The first project realized as part of this collaboration was a water management system for real-time water balances tailored to small communities with their limited resources. It has been in operation since July 2023 and is used by the municipality.

In the present case study, we extend this system to include predictions and alerts. The problems we are facing are typical for IoT sensor networks. We have limited edge devices (our battery-powered water meters) and an unreliable network with very limited data rates. Typically, the transmission of water meter data occurs once daily to save battery power. An immediate message should only be sent in the event of an exceptional incident, e.g. a burst pipe after the meter. So, the decision must be made locally at the edge device. And, since these devices are battery-powered, they are also limited in their processing power and memory. The objectives of this case study are to:

1. Predict water consumption patterns in the community.
2. Detect anomalies in water usage that may indicate leaks or other issues.
3. Demonstrate the effectiveness of the proposed architecture in reducing energy consumption, communication overhead, and latency.

17.7.2 Data collection and preprocessing

Data on water consumption were collected from various sensors deployed throughout the community. These data included hourly and daily water usage statistics, from the smart meters, as well as information on environmental

Table 17.1 Input and output information available at each IoT layer. ⏎

IoT Layer	Input from layer above	Other Inputs	Output to layer above	Time and space scale
Edge	Local water consumption threshold	Actual water consumption of each household	Difference between expected average local water consumption and the threshold	Seconds/Local (household)
Fog	Regional water consumption threshold	Regional water supply, Regional water pipe connection map	Difference between expected average regional water consumption and the threshold	Days/Regional
Cloud	-	Time, date and season, weather information from internet, water supply	Output to human: statistics, predictions and alarms over global water consumption	Months/Global

factors such as temperature and humidity, which may influence water consumption patterns. The data were preprocessed to remove noise and fill in missing values due to losses in the LoRaWAN network before being fed into the neuromorphic IoT framework. The following table shows the input and output information available at each IoT layer (see Table 17.1).

17.7.3 Prediction models and performance

To evaluate the effectiveness of the proposed architecture, we compared the performance of several machine learning models in predicting water consumption. The models used included a Multi-Layer Perceptron (MLP), Long Short-Term Memory (LSTM) network, Auto Regressive Integrated Moving Average (ARIMA) model, and Random Forest. The mean absolute percentage error (MAPE) was calculated for each model to assess their predictive accuracy.

$$MAPE = \frac{1}{n}\overset{n}{\underset{i=1}{}}\left|\frac{\overline{y}_i - y_i}{y_i}\right| \tag{17.8}$$

Where:
$\overline{y}_i$is predicted value for ith data point
y_i is actual value for ith data point
N is the number of observations

The results (see Table 17.2) indicate that the Random Forest model performed the best in terms of hourly prediction accuracy, with a MAPE of 26.05%. For daily predictions, the LSTM model achieved the lowest MAPE of 4.87%.

Overall, the ARIMA model achieves quite good performance in both hourly and daily water consumption prediction.

ARIMA (AutoRegressive Integrated Moving Average) is a classical model used for time-series forecasting, which combines autoregression, differencing, and moving average components.

Table 17.2 Hourly and daily prediction errors of different algorithms. ⏎

Algorithm	**Hourly Prediction MAPE [%]**	**Daily Prediction MAPE [%]**
MLP	41.10	5.05
LSTM	33.51	4.87
ARIMA	30.06	5.18
Random Forest	26.05	5.40

Reservoir computing can approximate ARIMA by leveraging its ability to model nonlinear relationships and memory of past inputs, which are essential components of ARIMA models. In particular, the recurrent nature of the reservoir allows it to capture the autoregressive and moving average aspects of the time series, while the nonlinear transformation within the reservoir can approximate the differencing and other complex relationships present in ARIMA models. The **equivalence of reservoir computing to nonlinear vector autoregression (NVAR)** is rooted in the way both models handle temporal data [16], [17]. NVAR is a statistical model that predicts future values of a time series based on past values, accounting for possible nonlinear relationships between variables. Reservoir computing, by transforming input sequences into a dynamic state within the reservoir, effectively performs a similar operation to NVAR. It creates a nonlinear mapping of past inputs, which can then be used to predict future values. This equivalence is particularly useful because it shows that reservoir computing can be viewed as a form of nonlinear autoregression, with the reservoir acting as the nonlinear transformation that enables the prediction of future states based on past data. This understanding opens up the possibility of using reservoir computing as a powerful tool for time-series prediction, where traditional methods like NVAR are used.

In summary, by exploiting the equivalence between reservoir computing and nonlinear vector autoregression, reservoir computing can be effectively used to approximate the behavior of ARIMA models, offering a powerful alternative for time-series forecasting that is particularly well-suited to handling nonlinear and complex patterns in the data. Neuromorphic reservoir computing is particularly interesting for IoT networks due to its low computational overhead and wide range of possibilities for physical implementations. In this way, not only is the IoT architecture neuromorphically inspired, but it can also be partially realized, especially at the edge, by neuromorphic hardware.

17.7.4 Anomaly detection

Anomaly detection is a critical aspect of water management, as it allows for the identification of unusual patterns in water usage that may indicate leaks or other issues. In this study, we applied both the mean + 3 sigma rule and an LSTM-based anomaly detection method to the water consumption data. The results showed that the mean + 3 sigma rule detected more anomalies than the LSTM model, suggesting that simple statistical methods may be more effective for this type of application in certain contexts.

17.8 Discussion

17.8.1 Energy efficiency and communication overhead

One of the primary advantages of the proposed neuromorphic IoT architecture is its energy efficiency. By processing data locally at the edge and using event-driven communication, the system significantly reduces the need for constant data transmission to centralized servers, thereby lowering energy consumption and communication overhead. This is particularly important in rural areas like Neuhaus, where energy resources may be limited.

17.8.2 System responsiveness and latency

The asynchronous processing and event-driven communication of the proposed architecture also contribute to improved system responsiveness. In real-time applications such as water management, where timely detection of anomalies is crucial, the ability to process data and make decisions quickly can prevent significant water loss and reduce the environmental impact.

17.8.3 Safety & security

Safety: For analog systems it is possible to use continuity properties when pondering system behavior in different points of their state space. If a system exhibits intended behavior in a state A and in a related state B, it can be argued that it will show intended behavior also when $C = \alpha A + (1 - \alpha)B$, where $0 < \alpha < 1$. So if the system is tested in states A and B, then it can be assumed that it will not change too much in the intermediate states C in between.

Security: It is more difficult to access and modify analog hardware like memristors or reservoirs than to perform a security attack over the internet.

17.8.4 Practical implications

The case study demonstrates that the proposed neuromorphic IoT architecture is not only theoretically sound but also practically viable. The deployment in Neuhaus and preliminary results show that the architecture can handle the complexities of a real-world environment while delivering tangible benefits in terms of energy savings, reduced communication overhead, and improved system responsiveness.

17.9 Conclusion

The exponential growth of IoT networks demands a new approach to system architecture, one that balances flexibility, learning capability, and energy efficiency. This paper has proposed a neuromorphic IoT architecture inspired by biological systems, designed to meet these challenges in edge computing scenarios. Through a case study on water management in the Carinthian community of Neuhaus, we have demonstrated the practical application and benefits of this architecture. The results show that the proposed architecture can deliver significant improvements in energy efficiency, communication overhead, and system responsiveness, making it a promising solution for the future of IoT networks.

17.10 Future Work

While the proposed architecture has shown promising results, further research is needed to optimize the integration of neuromorphic computing with existing IoT frameworks and to explore its application in other domains. Additionally, the development of more sophisticated anomaly detection methods, potentially integrating neuromorphic principles, could further enhance the system's capabilities. As IoT networks continue to evolve, the principles outlined in this paper will be crucial in guiding the development of next-generation systems that are both efficient and effective

Acknowledgements

The research reported in this paper has been partly funded by the European Union's Horizon 2020 research and innovation program within the framework of Chips Joint Undertaking (Grant No. 101112268) and by the City of Vienna Magistrat 23, FH-Call 29, project 29-22: 'Stiftungsprofessur—Artificial Intelligence' and FH-Call 30, project 30-25: 'Artificial Intelligence and Virtual Reality Lab'. We would also like to thank the municipality of Neuhaus for its great cooperation.

References

[1] S. F. Ahmed, et al., "Towards a secure 5G-enabled Internet of Things: A survey on requirements, privacy, security, challenges, and opportunities," *IEEE Access*, 2024.

[2] G. Indiveri and S.-C. Liu, "Memory and information processing in neuromorphic systems," *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1379–1397, 2015.

[3] S. Furber, "Large-scale neuromorphic computing systems," *Journal of Neural Engineering*, vol. 13, no. 5, p. 051001, 2016.

[4] C. Mead, *Analog VLSI and Neural Systems*. Addison-Wesley, 2020.

[5] A. Sandberg, "Energetics of the brain and AI," *arXiv preprint arXiv:1602.04019*, 2016.

[6] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, and B. Kay, "Opportunities for neuromorphic computing algorithms and applications," *Nature Computational Science*, vol. 2, no. 1, pp. 10–19, 2022.

[7] R. P. Rao and D. H. Ballard, "Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects," *Nature Neuroscience*, vol. 2, no. 1, pp. 79–87, 1999.

[8] R. A. Adams, S. Shipp, and K. J. Friston, "Predictions not commands: active inference in the motor system," *Brain Structure and Function*, vol. 218, pp. 611–643, 2013.

[9] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Federated learning for Internet of Things: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1622–1658, 2021.

[10] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[11] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, pp. 78–80, 2004.

[12] Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, and A. Hirose, "Recent advances in physical reservoir computing: A review," *Neural Networks*, vol. 115, pp. 100–123, 2019.

[13] K. Friston, "The free-energy principle: a unified brain theory?" *Nature Reviews Neuroscience*, vol. 11, no. 2, pp. 127–138, 2010.

[14] T. Parr and K. J. Friston, "Generalised free energy and active inference," *Biological Cybernetics*, vol. 113, no. 5, pp. 495–513, 2019.

[15] B. Sedlak, V. C. Pujol, P. K. Donta, and S. Dustdar, "Active inference on the edge: A design study," in *2024 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, Mar. 2024, pp. 550–555.

[16] E. Bollt, "On explaining the surprising success of reservoir computing forecaster of chaos? The universal machine learning dynamical system with contrast to VAR and DMD," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 31, no. 1, 2021.

[17] D. J. Gauthier, E. Bollt, A. Griffith, and W. A. Barbosa, "Next generation reservoir computing," *Nature Communications*, vol. 12, no. 1, pp. 1–8, 2021.

18

Online AI Benchmarking on Remote Board Farms

Maïck Huguenin, Baptiste Dupertuis, Robin Frund, Margaux Divernois, and Nuria Pazos

Haute Ecole Arc – HES-SO, Switzerland

Abstract

Benchmarks are essential for balancing the benefits and risks of AI by providing quantitative tools that guide responsible development. They offer objective and consistent metrics for accuracy, speed, and efficiency, enabling engineers to develop reliable products and services. Additionally, benchmarks help researchers gain new insights that can drive future innovations.

Today, numerous cloud-based AI development services allow software developers, even those without expertise in data science, to utilise AI models through APIs, SDKs, or applications. Benchmarking these models on cloud infrastructure is a feature offered by these services. However, few of these services are designed for edge deployment, where deep expertise in embedded programming and system integration is necessary to optimize and deploy AI models on specific embedded devices. Comparing benchmarking results across different embedded boards becomes increasingly complex when targeting devices from various providers.

The current project aims to design and implement a collaborative platform that enables researchers and developers to conduct experiments and research across various edge AI domains and edge AI devices. This will be achieved by sharing resources on a distributed virtual laboratory (dAIEdge-VLab). This platform will provide access to dedicated resources, tools, and services, allowing end users without expertise in embedded programming to perform live AI experiments, such as benchmarking, on remote embedded boards.

DOI: 10.1201/9788743808862-18

Keywords benchmarking, edge AI, Virtual Lab, distributed platform, AI-powered edge devices.

18.1 Introduction and Novelty Aspect

Edge AI and cloud AI offer two distinct approaches to deploying artificial intelligence, each with unique strengths and limitations depending on where data processing takes place. Edge AI is ideal for applications requiring real-time, on-site processing and enhanced data security, as computations happen directly on local devices. In contrast, cloud AI excels in scenarios demanding extensive computational power and large-scale data processing, leveraging remote servers for more complex tasks. These approaches can also work together, with edge devices handling initial data processing and sending more demanding tasks to the cloud for deeper analysis.

Edge AI processes data locally on devices positioned at the network's edge, closer to the source of data generation. This approach minimizes latency, allowing for quicker, real-time responses. Since data does not need to be transmitted to the cloud, edge AI also improves privacy and security while lowering bandwidth usage. Although edge devices generally have less computational power compared to cloud AI, recent advancements in hardware have significantly enhanced their ability to handle complex AI tasks.

Two key challenges in the development of edge AI for industrial applications are the difficulty in reproducing results consistently across different edge devices, and the complexity involved in configuring heterogeneous platforms, which can lead to lengthy evaluation times. As a result, comparing benchmarking results between various embedded boards becomes increasingly complicated, especially when targeting devices from different manufacturers.

This work addresses the identified challenge by introducing a collaborative platform, dAIEdge-VLab, which enables researchers and developers to remotely conduct experiments and research across various edge AI domains and devices. With dAIEdge-VLab, users without embedded programming expertise can easily deploy edge AI models and applications on remotely accessible embedded boards and retrieve the experimental results. The dAIEdge-VLab's architecture is designed for scalability, allowing owners of AI-powered edge devices to seamlessly integrate their hardware into the dAIEdge-VLab infrastructure.

The structure of the paper is as follows: Section 1.2 compares our dAIEdge-VLab solution with existing state-of-the-art methods for remote deployment on board farms. Section 1.3 presents two architectures of the dAIEdge-VLab, a centralized and a distributed version. Section 1.4 delves into the implementation details of the dAIEdge-VLab. Finally, Section 1.5 concludes the paper and outlines directions for future work.

18.2 State-of-the-art

AI Benchmarking can be defined as the process of evaluating and comparing the performance, efficiency, and capabilities of AI models, algorithms, or systems against standardized metrics and tasks. Key performance indicators can be classified into two main categories: (i) hardware-agnostic metrics, which are independent of the target device and applied libraries, such as accuracy, model size or number of parameters; and (ii) hardware-specific metrics, which depend on the target device, such as inference speed, power consumption, or memory usage. These metrics determine how well an AI model or system performs in real-world edge environments.

Benchmarking typically involves running AI models on predefined datasets or tasks, such as image recognition, natural language processing, or autonomous decision-making, and measuring their effectiveness against other models or industry standards. Since edge devices have limited computational resources compared to cloud servers, benchmarking helps identify the most suitable AI models and optimizations for these resource-constrained environments. The goal is to ensure that AI applications running on the edge can meet the necessary requirements for real-time decision-making, energy efficiency, and overall performance in diverse scenarios, such as autonomous systems, IoT, and industrial automation.

While many popular edge AI frameworks (both vendor-agnostic and proprietary) offer built-in benchmarking tools to extract key performance metrics, they do not guarantee a standardized procedure for fair comparison. To address this, initiatives like MLPerf Inference Edge [1] from the MLCommons foundation [2] aim to provide a representative benchmarking suite that fairly assesses edge ML system performance. However, even though all participants follow the same procedure to publish their benchmarking results, the tests are conducted at the edge device owner's facilities, and there is no remote control to ensure proper hardware setup.

In the scientific literature, various publications propose methodologies for the fair comparison of hardware, algorithms, and optimization techniques within the embedded design space. QuTiBench [3] introduces a novel multi-tier benchmarking framework that accommodates algorithmic optimizations, such as quantization, to help system developers assess the strengths and limitations of emerging compute architectures for specific neural networks. The QuTiBench team encourages community contributions to cover the full range of choices in Machine Learning system implementations. However, contributions ceased in 2021, and the project appears to no longer be maintained. Subsequent works, including [4], have adopted a similar approach to evaluate Machine Learning inference machines on Edge-class compute platforms. This testbed features two hardware compute engines—Raspberry Pi 4 (CPU-based) and Google Edge TPU accelerator—and two inference frameworks: TensorFlow-Lite and Arm NN.

Building on two decades of experience in developing embedded benchmarks, EEMBC has taken its first step into the Machine Learning space and is committed to keeping pace with industry advancements. To this end, they introduced the EEMBC MLMark® benchmark [5], specifically designed to assess the performance and accuracy of embedded inference systems. Unlike benchmarks that allow private optimizations, MLMark requires that all implementations be made publicly available in its repository. Version 1.0 provides source code and libraries for a range of platforms, including Intel® CPUs, GPUs, and neural compute sticks with OpenVINO®; NVIDIA® GPUs with TensorRT; and Arm® Cortex®-A CPUs and Arm Mali^{TM} GPUs using Neon^{TM} technology and OpenCL^{TM}, respectively.

To ensure fair comparison across different types of AI-powered edge devices (such as MPUs, GPGPUs, MCUs, NPUs, and VPUs), a unified benchmarking service offering remote access to this diverse range of embedded boards is essential. An online benchmarking platform would greatly facilitate this need by enabling consistent and equitable evaluation across varied hardware.

Recently, several edge device providers, including STM, Qualcomm and Intel, have introduced their initial solutions for online remote benchmarking. STM offers the "STM32Cube.AI Developer Cloud"[6], a free online platform designed to help developers create, optimize, benchmark, and generate AI solutions specifically for STM32 microcontrollers and microprocessors, which are based on ARM Cortex processors. The platform also supports AI hardware acceleration (NPU) when available on the target device.

Qualcomm® AI Hub [7] simplifies the deployment of AI models for vision, audio, and speech applications on edge devices by providing automatic optimization and validation tools. It offers over 100 pre-optimized AI models and enables seamless integration with Snapdragon® and Qualcomm platforms, with easy access to real devices for testing and profiling.

Similarly, Intel has launched the "Intel Tiber Developer Cloud" [8], which promises unrestricted access to Intel's latest edge computing hardware and software platforms for developer.

In addition, other vendor-agnostic commercial solutions have recently emerged, such as the Impulse Embedded Remote Benchmarking Service [9]. This platform enables developers to remotely test AI models on a wide range of GPU hardware, from entry-level devices like the Jetson Nano to high-performance systems such as the AGX Orin. Based on our understanding, only GPGPU devices are currently available for remote benchmarking.

To the best of our knowledge, no vendor-agnostic, multi-platform, and scalable solution for online remote benchmarking currently exists. This work addresses that gap by introducing a versatile, vendor-neutral, and scalable virtual laboratory designed to facilitate edge AI benchmarking experiments, bridging the divide between data scientists and the embedded systems domain.

18.3 dAIEdge-VLab Architecture

The dAIEdge-VLab is designed to help users who lack expertise in embedded programming or do not have direct access to specific hardware. It will enable them to conduct real-time AI experiments on a remote farm of embedded boards. These experiments could include AI model benchmarking (using randomly generated data), AI application benchmarking (with pre-existing test datasets), support for hardware-in-the-loop neural architecture search (NAS), and even benchmarking for on-device training.

In terms of hardware compatibility, the dAIEdge-VLab is built to support a wide range of embedded boards, spanning from high-performance MPUs and GPGPUs to energy-efficient MCUs and specialized NPUs. On the software side, it accommodates both Linux-based and real-time operating systems, as well as bare-metal solutions. Additionally, the platform will be compatible with widely used vendor-agnostic inference runtimes along with proprietary AI engines.

A virtual lab for online benchmarking of edge AI applications typically consists of several architectural components designed to facilitate remote experimentation, testing, and optimization of AI models across different edge hardware platforms. Below is an outline of its overall structure together with the main functionalities:

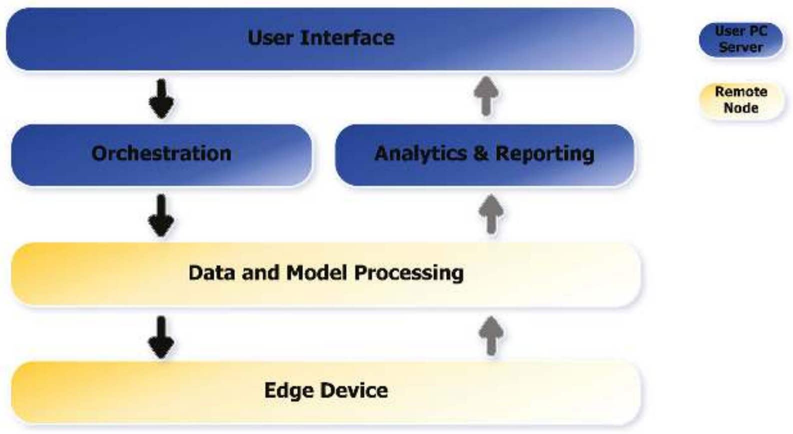


Figure 18.1 Virtual Lab Layer Model.

1. User Interface (UI) Layer

- **Web Interface/Dashboard**: The virtual lab offers a user-friendly web-based dashboard that allows data scientists and developers to interact with the system. This includes uploading AI models, selecting target hardware platforms, configuring benchmarking parameters, and monitoring results.
- **CLI API**: An API-based interface allows seamless integration with custom AI pipelines.
- **AI Application Management**: Users can upload pre-trained AI models, select from a model zoo, or choose a complete AI application. The system also supports model versioning, enabling easy retrieval and comparison of different versions.

2. Orchestration Layer

- **Resource Management and Scheduling**: This component handles the orchestration of resources across multiple platforms, ensuring efficient allocation of hardware for testing and benchmarking. It manages queues, schedules jobs, and optimizes resource usage based on platform availability.
- **Virtualization and Containerization**: Models and benchmarking tasks are often containerized (e.g., using Docker) to ensure compatibility across diverse hardware. This enables easy deployment across various edge devices, regardless of their underlying operating system or architecture. This layer deals with the management of containers on edge devices running general-purpose operating systems.

- **Benchmark Configuration**: Users define test cases, including hardware configurations, datasets, and performance metrics such as latency, power consumption, and accuracy. The ultimate goal is to define the technical requirements to compare benchmarking results among different configurations.

3. Edge Device Layer (Board farm)

- **Multi-Platform Hardware Pool**: The system integrates a variety of edge devices, ranging from low-power IoT devices (like STM32 MCUs, RISC-V based platforms, etc.) to high-performance systems (such as Raspberry Pi or Jetson Orin Nano or AGX). This diversity ensures comprehensive testing across different edge environments.
- **Remote Access & Control**: The lab allows remote access to real, physical hardware. Developers can deploy AI models directly to these devices for testing under real-world conditions, avoiding the limitations of simulated environments.

4. Data and Model Processing Layer

- **Data Preprocessing and Inference**: This layer handles the preprocessing of input data and manages the inference execution on edge devices. It ensures that the models are efficiently adapted to the target hardware's limitations, such as memory and computational power.
- **Performance Metrics Collection**: During benchmarking, this layer monitors critical performance metrics like inference time, throughput, power consumption, and memory usage, which are fed back into the system for analysis.

5. Analytics & Reporting Layer

- **Results Analysis & Visualization**: The benchmarking results are processed and presented through interactive dashboards, allowing users to compare metrics across different hardware platforms. Detailed reports include insights into energy efficiency, processing latency, accuracy, and other relevant performance factors.

This virtual lab architecture is designed to bridge the gap between data scientists and embedded systems by enabling seamless testing, monitoring, and optimization of AI models on real edge hardware in a scalable, vendor-agnostic manner.

The proposed dAIEdge-VLab platform features a modular architecture that allows for easy integration of remote embedded boards. For the implementation of the proposed dAIEdge-VLab, we have followed an agile four-step methodology: (i) classifying functional and non-functional requirements for the envisioned online AI benchmarking solution and identifying gaps in existing solutions; (ii) designing a layer model for the dAIEdge-VLab; (iii) developing an initial PoC of the dAIEdge-VLab based on a centralized architecture managed by a central server where all registered embedded boards are listed and orchestrated; and (iv) designing a distributed version to enhance the flexibility, scalability, and security of the dAIEdge-VLab.

Figure 18.2 illustrates the overall architecture of the centralized dAIEdge-VLab. The key components of this architecture are:

- **Remote User**: A user initiating an AI experiment on a remote node submits a request via a web-based user interface. No manual installation is required on the user's side.
- **dAIEdge-VLab Server**: The dAIEdge-VLab Server receives the user request and forwards it to the corresponding Remote Host that is connected to the target Remote Node. All available Remote Nodes are registered to the dAIEdge-VLab Server.
- **Remote Host**: Located at the facility of the Remote Node owner, the Remote Host is responsible for executing node-specific scripts. It compiles and deploys the AI application to the Remote Node, retrieves the benchmarking results, and sends them back to the user through the dAIEdge-VLab Server.
- **Remote Node**: This refers to the embedded board where the AI experiments, such as model benchmarking, are executed. A single Remote Host can manage multiple Remote Nodes at the owner's site.

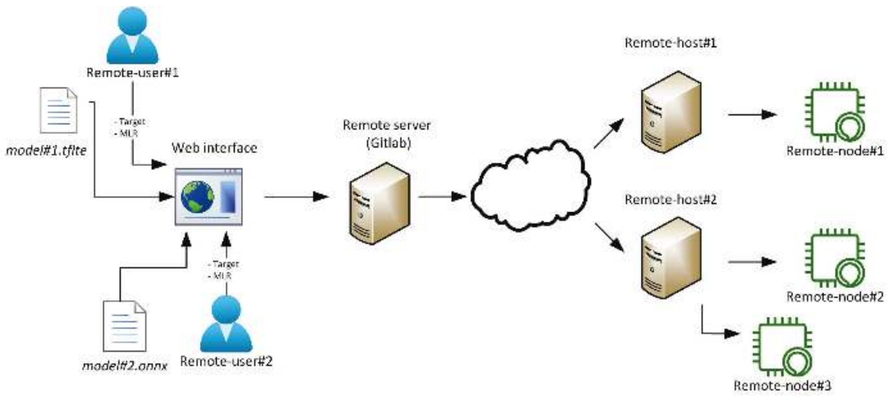


Figure 18.2 Centralized dAIEdge-VLab Architecture. ⏎

Figure 18.3 illustrates the architecture of the distributed dAIEdge-VLab, which differs slightly from the centralized version due to the absence of a dAIEdge-VLab server for managing the registered Remote Nodes. The primary components are as follows:

- **Remote User**: A user wishing to launch an AI experiment on a Remote Node submits their request through a web interface. Unlike the centralized version, a lightweight local installation of the web interface is required on the user's side to interact with the dAIEdge-VLab API.
- **Peer-to-Peer Content Delivery Network:** This open system manages the exchange of data between nodes without the need for a centralized server, enabling decentralized communication.
- **Remote Host:** Situated at the Remote Node owner's location, the Remote Host is responsible for running node-specific scripts, compiling and deploying the AI application to the Remote Node, retrieving the benchmarking results, and transmitting them back to the user via the peer-to-peer content delivery network.
- **Remote Node:** This is the embedded board where AI experiments, such as model benchmarking, are executed. Multiple Remote Nodes can be managed by a single Remote Host within the owner's facility.

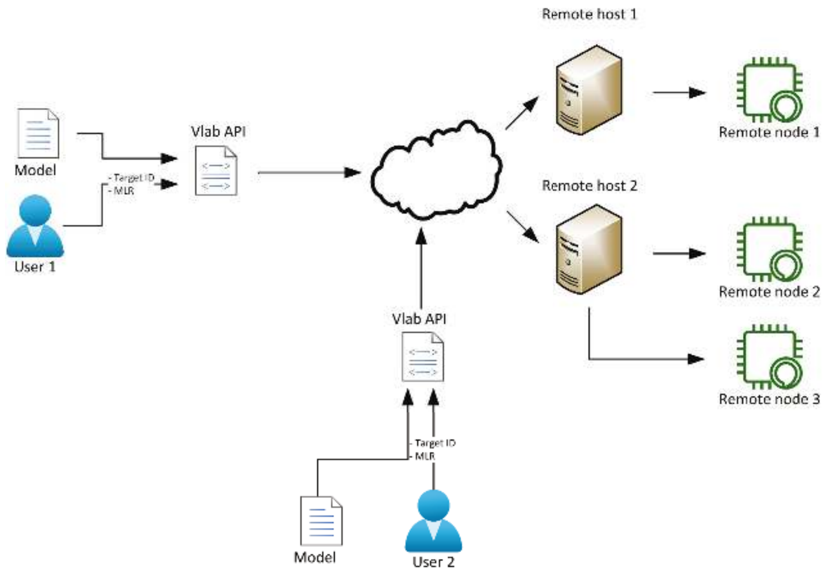


Figure 18.3 Decentralized dAIEdge-VLab Architecture. ⏎

18.4 dAIEdge-VLab Implementation

To demonstrate the feasibility of the centralized dAIEdge-VLab, the initial PoC implementation uses a GitLab server to run CI/CD pipelines on GitLab-hosted runners. Each Remote Host must install and register a runner for every Remote Node it manages. These runners execute node-specific tasks triggered by the GitLab CI/CD pipeline. The process followed by the centralized dAIEdge-VLab to initiate a benchmarking experiment is outlined as follows:

1. User Interface layer: A Remote User submits a benchmarking request via the web interface, specifying the trained model, target Remote Node, and Machine Learning Runtime (MLR) to be used from the available options.
2. Orchestration layer: The dAIEdge-VLab GitLab Server processes the request by triggering the GitLab CI/CD pipeline. A Docker container with the required tools for the specific node is also deployed on the Remote Host.
3. Data and Model Processing layer: The Remote Host executes the node-specific scripts (AI-Support, AI-Build, AI-Deploy, AI-Manager) using the tools within the Docker container, producing a binary tailored for the target Remote Node. This binary is then deployed onto the Remote Node.
4. Edge Device layer: Runs the inference/benchmarking process and sends back the collected benchmarking metrics to the Remote Host.
5. Analytics and Reporting layer: The Remote Host retrieves the benchmarking metrics from the Remote Node and returns them to the dAIEdge-VLab GitLab Server via the GitLab artifacts. The server generates a benchmarking report, which is then sent back to the Remote User for visualization through the web interface.

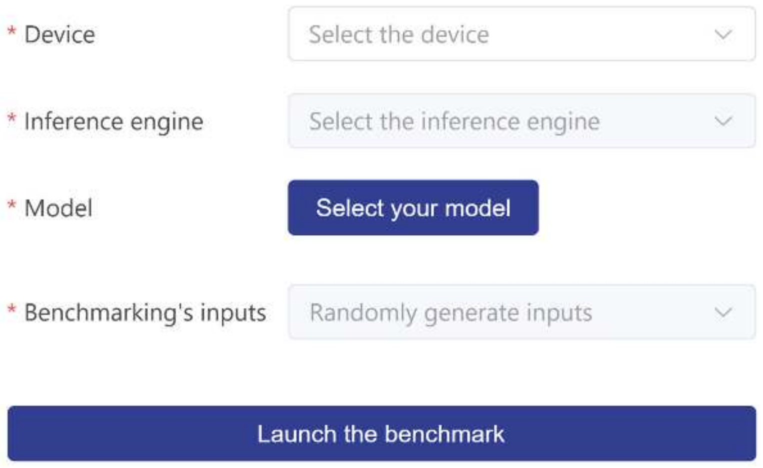
The distributed version of the dAIEdge-VLab differs from the centralized one primarily in its implementation of the orchestration layer, specifically the resource management and scheduling components. In this version, a dAIEdge-VLab API on the Remote User side is responsible for requesting access to the target Remote Node to run the benchmarking experiment. Once access is granted based on node availability, a peer-to-peer content delivery network, using IPFS [10], will transmit the request to the selected board via the associated Remote Host.

Managing the preprocessing of input data and inference execution on the target edge device requires a structured approach for implementing the node-specific scripts in the Data and Model Processing layer. This setup includes four main scripts and a Docker image containing all the necessary tools and packages for the specific node. These components are organized as follows:

1. **AI_Support**:
   - Provides documentation, dependencies, and installation support for the Remote Node. For edge devices with OS support, an OS kernel binary image along with system libraries will be provided for flashing onto the target board.
   - Includes scripts to initiate code generation for the benchmarking application specific to the Remote Node, which will be saved in the AI_Project folder.
2. **AI_Project**:
   - Stores the generated benchmarking application that is optimized for execution on the Remote Node.
3. **AI_Build**:
   - Contains the build environment with toolchains required by the inference framework to cross-compile the benchmarking application.
   - Provides scripts for cross-compiling the benchmarking application on the Remote Host.
4. **AI_Deploy**:
   - Includes tools for deploying the generated binary file onto the Remote Node.
   - Provides scripts to automate the deployment of the binary for the benchmarking application onto the Remote Node.
5. **AI_Manager**:
   - A management tool integrated with the inference framework to control the target board and monitor the system's status.
   - Includes scripts to establish a connection with the Remote Node and retrieve benchmarking metrics.

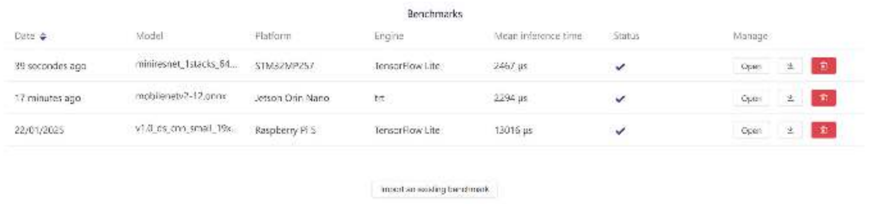
This structure ensures seamless execution of remote benchmarking tasks across various edge devices while maintaining consistency in deployment and management.

A first PoC for the centralized dAIEdge-VLab is operational. Below are screenshots showcasing the web interface dashboards for both input selection (see Figure 18.4) and visualization of benchmarking results (see Figure 18.6). A history of previously executed experiments is saved locally in the browser cache (see Figure 18.5), allowing users to review and compare different benchmarking experiments in the future.



* Device	Select the device
* Inference engine	Select the inference engine
* Model	Select your model
* Benchmarking's inputs	Randomly generate inputs

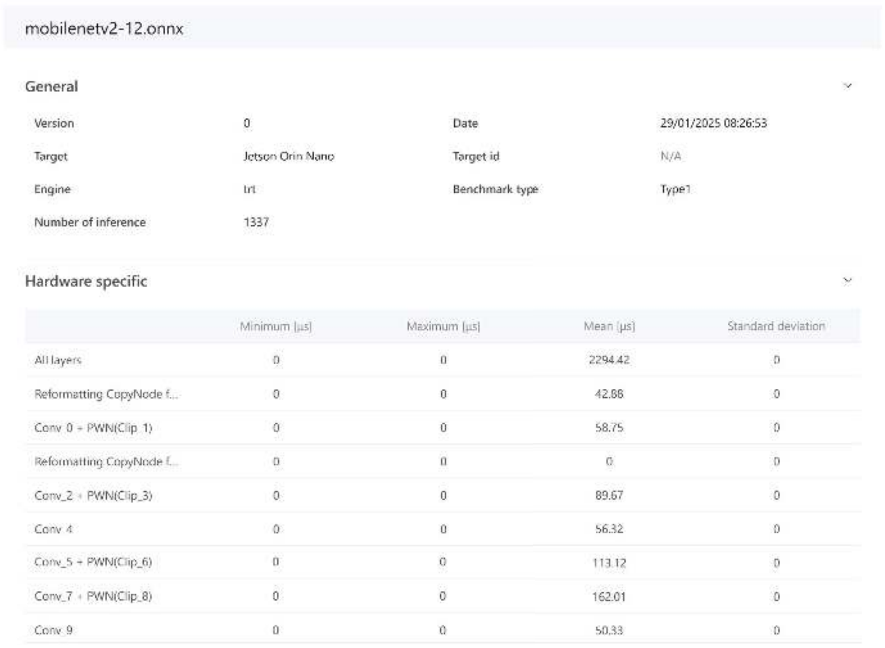
Figure 18.4 User Input Selection through Web Interface. ⏎



Benchmarks

Date	Model	Platform	Engine	Mean inference time	Status	Manage
39 secondes ago	miniresnet_1stacks_64...	STM32MP257	TensorFlow Lite	2467 µs	✓	Open
17 minutes ago	mobilenetv2-12.onnx	Jetson Orin Nano	trt	2294 µs	✓	Open
22/01/2025	v1.0_ds_cnn_small_19x...	Raspberry Pi 5	TensorFlow Lite	13016 µs	✓	Open

Figure 18.5 List of Benchmarking Results. ⏎


mobilenetv2-12.onnx

General

Version	0	Date	29/01/2025 08:26:53
Target	Jetson Orin Nano	Target id	N/A
Engine	trt	Benchmark type	Type1
Number of inference	1337		

Hardware specific

	Minimum [µs]	Maximum [µs]	Mean [µs]	Standard deviation
All layers	0	0	2294.42	0
Reformatting CopyNode f...	0	0	42.88	0
Conv_0 + PWN(Clip_1)	0	0	58.75	0
Reformatting CopyNode f...	0	0	0	0
Conv_2 + PWN(Clip_3)	0	0	89.67	0
Conv_4	0	0	56.32	0
Conv_5 + PWN(Clip_6)	0	0	113.12	0
Conv_7 + PWN(Clip_8)	0	0	162.01	0
Conv_9	0	0	50.33	0

Figure 18.6 Visualisation of Model Benchmarking Results ⏎

The generated benchmark report is a JSON file that includes all the relevant keys, even those not retrieved from the Remote Node. These keys are organized into four main categories and used throughout the benchmark report:

1. **Hardware specific keys**
    - *FLOPs* (type: float): Floating-point operations per second.
    - *inference_latency* (type: json): This key contains a JSON file containing the latency measures (in seconds) of all performed inferences (*mean* (type: float), *std* (type: float), *min* (type: float) and *max* (type: float)).
    - *throughput* (type: float): Throughput of the system in inferences per second.
    - *latency_per_layers* (type: list): This key contains a list of json composed of all the layer's latency measures (<*element list*> (type: json), *layer_name* (type: str), *mean* (type: float), *std* (type: float), *min* (type: float), and *max* (type: float)).

- *preprocess_time* (type: json): Time in seconds for the pre-processing (*mean* (type: float), *std* (type: float), *min* (type: float) and *max* (type: float)).
- *postprocess_time* (type: json): Time in seconds for the post-processing processing (*mean* (type: float), *std* (type: float), *min* (type: float) and *max* (type: float)).
- *ram_size* (type: int): Size in bytes of the available RAM of the system.
- *ram_peak* (type: float): Percentage of the peak RAM usage.
- *flash_size* (type int): Size in bytes of the available FLASH.
- *flash_usage* (type: float): Percentage of the FLASH usage. It might not be relevant for Linux targets.
- *load_cpu* (type: float): Percentage of the CPU usage.
- *load_accelerator* (type: float): Percentage of the accelerator (GPU, NPU, etc.) usage.
- *temperature* (type: float): Temperature of the board in °C.
- *ambient_temperature* (type: float): Room temperature in °C.
- *power_consumption* (type: float): Power consumption of the board in Watt.
- *energy_efficiency* (type: float): Operations Per Watt (OPW).

2. **Hardware agnostic keys**
   - *model_size* (type: int): Size in bytes of the trained model on the embedded board (after any compression / optimisation performed by the MLR).
   - *nb_parameters_model* (type: int): Number of parameters of the trained model.
   - *accuracy* (type: float): Accuracy of the trained model.

At the conclusion of each benchmarking experiment, two log files, *user.log* and *error.log*, are generated to help users track any issues. The contents of these logs are displayed on the web interface for user review:

- ***user.log***: This file provides general information or warnings related to the benchmarking process.
- ***error.log***: This file logs any errors encountered during the process, particularly in cases where the JSON benchmarking report could not be generated (e.g., due to an unsupported model format or insufficient memory on the target board).

These logs allow users to identify and troubleshoot potential problems in the benchmarking process.

A Continuous Testing (CT) strategy has been implemented to ensure the dAIEdge-VLab platform's stability. To achieve this, a suite of tests is automatically triggered on a weekly basis, as well as after every push or merge request. These pipelines are executed across all registered Remote Nodes, supported Machine Learning Runtimes (MLRs), and models from the internal model zoo.

Remote Node owners interested in adding their boards to the dAIEdge-VLab can find the guidelines under [11]. An MPU/linux template as well as an MCU/RTOS template are available to speed up the integration of new Remote Nodes.

18.5 Conclusion

Evaluating machine learning inference on edge devices is an essential step before selecting the optimal embedded board or optimizing the ML model to suit the target hardware. However, accessing boards from various manufacturers and getting familiar with their unique programming environments is a complex and often inaccessible task. To address this challenge and simplify access to a diverse range of embedded boards, we propose in this paper a solution for conducting ML benchmarking experiments on remote board farms.

For the implementation of the proposed dAIEdge-VLab, we adopted a five-layer model and followed a four-step methodology: (i) classifying functional and non-functional requirements for an online AI benchmarking solution on remote board farms and identifying gaps in existing solutions; (ii) designing a layer model for the dAIEdge-VLab; (iii) developing an initial PoC of the dAIEdge-VLab based on a centralized architecture managed by a central server; and (iv) designing a distributed version to enhance the flexibility, scalability, and security of the dAIEdge-VLab. This incremental approach has resulted in a scalable solution that is ready for release and capable of integrating third-party embedded boards in a distributed manner.

The current implementation of dAIEdge-VLab supports a wide range of Linux-based MPUs, including Raspberry Pi 4B and Raspberry Pi 5, as well as GPGPUs such as Nvidia Jetson Xavier and Nvidia Jetson Orin Nano. It also accommodates MCU+NPU platforms like the STM32MP257, along with bare-metal MCUs such as STM32L4R9 and NXP LPC55S69. Additionally, it includes support for vendor-agnostic Machine Learning runtimes, such as ONNX Runtime, TensorFlow Lite, and TensorFlow Lite for microcontrollers, as well as vendor-specific Machine Learning runtimes like CUBE-AI and TensorRT.

dAIEdge-VLab currently assists users who may not have expertise in embedded programming or access to the target embedded board by enabling real-time model benchmarking on a remote embedded board. In the future, additional experiments will be introduced, including AI application benchmarking, support for hardware-in-the-loop neural architecture search (NAS), and the ability to launch on-device training directly on the target board.

In the near future, we aim to enhance the distributed version of the dAIEdge-VLab by incorporating blockchain technology to share and manage information about the registered embedded boards in a decentralized way.

Acknowledgements

This work has been funded by the European Union's Horizon Europe project dAIEDGE under grant agreement No. 101120726. The associated partners from Switzerland received funding from the Swiss State Secretariat for Education, Research and Innovation (SERI).

References

[1] V. J. Reddi et al ; "MLPerf Inference Benchmark", 2020. [Online]. Available: https://arxiv.org/abs/1911.02549

[2] hhttps://mlcommons.org/benchmarks/inference-edge/

[3] M. Blott, L. Halder, M. Leeser, and L. Doyle, "Qutibench: Benchmarking neural networks on heterogeneous hardware," J. Emerg. Technol. Comput. Syst., vol. 15, no. 4, Dec. 2019. [Online]. Available: hhttps://doi.org/10.1145/3358700

[4] P. Amanatidis, G. Iosifidis, and D. Karampatzakis, "Comparative evaluation of machine learning inference machines on edge-class devices," in Proceedings of the 25th Pan-Hellenic Conference on Informatics, ser. PCI '21. New York, NY, USA: Association for Computing Machinery, 2022, p. 102–106. [Online]. Available: hhttps://doi.org/10.1145/3503823.3503843

[5] Torelli, Peter and Bangale, Mohit. "Measuring Inference Performance of Machine-Learning Frameworks on Edge-class Devices with the MLMark^{TM} Benchmark" . https://www.eembc.org/mlmark

[6] https://www.st.com/en/development-tools/stm32cubeai-dc.html

[7] https://aihub.qualcomm.com/
[8] https://www.intel.com/content/www/us/en/developer/tools/devcloud/services.html
[9] https://www.impulse-embedded.co.uk/remote-gpu-system-benchmarking.htm
[10] https://docs.ipfs.tech/
[11] http://public-virtuallab-docs-tica-23tica04-daiedge-poc-6eb4f1c0e44fb2.pages-rad.ing.he-arc.ch/docs/

19

Optimising Neural Networks for Water Stress Prediction in Europe: A Sustainable Approach

Laura Sanz-Martín[1], Manal Jammal[2], and Javier Parra-Domínguez[1]

[1]University of Salamanca, Spain
[2]IoT Digital Innovation Hub, Spain

Abstract

Sustainability in water resource management is critical, given the necessity to monitor and predict key indicators such as SDG 6.4.2: Water Stress. This indicator, which measures the ratio of water resources abstracted relative to their availability, is vital for assessing the pressure on water resources and ensuring their long-term sustainability. The objective of this study is to compare various neural network architectures utilizing different optimisers to predict water stress. The analysis was based on a dataset sourced from AQUASTAT, encompassing data from 28 European countries. Several neural network architectures, with configurations ranging from two to four layers, were implemented, and evaluated using optimisers including SGD, Adam, RMSprop, and Adagrad. The findings revealed that a three-layer architecture combined with the Adam optimiser delivered the best performance, achieving an MSE of 0.02187 and a R^2 of 0.9745, indicating high predictive accuracy. Nevertheless, a two-layer architecture with the SGD optimiser also exhibited strong performance, highlighting its simplicity and effectiveness. These results underscore the importance of meticulous selection of both architecture and optimiser when predicting critical indicators such as water stress. This study not only enhances the accuracy of water-related risk predictions

DOI: 10.1201/9788743808862-19

but also supports informed decision-making for sustainable water resource management in Europe.

Keywords: sustainability, water stress, artificial neural networks (ANN), artificial intelligence (AI), optimizers.

19.1 Introduction and Background

For several decades, climate change has affected the hydrological cycle, modifying rainfall patterns, raising water temperature, and intensifying extreme phenomena such as floods and droughts. These changes put pressure on water resources, with potential adverse impacts on ecosystems and human health. In addition, sea level rise may intensify the salinization of groundwater and estuaries, decreasing the presence of freshwater in coastal areas. Furthermore, climate variations in water volume and quality are expected to impact food availability and access to water, particularly in arid areas, and impact water infrastructure such as irrigation systems and hydropower [1]. In Europe, climate change will accentuate the differences in water resources between north and south. In the north, precipitation is expected to increase, improving water availability, but with risks of flooding and increased greenhouse gas emissions from the decomposition of carbon in the soil. In the south, especially in the Mediterranean regions, there will be a decrease in water supply, increasing irrigation demand and agricultural vulnerability, as well as problems of erosion, salinization and soil degradation. Changes in rivers and more frequent droughts and floods will affect the carbon cycle, complicating mitigation efforts [2].

Water stress occurs when the demand for water exceeds the quantity available during a given period or when poor quality limits its use, resulting in the deterioration of freshwater resources in terms of quantity and quality (https://www.eea.europa.eu/). Water stress, intensified by climate change, population growth and economic development, affects both human consumption and key sectors such as agriculture, energy and industry. This phenomenon, aggravated by the degradation of ecosystems and the depletion of water sources, calls for adaptive water management to ensure sustainability and economic stability. Global warming is expected to significantly alter water availability, with increased risks of droughts and floods. By 2030, global water consumption is projected to exceed 160% of available supply, exacerbating global water challenges [3].

Water stress is closely related to Sustainable Development Goal (SDG) 6. Modifications in the hydrological cycle, such as changes in precipitation and water salinization, directly affect access to clean and safe water, especially in coastal areas and arid regions. This underscores the need to implement solutions that address not only the quantity of water available, but also its quality, in line with SDG 6 targets. To address these global challenges, it is crucial to improve the capacity to predict and manage water stress and other indicators related to water availability. In this context, technologies such as Artificial Intelligence (AI) and machine learning models play an increasingly relevant role. These models provide advanced tools for the analysis and prediction of complex water-related phenomena, such as water stress, by processing large amounts of climatic, hydrological, and agricultural data efficiently.

Early prediction of water stress is vital to mitigate impacts on agriculture, energy and other sectors that are highly dependent on water. AI models enable more accurate, real-time analysis of key indicators such as the water stress index and other relevant parameters. These predictions help optimize decision-making on water use, facilitating more effective adaptation strategies in response to climate variations. Despite the growing interest in the use of machine learning models for water resources management, there is a lack of studies that systematically compare various neural network architectures and optimizers specifically in the context of water stress prediction in Europe. Many previous works have focused on general applications of neural networks in water data modelling but have not performed a comprehensive assessment of how different configurations affect predictive accuracy in a specific regional context. Furthermore, the integration of AQUASTAT data, which spans multiple European countries, provides a unique opportunity to address this gap by facilitating the identification of patterns and trends that may be critical for sustainable water management. This study seeks to close this gap by providing a rigorous and contextualized comparison of machine learning models applied to European water data, thus contributing to the development of more effective decision-making tools for water resources management.

The objective of this study is to compare different artificial neural network architectures using different optimizers to predict water stress in 28 European countries, using an AQUASTAT dataset. Through this comparison, we seek to evaluate the performance of each architecture and optimizer in terms of predictive accuracy. Identifying the most effective configuration will not only contribute to improve informed decision making for sustainable water resources management in Europe but will also allow policy makers and water managers to select analytical tools that optimize water use.

This article is structured as follows: first, we will present a state-of-the-art review of the main papers on water stress prediction, analyzing the methodologies and approaches used in previous research. Subsequently, the use of different neural network architectures and optimizers in this context will be discussed, highlighting their advantages and disadvantages. Next, the methodology employed in this study will be described, including data selection. Finally, the results obtained from the comparison between different neural network architectures and optimizers will be presented and discussed, as well as the implications of the findings for sustainable water resources management in Europe, together with recommendations for future research.

19.2 State of the Art

Concern for water availability has gained relevance in a global context where climate change and the growing demand for water resources are intertwined with sustainability objectives. In this regard, the energy sector faces significant challenges, particularly in thermoelectric generation, which requires large volumes of water for cooling. Recent studies have assessed how periods of extreme heat impact electricity generation, highlighting the reduction in production at 1,326 thermoelectric plants in the European Union. Despite efforts to reduce water withdrawals, the number of watersheds experiencing water stress is expected to increase, emphasizing the need to implement integrated approaches to water and energy resource management [4].

On the other hand, the sensitivity of river basins to climate change has been analyzed using advanced methodologies that combine climate modelling and multi-model simulations. This approach has made it possible to classify different basins according to their vulnerability, revealing that, although the Nordic basins show high sensitivity, those of southern and central Europe face greater challenges in overcoming low flow and water stress thresholds [5]. This research underlines the importance of proactive and adaptive management to cope with the adverse effects of climate change on water availability.

In addition, the use of artificial intelligence techniques to characterize and predict water stress has gained attention in several regions. A study conducted in Hyderabad, India, demonstrates how models such as support vector regression (SVR) outperform traditional approaches such as ARIMA in drought prediction. The results highlight the ability of artificial intelligence to capture complex patterns in data, which can significantly contribute to more efficient water resource management [6].

Finally, the application of thermography in agriculture has been shown to be a valuable resource for detecting water stress in crops such as maize. A study in Thailand suggests that continuous monitoring of the Crop Water Stress Index (CWSI) can effectively predict yield losses under drought conditions. This approach highlights the importance of integrating technology into agricultural management to optimize water use [7]. Taken together, these studies evidence the need to adopt holistic strategies that address water stress from multiple perspectives, combining climate research, artificial intelligence, and innovation in agricultural practices.

19.3 Material and Methods

19.3.1 Data

Our database includes 4 variables and 337 records, with information from 28 European countries related to SDG 6.4 indicators: Water Use Efficiency and Water Stress, covering the period from 2010 to 2021. The Water Use Efficiency indicator measures the economic value added generated by each unit of water used (expressed in dollars per cubic meter). Its objective is to evaluate how much economic value is produced with each cubic meter of water withdrawn and used in the economy, considering the following sectors:

1. Agriculture, the largest consumer of water.
2. Industry, where water is essential for production processes.
3. Services, where its use can be more efficient.

The Water Stress Level indicator measures the percentage of renewable freshwater withdrawal in a region or country, comparing water demand with total availability. It is expressed as the percentage of renewable water resources withdrawn in a year. A high value indicates higher water stress, while a low value reflects more sustainable water use.

Indicator 6.4.2 is defined as the ratio of total freshwater withdrawal (TFWW) in all major sectors to the difference between total renewable freshwater resources (TRWR) and environmental flow requirements (EFR). It is calculated using the following formula [12]:

$$\textit{Water Stress}\ (\%) = \frac{TFWW}{TRWR - EFR} \times 100. \tag{19.1}$$

Table 19.1 presents descriptive statistics on Water Use Efficiency and Water Stress. Water use efficiency averages 134.95 \$/m^3, with a standard deviation of 215.60 \$/m^3, indicating considerable variability among European countries. The minimum value observed is \$6.29/m^3 and the maximum reaches \$1,294.91/m^3, suggesting large differences in water management. The median is 80.41 \$/m^3, indicating that half of the observations are below this value.

For water stress, the average is 20.74%, with a standard deviation of 19.13%, reflecting significant variations. The minimum value is 0.99%, showing that some areas do not face water stress, while the maximum is 91.29%, indicating a risk to sustainability in certain regions. The median of 17.24% indicates that half of the areas analyzed have relatively low water stress.

Table 19.2 shows the descriptive statistics of water stress in European countries from 2010 to 2021. In general, there is a large variability in water stress levels between countries. Malta has the highest average level at 83.19%, indicating chronic pressure on its water resources. Other countries with high stress levels are Belgium (56.43%) and Bulgaria (41.19%), while Croatia (1.45%) and Austria (9.08%) show considerably lower levels.

Comparing the average data with that of 2021, some countries, such as Austria, Belgium and Czechia, have reduced their water stress in 2021 relative to the averages of the 2010-2021 period. For example, Belgium decreases from 56.43% to 51.88%. However, other countries, such as Cyprus and Denmark, have seen an increase in their water stress levels in 2021, which may reflect higher demand or lower water availability.

Although water stress in Malta decreased slightly in 2021 (78.28%), it is still very high. This highlights the continued pressure on its water resources, attributable to limited freshwater availability. In summary, the data highlight both improvements in water management in certain countries and increases in water stress in others, evidencing the various challenges Europe faces in managing its water resources.

Table 19.1 Descriptive statistics for the two variables under study ⏎

	Water Use Efficiency(\$/m^3)	**Water Stress (%)**
Mean	135,95	20,74
Std	215,60	19,13
Min	6,29	0,99
25%	34,99	6,01
50%	80,41	17,24
75%	143,09	29,8
Max	1294,91	91,29

Table 19.2 Descriptive statistics for the level of water stress of the countries under study

Country	Mean	Std	Min	50%	Max	2021
Austria	9,079396	0,349331	8,67643	9,052531	9,643548	8,67643
Belgium	56,42826	7,228664	49,06634	52,91819	73,13268	51,87961
Bulgaria	41,19484	2,675824	37,5194	41,01338	47,19492	37,5194
Croatia	1,44622	0,065285	1,289266	1,469994	1,518462	1,478435
Cyprus	30,09158	2,179796	27,46036	29,72936	34,89612	32,12138
Czechia	24,90686	2,826334	20,51399	24,8236	29,66849	20,51399
Denmark	24,78854	2,818227	19,69984	24,89713	29,90311	26,40427
Estonia	16,62548	3,929511	9,231085	18,13312	20,28139	10,8198
Finland	7,736582	1,909514	5,454114	7,114062	11,51814	7,114062
France	23,72531	1,359882	21,59814	23,68161	26,39062	21,59814
Germany	39,52151	5,377602	33,50192	37,46542	49,9216	35,35166
Greece	20,28583	0,252374	19,93815	20,28121	20,6838	20,6838
Hungary	8,098914	0,880236	6,775475	8,12532	9,274611	8,070121
Ireland	10,63118	7,986741	4,001176	5,971554	22,20506	22,20506
Italy	29,79749	0,111825	29,64578	29,80592	29,94407	29,64578
Latvia	1,259613	0,351461	0,992929	1,074131	2,175015	1,067831
Lithuania	2,986097	1,668811	1,834102	2,399063	7,467916	1,834102
Luxembourg	3,866086	0,206621	3,573798	3,8267	4,328358	3,963516
Malta	83,18589	5,022172	75,44555	82,18479	91,28713	78,28007
Netherlands	17,59585	2,071677	15,02981	16,90208	20,75185	16,07566
Northern Ireland	13,86417	0,665184	12,42114	14,22251	14,35465	14,35465
Poland	35,61036	3,509251	30	35,88958	41,22534	32,07684
Portugal	15,3874	2,708669	12,31571	17,2372	17,97442	12,31571
Romania	6,26299	0,465228	5,822489	6,060762	7,362606	7,362606
Slovakia	2,501927	0,15682	2,388124	2,421629	2,862737	2,436631
Slovenia	6,420104	0,518243	5,749155	6,296146	7,813387	6,294794
Spain	43,9606	3,191685	39,8289	43,25404	50,10225	43,25404
Sweden	3,592352	0,135445	3,427128	3,567388	3,880231	3,582973

A threshold of 25% has been established to evaluate water stress, where values below are considered safe and values above represent an increasing threat. This stress is classified into five categories: no stress (<25%), low (25-50%), medium (50-75%), high (75-100%) and critical (>100%). Figure 19.1 illustrates the frequency of water stress values in different ranges, using a colour palette ranging from green (low stress) to red (high stress). Most of the observations are in the safe range, indicating that many regions do not face significant pressure on their water resources, as reflected in the high frequency of values between 0 and 25%. However, there are a notable number of areas in the medium stress category (50-75%), with fewer areas in high stress (75-100%). Although there are not many observations that exceed

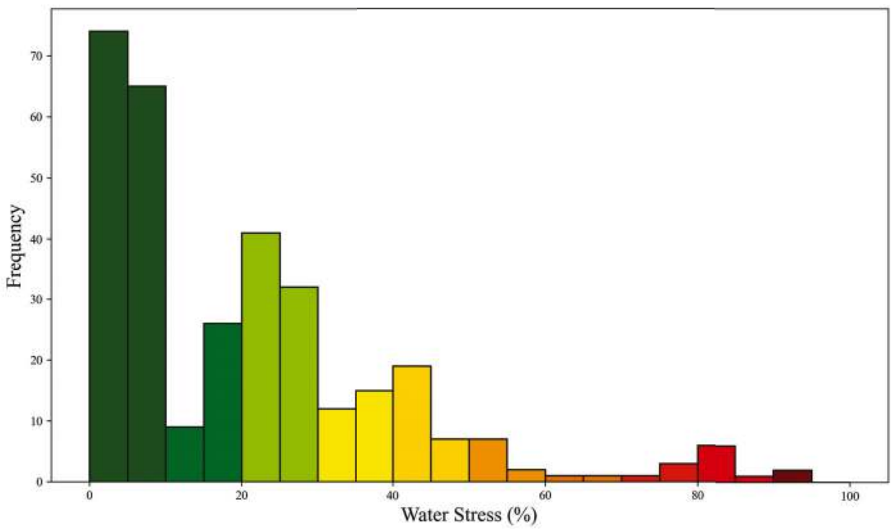


Figure 19.1 Frecuency of water stress (%). ⏎

100% (critical category), the presence of some indicates that these regions face serious water sustainability concerns.

19.3.2 Methodology

19.3.2.1 Data

The data were processed to select only the relevant column and then normalized using a MinMax scaler, which ensures that all values were within the range [0, 1]. Normalization is crucial to improve the stability and efficiency of neural network training. Subsequently, temporal sequences were created to capture the dependencies between the data over time. For each output (target) value, the values of the previous 28 observations were used as input. This process of creating sequences turns the problem into one of sequential prediction.

The normalized data were divided into three subsets: training set (70%), used to adjust the model weights, validation set (15%) to monitor model performance during training and avoid overfitting, and test set (15%), reserved for evaluating the final model performance on unseen data.

19.3.2.2 Neural network architecture

Different dense layer architectures with ReLU activations were evaluated for each hidden layer. The following layer configurations were used:

- Architecture (64, 32): This architecture, consisting of two layers with 64 and 32 neurons respectively, was used in the initial training phases. Its purpose was to evaluate the behaviour of the model in the prediction problem with a relatively simple structure. This allowed to obtain a benchmark to compare more complex architectures.
- Architecture (128, 64, 32): By adding an intermediate layer and increasing the number of neurons, this architecture increases the model's ability to learn more complex patterns in the data. This configuration is expected to improve the generalization of the model by capturing deeper relationships between input variables.
- Architecture (256, 128, 64, 32): This deeper architecture with larger number of neurons is designed to address highly complex prediction problems. The addition of more layers and neurons provides the model with the ability to learn high-level feature representations, which can be crucial for improving accuracy in water stress prediction.

In all architectures, optimization techniques, such as Adam, RMSProp and SGD, were applied in order to minimize the loss function during training. Each layer was regularized using the L2 penalty (regularization term) to avoid overfitting. In addition, a Dropout layer with a rate of 30% was included after each hidden layer to improve the robustness of the model and avoid overdependence of some neurons.

19.3.2.3 Model optimization

SGD (Stochastic Gradient Descent) with a learning rate of 0.01 and a momentum de 0.9

The Stochastic Gradient Descent (SGD) methodology, as presented by LeCun et al. (2002), can be summarized in the following key steps in the context of neural network training:

1. Definition of the optimization problem: The objective is to minimize a cost function E(W), which measures the difference between the expected value and the output predicted by the model. The most commonly used cost function is the mean square error (MSE), which is defined as:

$$E(W) = \frac{1}{P} \overset{P}{\underset{p=1}{\quad}} E_P. \tag{19.2}$$

Where E_P is the error associated with pattern p, W is the vector of model parameters, and P is the size of the training set.

2. Gradient calculation: At each iteration, the gradient of the cost function with respect to the model parameters is calculated:

$$\nabla_W E(W). \tag{19.3}$$

   This gradient indicates the direction in which the W parameters should be adjusted to reduce the error.
3. Parameter update: The model parameters are updated using the gradient of the cost function computed in the previous step. In SGD, the update is performed for each p training pattern stochastically, i.e., individual samples are used instead of the entire data set:

$$w_{t+1} = w_t - \eta \nabla_w E_p(w). \tag{19.4}$$

   Where η is the learning rate, a hyperparameter that controls the step size taken at each iteration.
4. Repetition: The process of updating the parameters is repeated over several epochs, going through all the samples in the training set several times until the model converges or until a predefined number of iterations is reached.
5. Improving generalization: In addition to minimizing the cost function, the paper also addresses the importance of improving the generalization capability of the model, i.e., the ability of the model to make correct predictions on unseen data. The SGD method is combined with techniques such as regularization to avoid overfitting and improve model generalization.

Adam with a learning rate of 0.001

The Adam (Adaptive Moment Estimation) optimizer is used to update the model parameters during training. Adam combines the advantages of the AdaGrad and RMSProp algorithms, providing an adaptive learning rate adjustment for each model parameter. Its implementation is straightforward, requiring only first-order gradients, making it computationally efficient and suitable for problems with nonstationary targets and noisy or sparse gradients.

The algorithm employs adaptive estimates of the first and second moments of the gradients, automatically adjusting the step size during the optimization process. The standard hyperparameters used were initial learning rate $\alpha = 0.001$, $\beta = 0.999$, and $\varepsilon = 10\text{-}8$, which did not require additional adjustments. During the training process, the initial biases of the moments were corrected by a correction mechanism to ensure proper convergence.

This setup ensures that the model parameters converge efficiently, maintaining a balance between speed of convergence and numerical stability of the training [9].

Key formulas of the Adam algorithm:

1. Moving average of the gradient (first moment):

$$m_t = \beta_1 - m_{t-1} + (1 - \beta_1) \cdot g_t. \tag{19.5}$$

2. Moving average of the squares of the gradient (second moment):

$$v_t = \beta_2 - v_{t-1} + (1 - \beta_2) \cdot g_t^2. \tag{19.6}$$

3. Updating of parameters:

$$\theta_t = \theta_{t-1} - \frac{\alpha \cdot m_t}{\sqrt{v_t} + \varepsilon}. \tag{19.7}$$

RMSprop with a learning rate of 0.0001

RMSProp (Root Mean Square Propagation) is an adaptive optimization algorithm that adjusts the learning rate by dividing the gradient by a moving average of the magnitudes of recent gradients. This technique is useful for handling the oscillations problem in optimization and for improving convergence in deep learning problems, especially in neural networks trained with mini-batches [10].

1. Adaptive update: Unlike AdaGrad, which accumulates gradients indefinitely, RMSProp maintains an exponentially decreasing average of the squares of past gradients. The update of the weights is performed as follows:

$$x_{t+1} = x_t - \frac{\eta}{\sqrt{E\left[g^2\right]_t + \epsilon}} g_t. \tag{19.8}$$

   Where E[g2]t is the exponential moving average of the squared gradients at time t, and ϵ is a small value to avoid divisions by zero.

2. Exponential moving average: The key to RMSProp is that it calculates a moving average of the squares of the gradients with a decay factor. This allows the algorithm to remain effective even in problems where the gradients vary significantly over time. The calculation of this moving average is defined as:

$$E\left[g^2\right]_t = \gamma E\left[g^2\right]_{t-1} + (1 - \gamma) g_t^2. \tag{19.9}$$

   Where γ is the decay factor, usually close to 0.9.

Advantages of RMSProp: RMSProp solves the problem of excessive learning rate decay in AdaGrad by limiting the influence of old gradients through the use of a moving average. This allows maintaining an adequate learning rate throughout the training, especially in deep neural networks where gradients can vary considerably between layers. This method has proven to be particularly effective in optimization tasks with redundant data or features of very different magnitudes.

Adagrad with a learning rate of 0.01

AdaGrad is an adaptive optimization algorithm that adjusts the learning rate of each parameter based on the cumulative magnitude of gradients in previous iterations. This adjustment allows the algorithm to be effective in scenarios where features are sparse or gradients vary considerably between dimensions [11].

1. Adaptive update: At each iteration t, the parameters x$_t$ are updated using the gradient g$_t$, adjusted by a factor that depends on the cumulative sum of squared gradients:

$$x_{t+1} = x_t - \eta \frac{g_t}{\sqrt{G_t + \varepsilon}}. \tag{19.10}$$

   Where Gt is the sum of the squared gradients up to t, and ε is a small value to avoid divisions by zero.
2. Adaptive advantages: AdaGrad automatically adjusts the learning rate for each parameter. Parameters associated with large gradients are updated more slowly, while those with small gradients are adjusted more quickly, improving learning efficiency on sparse data.
3. Mahalanobis norm: AdaGrad uses the Mahalanobis norm to project the updates, which ensures that the update steps are adjusted to the behaviour of the accumulated gradients, resulting in more stable and efficient updates.

This adaptive approach allows AdaGrad to achieve better convergence, especially in problems with sparse data or where features have different scales of importance. To improve the convergence of the model, the following callbacks were implemented during EarlyStopping and ReduceLROnPlateau training.

19.3.2.4 Evaluation and metrics

The model was evaluated using the following metrics:

- RMSE (Root Mean Square Error): measures the standard deviation of the predictions with respect to the actual values.

- MAE (Mean Absolute Error): Measures the average of the absolute errors.
- R^2 Score: Evaluates how well the predictions fit the actual values (coefficient of determination).

These metrics were calculated for both the training and test sets.

19.4 Results

Table 19.3 presents an analysis of various neural network architectures and optimizers, evaluated using the mean square error (MSE), the mean absolute error (MAE) and the coefficient of determination (R^2). These indicators are key to measure the accuracy and generalization capability of the model. In the two-layer architecture (64 and 32 neurons), the Stochastic Gradient Descent (SGD) optimizer stands out with an MSE of 0.0320 in training and 0.0254 in testing, showing good generalization. In contrast, Adam underperforms, while RMSprop and Adagrad offer intermediate results. For the three-layer architecture, Adam achieves the best performance with an MSE of 0.0284 in training and 0.0218 in test, indicating excellent generalization ability. SGD and RMSprop are not as effective in this more complex configuration. In the four-layer architecture, SGD maintains competitive performance with an MSE of 0.0445 in training and 0.0375 in test. Adam fails to match its previous performance, and both Adagrad and RMSprop show higher errors. In summary, SGD is effective on simple architectures, while Adam excels

Table 19.3 Results of the different architectures and optimizers ⏎

	MSE train	**MAE train**	**R^2 train**	**MSE test**	**MAE test**	**R^2 test**
2L(64,32) SGD	0.032	0.1149	0.9689	0.0255	0.1066	0.9703
2L(64,32) Adam	0.077	0.2019	0.9253	0.0634	0.1695	0.9261
2L(64,32) RMS	0.0398	0.1512	0.9614	0.0306	0.1228	0.9644
2L(64,32) Ada	0.0388	0.1245	0.9624	0.0344	0.1191	0.9599
3L(128,64,32) SGD	0.1223	0.1827	0.8814	0.0959	0.1646	0.8883
3L(128,64,32) Adam	0.0284	0.1174	0.9724	0.0219	0.0982	0.9745
3L(128,64,32) RMS	0.1205	0.2486	0.8831	0.081	0.1945	0.9057
3L(128,64,32) Ada	0.0544	0.1416	0.9472	0.0446	0.1301	0.948
4L(256,128,64,32) SGD	0.0446	0.1444	0.9568	0.0375	0.1361	0.9563
4L(256,128,64,32) Adam	0.0615	0.1935	0.9404	0.0515	0.1741	0.94
4L(256,128,64,32) RMS	0.0772	0.2415	0.9252	0.0858	0.2574	0.9001
4L(256,128,64,32) Ada	0.0547	0.1764	0.9469	0.0429	0.1636	0.95

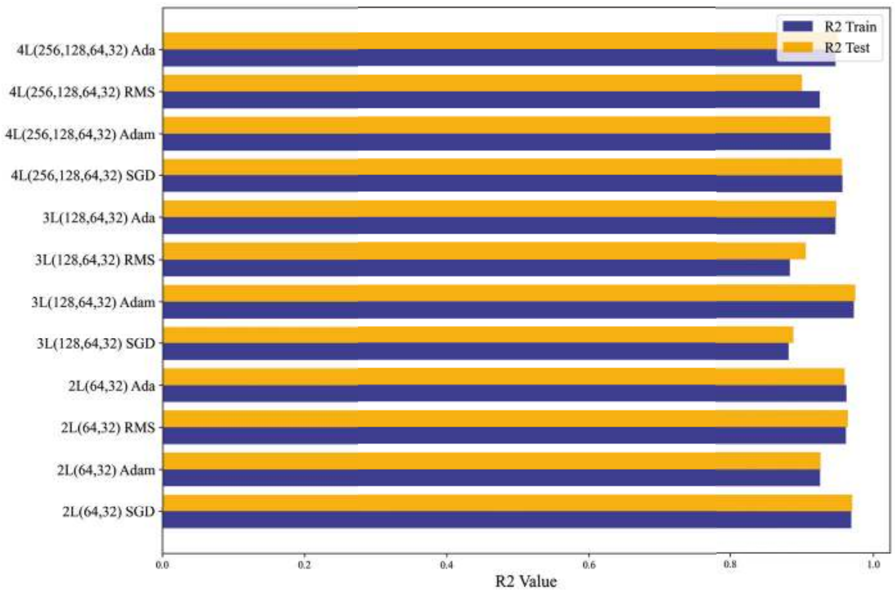


Figure 19.2 Comparison of R^2.

on more complex structures. The choice of optimizer should consider the complexity of the neural network to optimize performance.

Figure 19.3 presents the R^2 values for different combinations of model architectures and optimizers, evaluating their performance on the training and test sets. An R^2 close to 1 indicates a good ability to explain the variability of the data, and the results suggest that there is no overfitting, as the values are similar in both sets. The combination of 2L(64,32) with the SGD optimizer stands out with an R^2 of about 0.970 in both ensembles, showing excellent generalization. Likewise, the 3L(128,64,32) architecture with Adam achieves an R^2 of 0.975 on the test set, indicating optimal performance. However, more complex architectures, such as 4L(256,128,64,32), show slightly lower results, especially with RMSprop, which has an R^2 of 0.900 on test. This suggests that, despite their ability to capture complexities, they do not always improve performance. An important finding is that Adam offers superior performance on complex architectures, while SGD is very competitive on simpler configurations. In summary, simpler models with well-tuned optimizers achieve good results in R^2, and the generalization capability is robust, indicating stability in predicting unseen data.

Figure 4 compares the prediction errors of different neural network models using two metrics: MSE (Mean Squared Error) and MAE (Mean

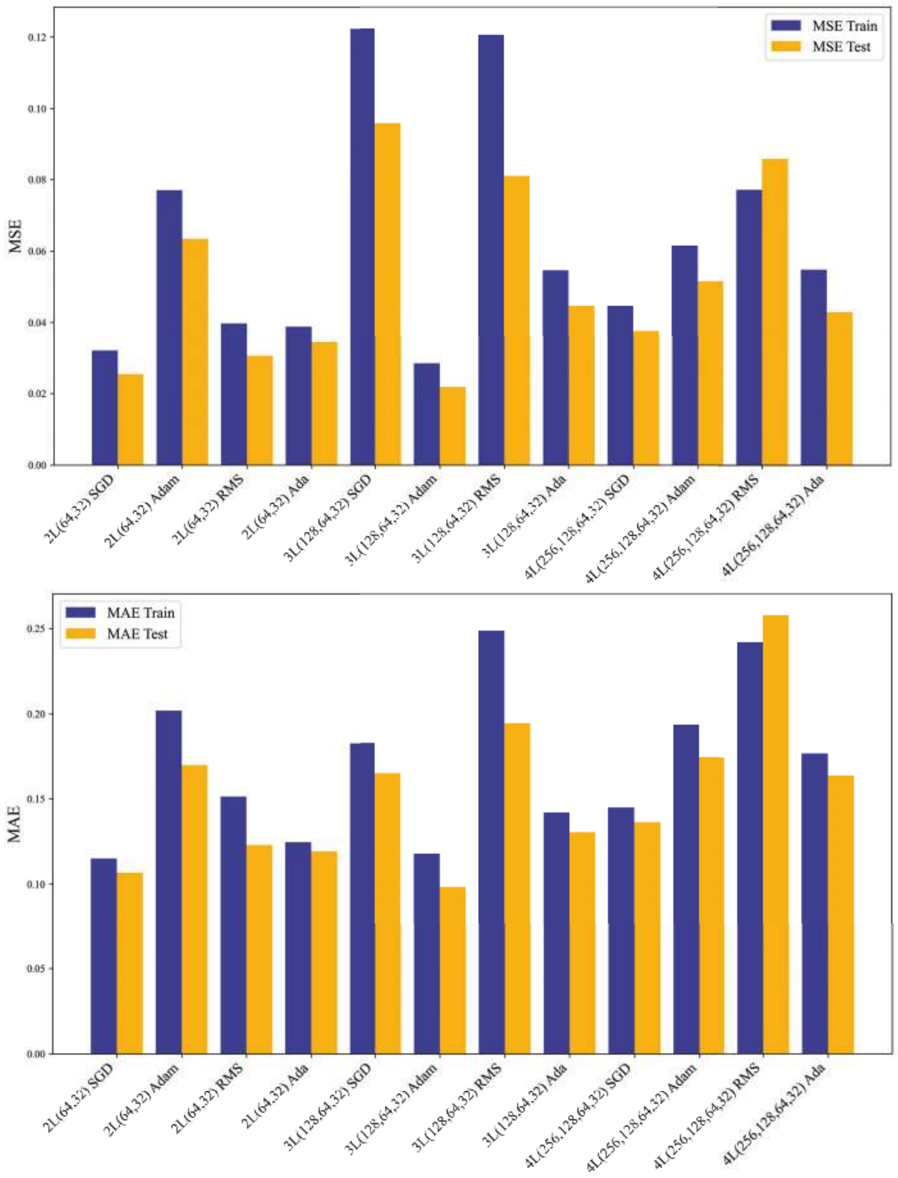


Figure 19.3 Comparison of MSE and MAE. ⏎

Absolute Error), for the training and test sets. In the MSE plot, models with optimizers such as Adam and SGD show better results, with the 2L(64,32) model with SGD achieving the lowest MSE, indicating a good fit and balanced prediction ability in both sets. However, more complex models such as 3L(128,64,32) with SGD and 4L(256,128,64,32) with RMS present higher MSE, suggesting difficulties in generalizing. The MAE plot reflects similar patterns, where 2L(64,32) with Adam shows lower absolute error, indicating good accuracy. In contrast, models such as 4L(256,128,64,64,32) with RMS and 3L(128,64,32) with SGD show higher errors, implying optimization problems. In general, simpler architectures, especially 2L(64,32) with optimizers such as SGD or Adam, tend to perform better on both metrics. More complex architectures, such as 4L(256,128,64,32), show worse results, with larger discrepancies between training and test performance, which may indicate overfitting or suboptimal optimization.

19.5 Conclusions

The conclusions of this study highlight key implications for policy adoption and future research design in the field of deep learning. The findings reinforce the need for clear guidelines on the selection of architectures and optimizers in neural networks, not only from a technical perspective, but also to facilitate efficient implementation in practical applications. Simplicity, as observed in the performance of less complex architectures such as 2L(64,32), is a principle that can guide strategic decisions in the development of predictive models, suggesting that in many contexts it is preferable to opt for less complex solutions that balance accuracy and resource efficiency. This approach could be crucial for industries seeking to integrate artificial intelligence into their processes, as it minimizes the risks associated with computational overhead and overfitting.

As for optimizers, the results reinforce the adoption of approaches such as SGD and Adam, which proved to be the most effective depending on the complexity of the architecture. The superior performance of SGD on simple architectures and Adam on more complex models suggests that model development policies should be adaptive, adjusting both the architecture and the optimizer according to the context and specific data needs.

For future research, these results open multiple avenues of exploration. It is essential to continue to evaluate how different datasets, particularly those with more complex nonlinear relationships or larger volumes, affect the performance of more complex architectures. In addition, the impact of hyperparameters should be further investigated and alternative optimizers that have not yet been widely studied in this context, such as AdamW or Nadam, should be tested to confirm the validity of these patterns in different scenarios.

Finally, the transfer of this knowledge to the applied domain, whether in AI policies for industrial sectors or to improve technology adoption in the public sector, requires an evidence-based approach such as the one presented here. The promotion of policies that incentivize simplicity and efficient optimization in model development could have a significant impact, encouraging a more sustainable and effective use of artificial intelligence. The implementation of policies that promote simplicity and optimization in the development of AI models can be key to accelerating their adoption in both industrial and public sectors. The study highlights that simple architectures and efficient optimizers such as SGD or Adam not only improve performance, but also facilitate the scalability and maintainability of technology solutions. In the industrial domain, this could reduce costs and barriers to entry, enabling a democratization of AI. In the public sector, the use of accessible models can contribute to scalable and sustainable solutions, such as predictive health systems and urban resource management. In addition, promoting simplicity in AI design is also aligned with green objectives, reducing energy consumption associated with complex models.

Acknowledgements

This work has been partially funded by the EU Project dAIEDGE (GA Nr 101120726).

References

[1] P. Quevauviller, "Adapting to climate change: reducing water-related risks in Europe–EU policy and research considerations," Environmental Science & Policy, vol. 14, no. 7, pp. 722-729, 2011. https://doi.org/10.1016/j.envsci.2011.02.008

[2] P. Falloon and R. Betts, "Climate impacts on European agriculture and water management in the context of adaptation and mitigation—the importance of an integrated approach," Science of the Total Environment, vol. 408, no. 23, pp. 5667-5687, 2010. https://doi.org/10.1016/j.scitotenv.2009.05.002

[3] S. Lavrnić, M. Zapater-Pereyra, and M. L. Mancini, "Water scarcity and wastewater reuse standards in Southern Europe: focus on agriculture," *Water, Air, & Soil Pollution*, vol. 228, pp. 1-12, 2017. https://doi.org/10.1007/s11270-017-3425-2

[4] P. Behrens, M. T. Van Vliet, T. Nanninga, B. Walsh, and J. F. Rodrigues, "Climate change and the vulnerability of electricity generation to water stress in the European Union," *Nature Energy*, vol. 2, no. 8, pp. 1-7, Aug. 2017. https://doi.org/10.1038/nenergy.2017.114

[5] M. Weiß and J. Alcamo, "A systematic approach to assessing the sensitivity and vulnerability of water availability to climate change in Europe," *Water Resour. Res.*, vol. 47, W02549, 2011, doi:10.1029/2009 WR008516.

[6] A. Gorlapalli et al., "Characterization and Prediction of Water Stress Using Time Series and Artificial Intelligence Models," *Sustainability*, vol. 14, no. 11, pp. 6690, 2022. https://doi.org/10.3390/su14116690

[7] C. Pradawet et al., "Thermal imaging for assessment of maize water stress and yield prediction under drought conditions," *Journal of Agronomy and Crop Science*, vol. 209, pp. 56–70, 2023. https://doi.org/10.3 390/su14116690

[8] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade*, 2nd ed., Berlin, Germany: Springer, 2002, pp. 9-50. https://doi.org/10.1007/3-540-49430-8_2

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," presented at the *Int. Conf. on Learning Representations (ICLR)*, 2014.

[10] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning: Overview of mini-batch gradient descent," presented at the *Univ. of Toronto*, 2012.

[11] J. K. Fang et al., "Adagrad gradient descent method for AI image management," in *Proc. 2020 IEEE Int. Conf. on Consumer Electronics-Taiwan (ICCE-Taiwan)*, 2020, pp. 1-2, https://doi.org/10.1109/ICCE-T aiwan49838.2020.9258085

[12] Food and Agriculture Organization of the United Nations FAO Terminology Portal. https://www.fao.org/faoterm/viewentry/en/?entryId=172 346.

20

The Accountability Strikes Back: Decentralizing the Key Generation in CL-PKC with Traceable Ring Signatures

Varesh Mishra, Aysajan Abidin, and Bart Preneel

COSIC, KU Leuven, Belgium

Abstract

In the context of Federated Learning, it is essential to authenticate the incoming messages from various entities. Many works have utilized Certificateless Public Key Cryptography (CL-PKC) and Signature Schemes. These recent developments focus on the existence of a single Trusted Authority (TA) that can issue partial private signatures truthfully without any malicious intent. This assumption is not pragmatic when multiple competing entities with some assets are involved. Additionally, dependence on a single TA introduces a single point of failure and a center of malice. In this work, we first propose a mechanism for partial key exchange for CL-PKC employing Traceable Ring Signatures. Furthermore, we utilize the existing blockchain or logging infrastructure to extend our model to provide accountability and disincentivization for malicious TAs. The cryptographic tools used in this work can be parallelized to get more efficiency out of these trusted nodes. To evaluate our protocol, we also simulate it to argue for the communication and computation costs.

Keywords: certificateless public key cryptography, distributed learning, accountability.

DOI: 10.1201/9788743808862-20

20.1 Introduction

Federated Learning (FL) is a distinct approach to machine learning in which distributed and decentralized entities train a local model using the localized data and provide updates to the global server to improve the global model. There are many inherent benefits to such an approach as it avoids the unnecessary sharing of raw data from one point to another. It also has numerous advantages in terms of scalability and efficiency. However, this approach generates a complex network structure wherein there is a huge amount of data aggregation to a local node, usually called the aggregator, where local data processing occurs. Due to the complexity of the underlying network structure, enabling key distribution and management is a challenging task. The key management and distribution are essential to implement the necessary data security guarantees. Moreover, it is equally important to handle access control and data authentication from various entities in the network. Without the necessary steps, malicious entities can exploit the vulnerabilities to inflict loss of data and assets. Thus, distributing keys and authenticate various messages from the network entities is of utmost importance. A critical aspect of such networks is that the end devices that collect the data are usually resource-constrained with limited computation power. Therefore, we cannot deploy expensive cryptographic tools for data security.

20.2 Related Work and Contributions

Identity-based Encryption (IBE) [1] [2] allows users to communicate and share secrets and signatures in which the user's public key is derived from a known string associated with the user. However, this requires complete trust in a Trusted Authority (TA), also known as Key Generation Centre (KGC). This introduces a single point of failure that can reveal all the secret keys associated with the system users when compromised. Certificateless Public Key Cryptography (CL-PKC) [3] allows the system users to derive keys using the trusted KGC without the key-escrow problem in IBE. In CL-PKC, the users use KGC to get partial private keys, which they can use to generate their full private keys, reducing the role of KGC. This scheme does not require certificates, but a trust in the KGC is still required. Other issues include securely transmitting the partial private key to the user who requests it. Hierarchical Certificateless Cryptography (HCLC) [4] [5] aims to solve the concerns regarding trust in KGC by introducing a tree-like structure involving a root KGC, lower-level KGCs, and the users. The structure dictated by

HCLC has root KGC at the top, the users as the leaf nodes, and the lower-level KGCs as the intermediate nodes. This model reduces trust in the root KGC but requires a complex setup procedure and has scalability issues when the hierarchy grows. There are concerns regarding communication costs and the security of higher-level KGCs. However, CL-PKC is very efficient and computationally inexpensive for resource-constrained devices. Many recent works propose using CL-PKC in multiple domains, such as Wireless Body Area Networks, VANET authentication, and Federated Learning with privacy and anonymity as a feature [6], [7], [8], [9]. However, all these works rely on a single KGC or TA. In this work, we introduce the usage of multiple and distributed TAs for the full private key generation. The proposed protocol also provides accountability for the key generation process using Traceable Ring Signatures. The TA trying to generate an inconsistent or another key using the partial private key (key replacement) can be disincentivized. Another benefit of using a Traceable Ring Signature is that when the key is successfully generated, the user requesting the key and the TA generating the key only know about the key generation process, introducing uncertainty regarding where it was generated, provided appropriate measures are taken to hide the network traffic. The other participants may become aware that the key has been generated for a user who desires it in specific applications. The user must, however, provide certain publicly known parameters related to TA for consistent encryption, which can be predetermined between the parties if required.

20.3 Preliminaries

20.3.1 Pairing Based Cryptography

Consider an additive group $(G_{\mathbb{1}},\ +)$ and a multiplicative group $(G_{\mathbb{2}}, .)$, both finite groups of prime order q. Let us denote P as a generator of $G_{\mathbb{1}}$. A pairing e is a map $e\ :\ G_{\mathbb{1}} \times\ G_{\mathbb{1}} \to\ G_{\mathbb{2}}$ such that the following properties hold true:

- The map e is bilinear: given $A,\ B,\ C\ \in\ G_1$, we have

$$e(A,\ B\ +\ C)\ =\ e(A,\ B)\ .e(A,\ C),$$

$$e(A\ +\ B,\ C)\ =\ e(A,\ C)\ .e(B,\ C).$$

  Additionally, for any $x, y \in Z_q^*$, we have $e(xA,\ \ yB)\ =\ e(A,\ B)xy\ = e(xyA,\ B)$ etc.
- The map e is non-degenerate: $e\,(P,\ P\ ) \neq\ 1.$

- There exists an efficient algorithm such that computation of map e is efficient.

20.3.2 Certificateless Public Key Cryptography (CL-PKC)

CL-PKC is a public key cryptosystem that overcomes the limitations of traditional Public Key Infrastructure and Identity-Based Cryptography (IBC). The main objective of CL-PKC is the elimination of digital certificates (as required in traditional PKI) and the inherent key-escrow problem where the TA possesses the knowledge of the private keys of all the entities. In CL-PKC, a TA or KGC facilitates an entity in the private key generation by generating a partial private key with its secret value. The entity computes its public key using its secret value. It is interesting to observe that an entity can generate its public key before generating the private key.

20.3.3 Traceable Ring Signatures

A ring signature allows the signer to sign a message on behalf of a group of signers and the ability to remain anonymous [10]. Thus, with a ring signature, an entity can ensure that the message was signed by one of the group members but cannot identify the signer amongst the group. Blockchains such as Monero use ring signatures for anonymous transactions [11]. However, in specific applications such as e-voting, total anonymity can help certain entities cast double votes without repercussion. Traceable Ring Signature (TRS) [12] provides functionality to trace a signer's public key in case the signer issues a signature for two messages using the same tag. The tag usually consists of an $issue$ and a ring of public keys pk_{ring}. The $issue$ reflects the context in which a particular vote is to be cast. Let $\lambda \in N$ be a security parameter which denotes the desired level of security. We can define Traceable Ring Signatures in technical terms as follows. A Traceable Ring Signature scheme consists of algorithms $< Gen, \ Sign, \ Verify, \ Trace >$ such that:

- Gen: is a probabilistic polynomial-time algorithm that takes a security parameter $\lambda \in N$ as input and outputs a public and secret key pair $(pk, \ sk)$.
- $Sign$: is a probabilistic polynomial-time algorithm that takes a secret key, sk_i, where $i \in N$, tag $\mathcal{L} = (issue, \ pk_{ring})$, and message $m \in \{0, 1\}^*$ as input and outputs a signature σ.
- $Verify$: is a deterministic polynomial-time algorithm that takes tag $\mathcal{L} = (issue, \ pk_{\text{ring}})$, message $m \in \{0, 1\}^*$, and signature σ as input and outputs a bit indicating the validity of the signature.

- $Trace$: is a deterministic polynomial-time algorithm that takes tag $\mathcal{L} = (issue,\ pk_{\text{ring}})$, and two message-signature pairs, (m, σ), $\left(m', \sigma'\right)$ as input and outputs string result$\in\{\text{indep},\ \text{linked},\ pk\}$, where $pk \in pk_{ring}$ subject to the conditions implied by public traceability as defined in [12].

20.3.4 Merkle Patricia Trie

The Merkle Patricia Trie (MPT) is a distributed data structure that maintains a consistent and efficient key-value database [13]. It combines Merkle Tree [14] and Patricia Trie [15]. The Merkle Trees guarantee data integrity and verification using cryptographic hashing, while the Patricia Tries have an inherent structure to support efficient storage and retrieval properties. The data can be accessed through the node path traversal. A generic MPT structure has three node types: leaf, branch, and extension. One of the main features of MPT is the easy verification of state changes. The root of the MPT can be used to ensure that a particular value is consistent throughout the various distributed instances of a key-value database. This property is supported by the standard security guarantees of a cryptographic hash function [16]. Currently, MPT is employed in various blockchains such as Ethereum, Quorum, and many more, for storage tracking state changes of blockchain global state (in the form of a state trie), transactional data, and similar associated data [13].

20.4 Proposed Model

20.4.1 Notation

The notation throughout the text follows an indexed superscript for a parameter to form an association with network participants. In the subsequent sections, we will use TA to denote a KGC throughout our discussion. The notation $E^{(i)}$ and $TA^{(j)}$ denote Entity with index i and TA with index j, respectively. $Enc_s\,(m,\ k)$ and $Dec_s\,(c,\ k)$ denote symmetric key encryption of message m with key k and symmetric key decryption of ciphertext c with key k respectively. The state trie is denoted by $\mathcal{ST}$. The existence of an element e in the $\mathcal{ST}$ is denoted by $e \in \mathcal{ST}$, and consequently, the non-existence is denoted by $e \notin \mathcal{ST}$. The inclusion of an element e within $\mathcal{ST}$ is denoted by $\mathcal{ST} \leftarrow \mathcal{ST} \bigcup \{e\}$. Every network participant can query $\mathcal{ST}$ and can check for the existence of e in $\mathcal{O}\,(1)$ time through a query to a subset of validators. The validators or authorized participants can also include an element within the state trie through consensus. The validators also have an internal mechanism $Disincentivize\,(\mathcal{S})$, which takes a set $\mathcal{S}$, and disincentivizes the public keys in the set $\mathcal{S}$ through consensus amongst the validators. The messages within the network participants are denoted as $\langle type, param1, param2 \ldots \rangle$ where the first parameter is always message type. The usage of other parameters will be highlighted in the respective algorithms. The methods $broadcast\,(msg)$ and $send\,(dest, msg)$ are used for message transmission. The broadcast sends the message msg to every validator in the network while the method $send$, sends the message msg to a network participant denoted by $dest$. For instance, $send\ TA^{(j)},\ \langle \text{ init } \rangle$ denotes a message with type $init$ to $TA^{(j)}$. The source of this message depends on the context of other parameters or where it is mentioned in the text. The process of receiving a message is denoted by $on\!-\!recv\,(< message >)$. The operations regarding Traceable Ring Signatures are denoted by a subscript trs. The operations $Gen_{trs}(\lambda)$, $Sign_{\,trs}(sk_{trs}^{(i)},\ \mathcal{L},\ m)$, $Verify_{trs}(\mathcal{L},\ m, \sigma)$, $Trace_{trs}\left(\mathcal{L},\ (m, \sigma), (m', \sigma')\right)$ are defined as in Subsection 1.3.3.

There exists a method called $lookup\left(P_0^{(j)}\right)$ which takes $P_0^{(j)}$ associated with $TA^{(j)}$ and returns corresponding $pk_{trs}^{(j)}$. The lookup table is built up during the network bootstrapping, as discussed in Subsection 1.4.3.1. The parameters associated with the network participants are mentioned in Table 20.1.

20.4.2 Network Architecture

The network architecture is assumed to take the form depicted in Figure 20.1. Two major network participants are Trusted Authorities (TA) and Entities (E). Formally we can state that the set of TAs is denoted by $TA = \{TA^{(1)}, TA^{(2)}, \ldots, TA^{(w)}\}$ and set of entities is denoted by $E = \{E^{(1)},\ E^{(2)}, \ldots, E^{(x)}\}$. It is also assumed that $|TA| = w$ and $|E| = x$. The network participants have communication links between each other where they can exchange data. The role of validators mentioned in Subsection 1.4 is delegated to the network participants of the set TA. It is also assumed that the majority of the validators are honest. Therefore, the network participants in set TA have the responsibility of updating and maintaining $\mathcal{ST}$ and executing $Disincentivize\,(\mathcal{S})$ consistently. $\mathcal{ST}$ is assumed to remain consistent for all operations. All the network participants can query $\mathcal{ST}$

Table 20.1 Notations. ⏎

Notation	Explanation
$TA^{(j)}$	trusted authority (TA) with index j
$E^{(i)}$	entity (E) with index i
$ID^{(i)}$	identifier for $E^{(i)}$
Z_q^*	the multiplicative group of integers modulo prime q
$s^{(i)}$	secret value of $E^{(i)}$
$s_{ta}^{(j)}$	secret value of $TA^{(j)}$
$P_0^{(j)}$	public parameter P_0 for $TA^{(j)}$
$Q^{(i)}$	intermediate value generated during partial key generation for entity $E^{(i)}$
$D^{(i)}$	partial private key for $E^{(i)}$
${D_{enc}}^{(i)}$	encrypted value of $D^{(i)}$
N	set of natural numbers
λ	a security parameter for desired security guarantees
$\sigma^{(i,j)}$	the TRS signature signed by $TA^{(j)}$ for key generation request from $E^{(i)}$
pk_{ring}	ring associated with TRS consisting of a fixed set of public keys
$\mathcal{L}^{(i)}$	tag value associated with the key generation for $E^{(i)}$

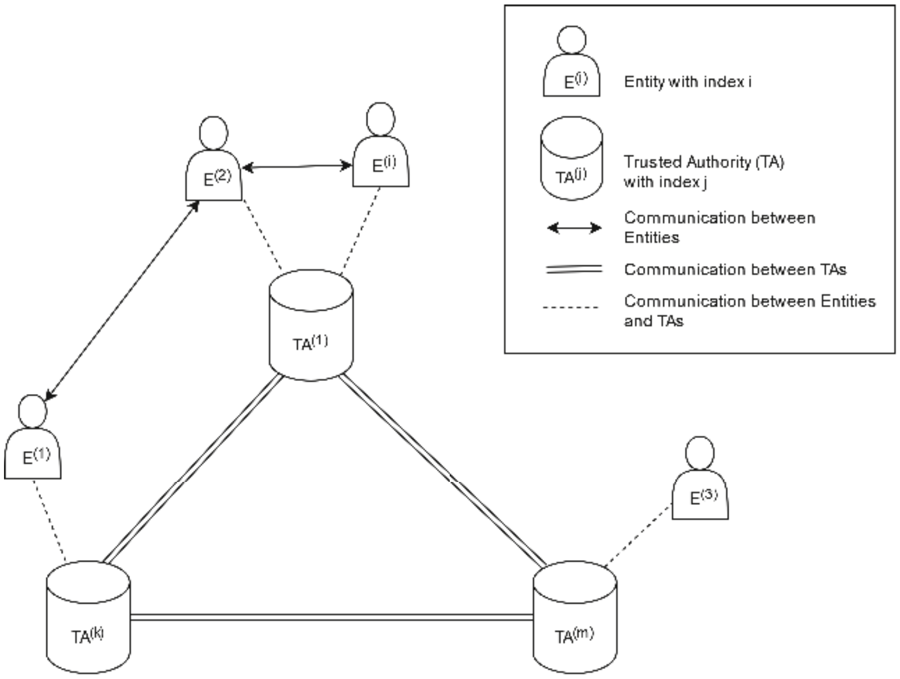

E(i)
Entity with index i
TA(j)
Trusted Authority (TA) with index j
Communication between Entities
Communication between TAs
Communication between Entities and TAs
E(2)
E(i)
TA(1)
E(1)
E(3)
TA(k)
TA(m)

Figure 20.1 The underlying architecture consists of TA nodes and Entities requesting partial private key generation. In this model, any Entity node can start the protocol with a TA node of its choice. ⏎

in constant time by issuing a request to one or multiple $\mathcal{ST}$ maintainers. The network is assumed to be synchronous, and Δ_s bounds the network packet delivery time. The maximum time for the inclusion of an element in the $\mathcal{ST}$ and maximum processing time is bounded by Δ_a and Δ_p, respectively.

20.4.3 Protocol

This section discusses the mechanism for partial private and full private key generation. Firstly, we discuss the process of network bootstrapping and the protocol for private key generation between a TA and an Entity. In further discussion, we introduce algorithms for updating $\mathcal{ST}$, the $Audit$ Algorithm for enforcing accountability, and finally, the $collectAndTrace$ Algorithm, which is used in the detection of malicious TA (or TAs).

20.4.3.1 Network Bootstrapping

The process of network bootstrapping is presented in Figure 20.2. The TAs in the set TA all have the reference to public parameters $pp :=< G_{\mathbb{1}}, G_2, e, P, \mathcal{H}_1, \mathcal{H}_2 >$. $G_{\mathbb{1}}$ is an additive group and G_2 is a multiplicative group. Both $G_{\mathbb{1}}$ and G_2 are of prime order q. P is a generator of $G_{\mathbb{1}}$. The pairing e is an efficient Billinear Pairing such that $e : G_{\mathbb{1}} \times G_{\mathbb{1}} \rightarrow G_2$. The functions $\mathcal{H}_1$ and $\mathcal{H}_2$ are cryptographic hash functions such that $\mathcal{H}_1 : \{0,1\}^* \rightarrow G_{\mathbb{1}}$ and $\mathcal{H}_2 : \{0,1\}^* \rightarrow \{0,1\}^n$. It is important to note

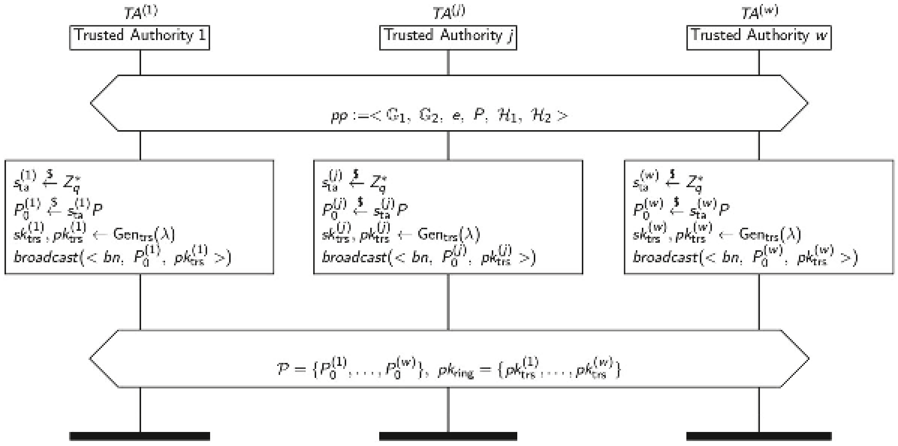

$TA^{(1)}$
Trusted Authority 1
$TA^{(j)}$
Trusted Authority j
$TA^{(w)}$
Trusted Authority w
$pp :=< G_1, G_2, e, P, \mathcal{H}_1, \mathcal{H}_2 >$
$s_{ta}^{(1)} \xleftarrow{\$} Z_q^*$
$P_0^{(1)} \xleftarrow{\$} s_{ta}^{(1)} P$
$sk_{trs}^{(1)}, pk_{trs}^{(1)} \leftarrow \mathrm{Gen}_{trs}(\lambda)$
$broadcast(< bn, P_0^{(1)}, pk_{trs}^{(1)} >)$
$s_{ta}^{(j)} \xleftarrow{\$} Z_q^*$
$P_0^{(j)} \xleftarrow{\$} s_{ta}^{(j)} P$
$sk_{trs}^{(j)}, pk_{trs}^{(j)} \leftarrow \mathrm{Gen}_{trs}(\lambda)$
$broadcast(< bn, P_0^{(j)}, pk_{trs}^{(j)} >)$
$s_{ta}^{(w)} \xleftarrow{\$} Z_q^*$
$P_0^{(w)} \xleftarrow{\$} s_{ta}^{(w)} P$
$sk_{trs}^{(w)}, pk_{trs}^{(w)} \leftarrow \mathrm{Gen}_{trs}(\lambda)$
$broadcast(< bn, P_0^{(w)}, pk_{trs}^{(w)} >)$
$\mathcal{P} = \{P_0^{(1)}, \ldots, P_0^{(w)}\}, pk_{ring} = \{pk_{trs}^{(1)}, \ldots, pk_{trs}^{(w)}\}$

Figure 20.2 Network Bootstrapping. ⏎

that these hash functions are cryptographically secure with standard security guarantees.

Each $TA^{(j)}$ randomly samples its secret value $s_{ta}^{(j)}$ from Z_q^*. Then, it calculates its public key for the CL-PKC key generation mechanism $P_0^{(j)}$. After generating $P_0^{(j)}$, $TA^{(j)}$ runs $Gen_{trs}(\lambda)$ to obtain $sk_{trs}^{(j)}$ and $pk_{trs}^{(j)}$, which are its private and public keys for generating Traceable Ring Signatures. The node then broadcasts $P_0^{(j)}$ and $pk_{trs}^{(j)}$ with message type bn to all the other TAs in the set TA. Finally, all the TAs obtain set $\mathcal{P} = \{P_0^{(1)}, \ldots, P_0^{(w)}\}$ and the ring $pk_{ring} = \{pk_{trs}^{(1)}, \ldots, pk_{trs}^{(w)}\}$.

20.4.3.2 Key Generation

The flowchart in Figure 20.3 illustrates the logical sequence of the key generation process along with updating $\mathcal{ST}$ and executing of audit mechanism. The key generation procedure is illustrated in Figure 20.4. The private key generation is a procedure between two network participants, namely, $E^{(i)}$ with $ID^{(i)}$ and $TA^{(j)}$. Firstly, $E^{(i)}$ samples its secret value $s^{(i)}$ from Z_q^* and also generates the associated set of public keys $X^{(i)} \leftarrow s^{(i)} P_0^{(j)}$ and $Y^{(i)} \leftarrow s^{(i)} P$. After the generation of public keys $E^{(i)}$, sends an *init* message containing $ID^{(i)}$, $X^{(i)}$ and $Y^{(i)}$. $TA^{(j)}$ on reception of *init* message starts

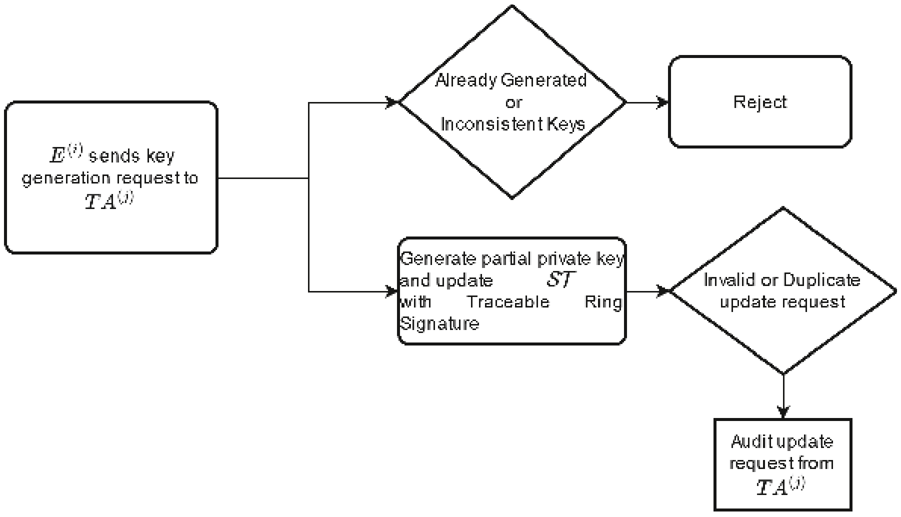

$E^{(i)}$ sends key generation request to $TA^{(j)}$
Already Generated or Inconsistent Keys
Reject
Generate partial private key and update $\mathcal{ST}$ with Traceable Ring Signature
Invalid or Duplicate update request
Audit update request from $TA^{(j)}$

Figure 20.3 Flowchart for the full key generation and audit procedure. ⏎

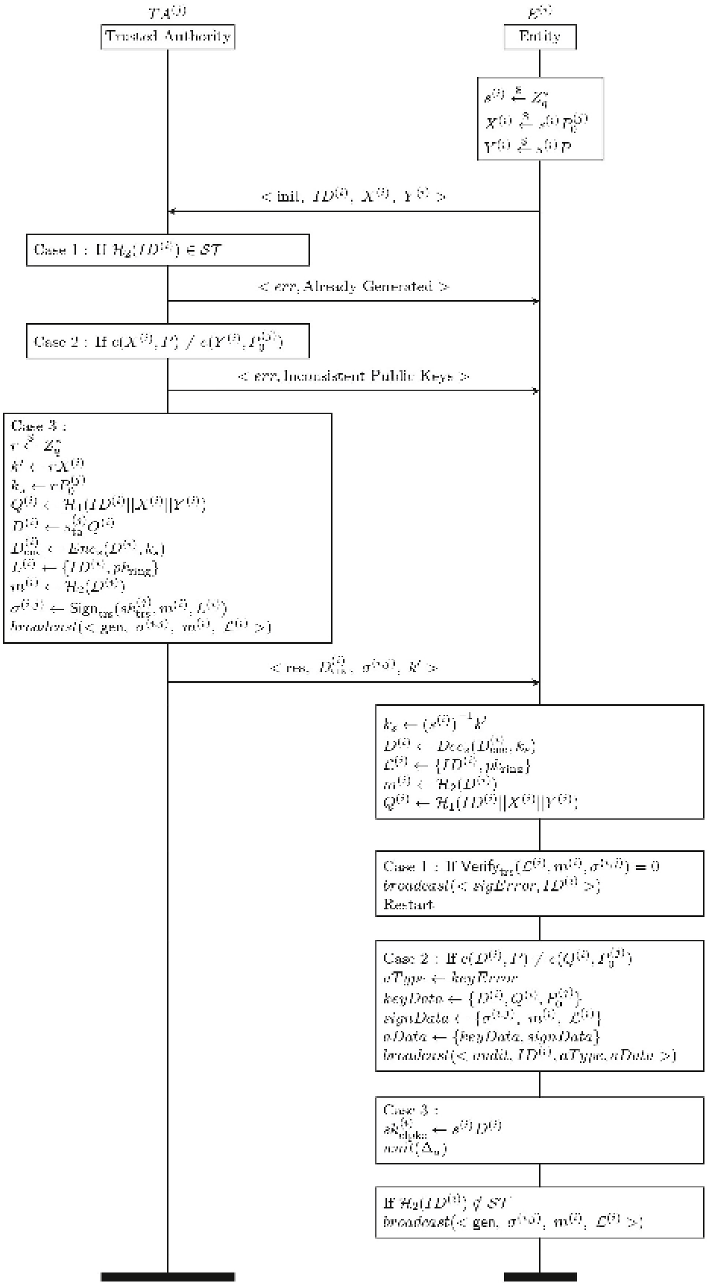

$TA^{(j)}$
Trusted Authority
$E^{(i)}$
Entity
$s^{(i)} \xleftarrow{\$} Z_q^*$
$X^{(i)} \xleftarrow{\$} s^{(i)} P_0^{(j)}$
$Y^{(i)} \xleftarrow{\$} s^{(i)} P$
$< \text{init},\ ID^{(i)},\ X^{(i)},\ Y^{(i)} >$
Case 1 : If $\mathcal{H}_2(ID^{(i)}) \in ST$
$< err$, Already Generated $>$
Case 2 : If $e(X^{(i)}, P) \neq e(Y^{(i)}, P_0^{(j)})$
$< err$, Inconsistent Public Keys $>$
Case 3 :
$r \xleftarrow{\$} Z_q^*$
$K' \leftarrow rX^{(i)}$
$k_s \leftarrow rP_0^{(j)}$
$Q^{(i)} \leftarrow \mathcal{H}_1(ID^{(i)}||X^{(i)}||Y^{(i)})$
$D^{(i)} \leftarrow s_{\text{ta}}^{(j)} Q^{(i)}$
$D_{\text{enc}}^{(i)} \leftarrow Enc_s(D^{(i)}, k_s)$
$L^{(i)} \leftarrow \{ID^{(i)}, pk_{\text{ring}}\}$
$m^{(i)} \leftarrow \mathcal{H}_2(D^{(i)})$
$\sigma^{(i,j)} \leftarrow \text{Sign}_{\text{trs}}(sk_{\text{trs}}^{(j)}, m^{(i)}, L^{(i)})$
$broadcast(< \text{gen},\ \sigma^{(i,j)},\ m^{(i)},\ L^{(i)} >)$
$< \text{res},\ D_{\text{enc}}^{(i)},\ \sigma^{(i,j)},\ K' >$
$k_s \leftarrow (s^{(i)})^{-1} K'$
$D^{(i)} \leftarrow Dec_s(D_{\text{enc}}^{(i)}, k_s)$
$L^{(i)} \leftarrow \{ID^{(i)}, pk_{\text{ring}}\}$
$m^{(i)} \leftarrow \mathcal{H}_2(D^{(i)})$
$Q^{(i)} \leftarrow \mathcal{H}_1(ID^{(i)}||X^{(i)}||Y^{(i)})$
Case 1 : If $\text{Verify}_{\text{trs}}(L^{(i)}, m^{(i)}, \sigma^{(i,j)}) = 0$
$broadcast(< sigError, ID^{(i)} >)$
Restart
Case 2 : If $e(D^{(i)}, P) \neq e(Q^{(i)}, P_0^{(j)})$
$aType \leftarrow keyError$
$keyData \leftarrow \{D^{(i)}, Q^{(i)}, P_0^{(j)}\}$
$signData \leftarrow \{\sigma^{(i,j)},\ m^{(i)},\ L^{(i)}\}$
$aData \leftarrow \{keyData, signData\}$
$broadcast(< audit, ID^{(i)}, aType, aData >)$
Case 3 :
$sk_{\text{clpke}}^{(i)} \leftarrow s^{(i)} D^{(i)}$
$wait(\Delta_a)$
If $\mathcal{H}_2(ID^{(i)}) \notin ST$
$broadcast(< \text{gen},\ \sigma^{(i,j)},\ m^{(i)},\ L^{(i)} >)$

Figure 20.4 Key generation protocol. ⏎

the generation of partial private key $D^{(i)}$. Three scenarios can occur. Note that each case represents a stage in the process. Each case is a check which must be followed to reach the next one.

- **Case 1**: $D^{(i)}$ has already been generated for $ID^{(i)}$ as there exists an entry for $ID^{(i)}$ in the $\mathcal{ST}$. In this case, $TA^{(j)}$ aborts the partial key generation process and sends a message with type err stating that the key has already been generated.
- **Case 2**: If $e\ \ X^{(i)}, P\ \ = e\left(Y^{(i)}, P_0^{(j)}\right)$, it implies that the public keys generated are inconsistent or $P_0^{(j)}$was not used in for public key generation.
- **Case 3**: In this case, the public keys $X^{(i)}$and $Y^{(i)}$ are consistent and $TA^{(j)}$ can begin the partial private key generation process. It is important to note that the $D^{(i)}$ must be transmitted over an insecure channel. Therefore, it must remain confidential between $TA^{(j)}$and $E^{(i)}$. To this end, $TA^{(j)}$ samples r randomly from Z_q^* and calculates $k' \leftarrow rX^{(i)}$. k' is the masked key for symmetric key encryption. Now, $TA^{(j)}$ calculates the actual key k_s for symmetric encryption by calculating $k_s \leftarrow rP_0^{(j)}$. In order to generate $D^{(i)}$, intermediate value $Q^{(i)}$ must be calculated as $Q^{(i)} \leftarrow \mathcal{H}_1(ID^{(i)} \parallel X^{(i)} \parallel Y^{(i)})$. $D^{(i)}$ is calculated as $D^{(i)} \leftarrow s_{ta}^{(j)} Q^{(i)}$. Next, $TA^{(j)}$ encrypts $D^{(i)}$ with key k_s using symmetric key encryption to obtain $D_{enc}{}^{(i)}$. In the subsequent step, $TA^{(j)}$ signs message $m^{(i)} \leftarrow \mathcal{H}_2\ \ D^{(i)}$ and tag $\mathcal{L}^{(i)} \leftarrow \{ID^{(i)},\ \ pk_{ring}\}$ with $sk_{trs}^{(i)}$ to obtain $\sigma^{(i,j)}$. Finally, $TA^{(j)}$ broadcasts a message with type gen containing $< \sigma^{(i,j)}, m^{(i)}, \mathcal{L}^{(i)} >$ to every $TA \in TA$ (including itself) for updating $\mathcal{ST}$ and sends a message with type res to $E^{(i)}$ containing $< D_{enc}{}^{(i)},\ \sigma^{(i,j)},\ k'\ >$.

20.4.3.3 Full Private Key Generation

On the reception of message of type res from $TA^{(j)}$, $E^{(i)}$ first tries to decrypt $D_{enc}{}^{(i)}$ to obtain $D^{(i)}$. For this it first obtains k_s by calculating $k_s \leftarrow\ s^{(i)}\ \ ^{-1} k'$. Using key k_s, it obtains $D^{(i)}$ by performing symmetric decryption operation as $D^{(i)} \leftarrow Dec_s\left(D_{enc}^{(i)}, k_s\right)$. Using the knowledge of $ID^{(i)}$ and pk_{ring}, it recovers the tag $\mathcal{L}^{(i)}$ as $\mathcal{L}^{(i)} \leftarrow \{ID^{(i)}, pk_{ring}\}$. The $m^{(i)}$ for traceable ring signature verification can be calculated as $m^{(i)} \leftarrow \mathcal{H}_2\ \ D^{(i)}$. Then $E^{(i)}$ calculates $Q^{(i)}$ as

$Q^{(i)} \leftarrow \mathcal{H}_1(ID^{(i)} \mid\mid X^{(i)} \mid\mid Y^{(i)})$. From this point on, there are three scenarios. Here, as well, note that each case represents a stage in the process. Each case is a check which must be followed to reach the next one.

- Case 1: If $Verify_{trs}\ \ \mathcal{L}^{(i)}, m^{(i)}, \sigma^{(i,j)}\ \ = 0$, the entity $E^{(i)}$ broadcasts the message $< sigError, ID^{(i)} >$ to stop the inclusion of $ID^{(i)}$ in $\mathcal{ST}$. It then restarts the key generation process. It is critical to note that in this case, we do not try to disincentivize the $TA^{(j)}$ as it is trivial to construct an invalid traceable ring signature. This might occur due to an adversary trying to block communication links for $E^{(i)}$ and sending an invalid signature. It is unfair if we try to disincentize $TA^{(j)}$ in such a case. Therefore, in this case, $E^{(i)}$ must restart the key generation procedure with the same or any other TA.
- Case 2: If $e\ \ D^{(i)}, P\ \ = e\left(Q^{(i)}, P_0^{(j)}\right)$ holds, it implies that the $TA^{(j)}$ did not calculate the value of $D^{(i)}$ consistently. The message signer can be held accountable since the traceable ring signature is valid. To this end, $E^{(i)}$ prepares the audit parameters $aType,\ keyData,\ signData,\ aData$ as follows:

$$aType \leftarrow keyError$$

$$keyData \leftarrow \{D^{(i)}, Q^{(i)}, P_0^{(j)}\}$$

$$signData \leftarrow \{\sigma^{(i,j)}, m^{(i)}, L^{(i)}\}$$

$$aData \leftarrow \{keyData,\ signData\ \}$$

  Finally, it broadcasts the message with type audit containing $< ID^{(i)},\ aType,\ aData >$ to all $TA \in TA$. The audit procedure by TA is discussed in Algorithm 2.
- Case 3: Finally, the full private key $sk_{\text{clpkc}}^{(i)}$ can be calculated by $E^{(i)}$ as $sk_{\text{clpkc}}^{(i)} \leftarrow s^{(i)} D^{(i)}$. For completeness we specify that $E^{(i)}$broadcasts the message with type gen containing $< sigma^{(i,j)},\ m^{(i)},\ \mathcal{L}^{(i)}\ >$ to all the $TA \in TA$. However, this can be optimized by checking if after Δ_a time, If $\mathcal{H}_2\ \ ID^{(i)}\ \ \in \mathcal{ST}$ is true or not, if not, then broadcast the message for the updating $\mathcal{ST}$.

20.4.3.4 Update $\mathcal{ST}$

During the key generation procedure between $TA^{(j)}$ and $E^{(i)}$, a $TA \in TA$, say $TA^{(k)}$ can receive multiple messages. These can be messages of type gen, $sigError$, and audit. Updating $\mathcal{ST}$ concerns gen and $sigError$ message types. This process is described by Algorithm 1 . On reception of $< gen, \sigma^{(i,j)}, m^{(i)},\ \mathcal{L}^{(i)} >$ in reference to $\mathcal{L}^{(i)}, TA^{(k)}$ first verifies the traceable ring signature. If the verification fails, it notifies $E^{(i)}$ to restart partial key generation and returns $\perp$. If $ID^{(i)}$ for which registration is done, exists already in $\mathcal{ST}$, $TA^{(k)}$ prepares to start the audit procedure as discussed in Algorithm 2. For this, initializes audit Type parameter $aType$ as $aType \leftarrow dupRegError$. It accumulates the received signature data in $signData$ and further includes it in audit data represented by $aData$. It then broadcasts the message with type audit attaching data $< ID^{(i)},\ \ aType,\ \ aData >$. On the other hand, if $ID^{(i)} \notin \mathcal{ST}$, $TA^{(k)}$ waits for $2\Delta_s + \Delta_p$ time. If it does not receive message of type $sigError$ in reference to $ID^{(i)}$, it includes the $ID^{(i)}$ in the $\mathcal{ST}$, otherwise it notifies $E^{(i)}$ to restart partial key generation.

20.4.3.5 Audit Algorithm

In our protocol, we audit and disincentivize an entity for two cases, namely for $keyError$ and $dupRegError$, which correspond to inconsistent and duplicate key generation, respectively. The issuance of both cases has been discussed earlier. The audit of type $keyError$ is generated by an entity $E^{(i)}$ when the relation $e\ \ D^{(i)}, P\ \ = e\left(Q^{(i)}, P_0^{(j)}\right)$ does not hold. Similarly, the audit of type $dupRegError$ is initiated by a $TA^{(k)}$ when there is an attempt to generate partial keys for entity $E^{(i)}$ for which a key has already been generated. This can be an attempt to replace keys for entity $E^{(i)}$ or any other attack with similar intent. In the first step, $TA^{(k)}$ initializes an empty set $\mathcal{M}$ to store public keys of malicious TA (or TAs) for disincentivization. Next, it verifies whether the traceable ring signature in $aData$ is valid. If it is invalid, disincentivization cannot occur, and algorithm will return false results. This is because it is trivial to produce invalid signatures. To discuss the Audit Algorithm, we need to discuss the collectAndTrace as defined in Algorithm 3. It takes $ID^{(i)}, \mathcal{L}^{(i)}$ and $aData$. This Algorithm is executed as a subroutine in Algorithm 2. By the end of algorithm, $collectAndTrace$ returns set $\mathcal{M}$ containing the public key (or keys) of malicious TA (or TAs).

It is crucial to note that when it is called, it is executed in parallel, and its execution time is bounded by $\Delta_s + \Delta_p$. It first initializes the set $\mathcal{T}$ and the set $\mathcal{M}$. The set $\mathcal{T}$ stores received traceable ring signatures in form of a tuple $\ m^{(l)}, \sigma^{(l)}\ $. The set $\mathcal{M}$ will be used to add the public key (or keys) of malicious TA (or TAs). Next, it obtains m_{mal} and σ_{mal}, the suspected message and a valid traceable ring signature generated by

Algorithm 1: Update $\mathcal{ST}$ by $TA^{(k)}$

```
Data: First message $< gen,\ \sigma^{(i,j)},\ m^{(i)},\ \mathcal{L}^{(i)} >$ in reference to $\mathcal{L}^{(i)}$
Result: Update $\mathcal{ST}$ and execute Audit if necessary $on-recv(< gen,\ \sigma^{(i,j)},\ m^{(i)},\ \mathcal{L}^{(i)} >)$
 if  $Verify_{trs}(\mathcal{L}^{(i)},\ m^{(i)}, \sigma^{(i,j)}) = 0$ then
    Failed! Notify $E^{(i)}$ to restart partial key generation
    return $\perp$
 end
 $id \leftarrow \mathcal{L}^{(i)}.issue$
 if  $\mathcal{H}_2(id) \in \mathcal{ST}$
    $aType \leftarrow dupRegError$
    $signData \leftarrow \{\sigma^{(i,j)}, m^{(i)}, \mathcal{L}^{(i)}\}$
    $aData \leftarrow \{signData\}$
    $broadcast(< audit, ID^{(i)}, aType, aData >)$
 else
    $wait(\Delta_s + \Delta_p)$
    if  not received $< sigError, ID^{(i)} >$ from $ID^{(i)}$ then
       $\mathcal{ST} \cup \{\mathcal{H}_2(\mathcal{L}^{(i)}.issue)\}$
    else
       Failed! Notify $E^{(i)}$ to restart partial key generation
    end
 end
```

a malicious TA. It then starts a timer and waits for Δ_s time receiving traceable ring signatures generated by $TA \in TA$ for auditing in form of a message $< auditRes, ID^{(i)}, \mathcal{L}^{(i)}, m^{(l)}_{\text{audit}}, \sigma^{(l)}_{\text{audit}} >$. It verifies the $\mathcal{L}_{audit}{}^{(i)}$ and received tag $\mathcal{L}^{(i)}$ are same and the signature is valid. If both are true, then it includes the tuple $m^{(l)},\ \sigma^{(l)}$ in the set $\mathcal{T}$. It is important to note that this procedure of receiving and processing signatures also occurs in parallel. Next, the $Trace_{trs}$ is used to find the malicious public key by running $Trace_{trs}\left(\mathcal{L}^{(i)},\ (m,\sigma),\left(m^{(l)}_{\text{audit}},\sigma^{(l)}_{\text{audit}}\right)\right),\ \forall\ \left(m^{(l)}_{\text{audit}},\sigma^{(l)}_{\text{audit}}\right)\in\mathcal{T}$. If the $|\mathcal{T}| = w$, then we will have at least one $pk^{(y)}\in pk_{ring}$ in the set $\mathcal{M}$ with very high probability. If $|\mathcal{T}| < w$, algorithm includes all the non-signers. Lastly, if $|\mathcal{T}| > w$, few TAs signed the message more than once.

Thus, to get the signers who signed more than once, we run $Trace_{trs}$ for every pair of $\left(m^{(l)}_{\text{audit}},\sigma^{(l)}_{\text{audit}}\right)\in\mathcal{T}$ with every other such pair. In the end, we include such public keys in the set $\mathcal{M}$ and return $\mathcal{M}$. With the discussion of $collectAndTrace$, we can focus on the two cases for Audit Algorithm. Now, depending on the type of error, there are two possibilities:

Algorithm 1: Audit algorithm executed by $TA^{(k)}$

```
**Data:** First Audit Message $< audit, ID^{(i)}, aType, aData >$ in reference to $ID^{(i)}$
**Result:** $\top$ for successful disincentivization or $\perp$ for error
// Initialize empty set $\mathcal{M}$ for storing public keys of malicious TAs
$\mathcal{M} \leftarrow \phi$
$\sigma^{(i,j)} \leftarrow aData.signData.\sigma^{(i,j)}$
$\mathcal{L}^{(i)} \leftarrow aData.signData.\mathcal{L}^{(i)}$
$m^{(i)} \leftarrow aData.signData.m^{(i)}$
if $Verify_{trs}(\mathcal{L}^{(i)}, m^{(i)}, \sigma^{(i,j)}) = 0$ then
    return $\perp$
end
if $aType = keyError$
    if $e(D^{(i)}, P) \neq e(Q^{(i)}, P_0^{(j)})$ then
        $\mathcal{L}_{audit}{}^{(i)} \leftarrow \{ID^{(i)}, pk_{ring}\}$
        $m^{(k)}_{\text{audit}} \leftarrow P_0^{(k)}$
        $\sigma^{(k)}_{audit} \leftarrow Sign_{trs}(sk^{(k)}_{trs}, m^{(k)}_{audit}, \mathcal{L}^{(i)})$
        $broadcast(< auditRes, ID^{(i)}, \mathcal{L}^{(i)}, m^{(k)}_{audit}, \sigma^{(k)}_{audit} >)$
        $\mathcal{M} \leftarrow collectAndTrace(ID^{(i)}, \mathcal{L}_{audit}{}^{(i)}, aData)$
        $wait(\Delta_s + \Delta_p)$
        $Disincentivize(\mathcal{M})$
    end
end
if  $aType = dupRegError$ then
    ${\mathcal{L}_{audit}}^{(i)} \leftarrow \{ID^{(i)}, pk_{ring}\}$
    $m_{\text{audit}}^{(k)} \leftarrow P_0^{(k)}$
    $\sigma_{audit}^{(k)} \leftarrow Sign_{trs}(sk_{trs}^{(k)}, m_{audit}^{(k)}, \mathcal{L}^{(i)})$
    $broadcast(< auditRes, ID^{(i)}, \mathcal{L}^{(i)}, m_{audit}^{(k)}, \sigma_{audit}^{(k)} >)$
    $\mathcal{M} \leftarrow collectAndTrace(ID^{(i)}, {\mathcal{L}_{audit}}^{(i)}, aData)$
    $wait(\Delta_s + \Delta_p)$
    $Disincentivize(\mathcal{M})$
end
return T
```

- Case $keyError$: In the case of $keyError$, the validity of the claim that the generated key is invalid is verified. If the claim is true, $TA^{(k)}$ prepares a traceable ring signature for the audit process endorsing its public key $P_0^{(k)}$ as the message and tag ${\mathcal{L}_{audit}}^{(i)} \leftarrow \{ID^{(i)}, pk_{\text{ring}}\}$. This tag facilitates the finding of malicious TA. After this, $TA^{(k)}$ broadcasts the message of type $auditRes$ with contents $ID^{(i)}, L^{(i)}$and $m_{\text{audit}}^{(k)}$ is broadcasted to every other $TA \in TA$. $TA^{(k)}$ calls the function $collectAndTrace$ with parameters ${\mathcal{L}_{audit}}^{(i)}$ and $aData$. The $collectAndTrace$ executes in parallel to return the set $\mathcal{M}$ in utmost $\Delta_s + \Delta_p$ time. Finally, the nodes disincentivize the participants in the set $\mathcal{M}$ by executing $Disincentivize\,(\mathcal{M})$.
- Case $dupRegError$: In this case as well, $TA^{(k)}$ prepares a traceable ring signature for the audit process endorsing its public key $P_0^{(k)}$ as the message and tag ${\mathcal{L}_{audit}}^{(i)} \leftarrow \{ID^{(i)}, pk_{ring}\}$ and then it broadcasts it with type $auditRes$. Finally, $\mathcal{M}$ is returned by executing $collectAndTrace$ in parallel in at most $\Delta_s + \Delta_p$ time, and the public keys in the set $\mathcal{M}$ are disincentivized.

Algorithm 3: *collectAndTrace* algorithm executed by $TA^{(k)}$

```
Data: $ID^{(i)}, \mathcal{L}_{audit}{}^{(i)}, aData$
Result: Build and Return the set $\mathcal{M}$
\\ Initialize empty set M for storing malicious TRS
$\mathcal{M} \leftarrow \phi$
$m_{mal} \leftarrow aData.signData.m^{(i)}$
$\sigma_{mal} \leftarrow aData.signData.\sigma^{(i,j)}$
$timer \leftarrow 0$
while $timer \leq \Delta_s$
    $on - recv\left(< auditRes, ID^{(i)}, \mathcal{L}^{(i)}, m_{audit}^{(l)}, \sigma_{audit}^{(l)} >\right)$
    if $\mathcal{L}^{(i)} = \mathcal{L}_{audit}{}^{(i)} \wedge Verify_{trs}\left(\mathcal{L}^{(i)}, m_{audit}^{(l)}, \sigma_{audit}^{(l)}\right) = 1$ then
        $\mathcal{T} \cup \{m_{audit}^{(l)}, \sigma_{audit}^{(l)}\}$
    end
end
$\mathcal{M} \cup \{pk^{y}\} \mid \forall \{m_{audit}^{(y)}, \sigma_{audit}^{(y)}\} \in \mathcal{T} \wedge Trace_{trs}\left(\mathcal{L}^{(i)}, (m, \sigma), \left(m_{audit}^{(y)}, \sigma_{audit}^{(y)}\right)\right) = pk^{(y)}$
if $|\mathcal{T}| < w$
    $signers \leftarrow \phi$
    $signers \cup \{lookup(m_{audit}^{(s)}) \mid \exists \{m_{audit}^{(s)}, \sigma_{audit}^{(s)}\} \in \mathcal{T}$
    $\mathcal{M} \cup \{pk_{ring} - signers\}$
end
if $|\mathcal{T}| > w$
    $\mathcal{M} \cup \{ pk\text{^}\{(y)\} \} \mid \{ \forall \{m_{audit1}^{(y)}, \sigma_{audit1}^{(y)}\}, \{m_{audit2}^{(y)}, \sigma_{audit2}^{(y)}\} \in \mathcal{T} \wedge$
    $Trace_{trs}\left( \mathcal{L}^{(i)}, \left(m_{audit1}^{(y)}, \sigma_{audit1}^{(y)}\right), \left(m_{audit2}^{(y)}, \sigma_{audit2}^{(y)}\right)\right) = pk^{(y)}$
end
return $\mathcal{M}$
```

This concludes the discussion of the established protocols and associated algorithms.

20.5 Empirical Results and Analysis

To evaluate the protocol, we designed a simulation using Golang [17]. For communication, protocol buffers were used. We implemented the Fujisaki-Suzuki Traceable Ring Signatures using the Ristretto prime-order group [18]. The group operations were done using the PBC library [19] to implement the original protocol. All the simulations were done on a Laptop with Intel(R) Core (TM) Ultra 7 165H 1.40 GHz processor with 32 GB Memory.

Since the generation of ring signatures is a bottleneck, we evaluated our implementation of traceable ring signatures sequentially and in parallel. As depicted in Figure 20.5, we evaluated the signature generation and verification process with ring sizes 10, 50, and 100. The time required to generate and verify 1000 signatures was recorded. As evident in Figure 20.5 (c), for a ring size of 100, the generation and verification took about 20 s. However, this is resolved when multiple signature generation and verifications are done in parallel; for a similar ring size of 100, the duration was reduced to 2.2 s. A similar reduction by a factor of 10 was also observed for ring sizes 10 and 50. Our evaluation shows that the scheme can be used for practical purposes.

Moreover, we simulated different instances with variations in the number of TAs and Entities. The results of the simulation are presented in Figure 20.6. The packet delays followed the Poisson distribution. We varied the number of Entities $(nEntities)$ by 50, 100, and 200. The number of Trusted Authorities (nTA) varied by 5, 10, and 20. For $nEntities = 50$, we observe that the

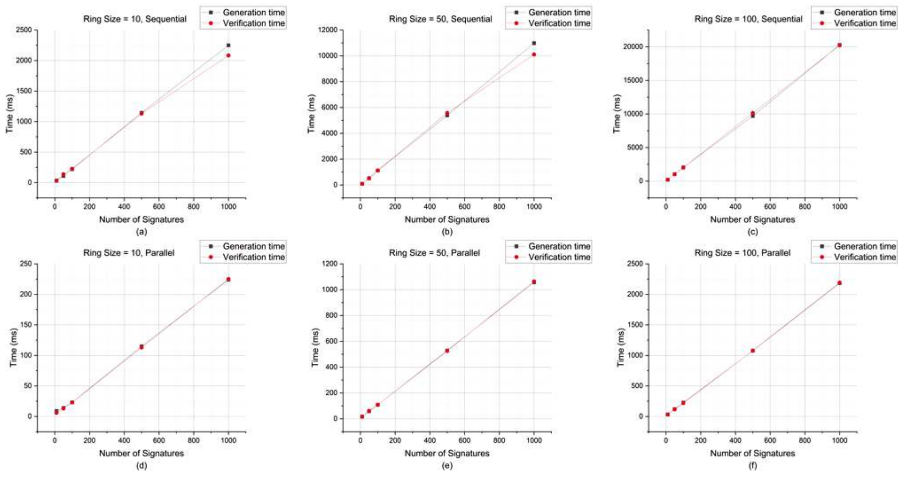


Figure 20.5 Benchmark for traceable ring signatures for sequential and parallel execution. ⏎

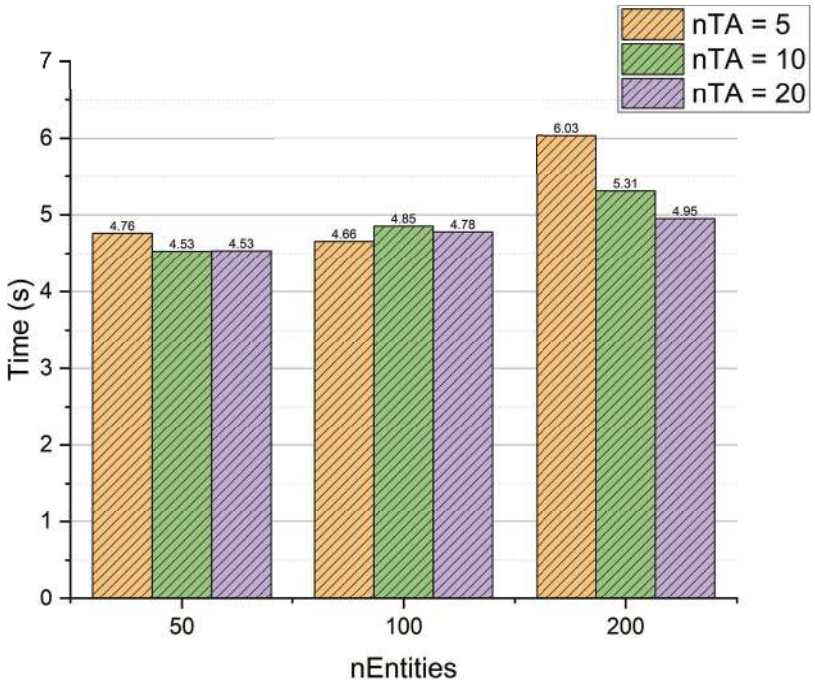


Figure 20.6 Key generation benchmarking for $\boldsymbol{nTA} = 5, 10, 20$ with different number of Entities. ⏎

total time for both $nTA = 10$ and $nTA = 20$ is upper bound by 4.5 s. This result is in line with the expectation that as more TAs are present, the load of key generation is distributed equitably. However, it is essential to notice that the size of the response from TA to Entity will be larger due to the increased size of the ring signature. This results in transmission and processing delays. The effect of increased signature sizes can be seen from the case for $nEntities = 100$. In this case, the configuration with $nTA = 5$ outperforms the configuration for $nTA = 10$ and $nTA = 20$. It is also interesting to observe that the configuration with $nTA = 20$ outperforms the configuration with $nTA = 10$. However, as more entities are added to the system, the effect of this decentralization is evident. The time required for key generation for $nEntities = 200$ is the minimum for the configuration with $nTA = 20$. Therefore, as more entities are added to the system, the protocol performs better with more trusted authorities. However, the requirement for the number of TAs varies from case to case.

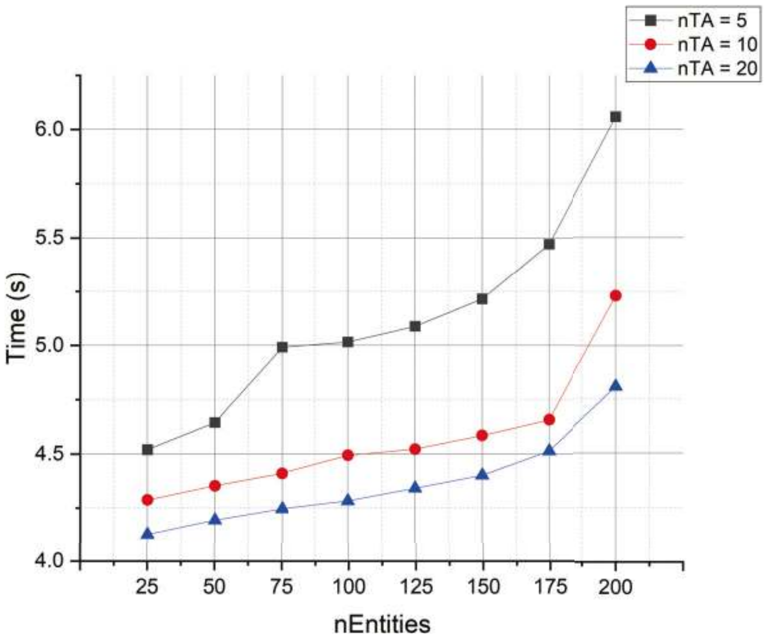


Figure 20.7 Single run comparison of key generation for $\boldsymbol{nTA} = 5$, 10, 20 with different number of Entities. ⏎

We also evaluate a single protocol run with various configurations with $nTA = 5$, 10, and 20, respectively. The time was recorded as more entities joined the system and requested a partial key generation and the corresponding data is reflected in Figure 20.7. Finally, the duration for successful completion of the complete protocol was recorded as well. The results are depicted in Figure 20.6. In this specific case, it can be inferred that the performance of the configurations $nTA = 10$ and $nTA = 20$ are very close to each other until $nEntities = 175$. As more entities are introduced, configuration with $nTA = 20$ achieves better time performance. However, as stated previously, the performance may vary on average due to large ring signatures. This can be optimized by fixing a specific ring for every key generation response.

20.6 Conclusions and Future Works

In this work, we have established a protocol to generate a CL-PKC-based private key with the network model and multiple trusted authorities. We have also evaluated the computation and communication bottlenecks for efficient protocol implementation. The results indicate that the model can be adapted to accommodate the distributed nature of various applications. Considering the applications, the network model can have multiple use cases. For instance, in Federated Learning, a sensor node may not want to opt for a specific Trusted Authority confined to a geographic location or organization. This model facilitates the decentralization of the trust model in this case. It is important to note that we have established the accountability model for the general case of certificateless partial key generation. This work can be extended to accommodate the multiple trusted authorities in the existing models for Distributed Learning. Many of these models have established privacy-preserving solid models. However, inclusion must be made so the pre-existing guarantees are intact.

Acknowledgements

This work is supported by the European Network of Excellence dAIEDGE under Grant Agreement Nr. 101120726 and by CyberSecurity Research Flanders with reference number VR20192203.

References

[1] A. Shamir, "Identity-Based Cryptosystems and Signature Schemes," Advances in Cryptology, pp. 47-53, 1985. W442W7307

[2] D. Boneh and M. K. Franklin, "Identity-Based Encryption from the Weil Pairing," Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology, pp. 213-229, 2001. W442W7307

[3] A.-R. S. S., "Certificateless Public Key Cryptography," Advances in Cryptology - ASIACRYPT 2003, pp. 452-473, 2003. W442W7307

[4] L. Zhang, Q. Wu, J. Domingo-Ferrer, B. Qin and P. Zeng, "Signatures in hierarchical certificateless cryptography: efficient constructions and provable security," Information Sciences, vol. 272, pp. 223-237, 2014. W442W7307

[5] L. Zhang, Q. Wu, J. Domingo-Ferrer and B. Qin, "Hierarchical certificateless signatures," 2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, pp. 572-577, 2010. W442W7307

[6] Y. Jiang, K. Zhang, Y. Qian and L. Zhou, "Anonymous and efficient authentication scheme for privacy-preserving distributed learning," IEEE Transactions on Information Forensics and Security, vol. 17, pp. 2227-2240, 2022. W442W7307

[7] Y. Ma, Q. Cheng and X. Luo, "2PCLA: Provable Secure and Privacy Preserving Enhanced Certificateless Authentication Scheme for Distributed Learning," IEEE Transactions on Information Forensics and Security, 2023. W442W7307

[8] X. Yuan, J. Liu, B. Wang, W. Wang, T. Li, X. Ma and W. Pedrycz, "Fedcomm: A privacy-enhanced and efficient authentication protocol for federated learning in vehicular ad-hoc networks," IEEE Transactions on Information Forensics and Security, 2023. W442W7307

[9] L. Zhang, J. Liu and R. Sun, "An efficient and lightweight certificateless authentication protocol for wireless body area networks," 2013 5th International Conference on Intelligent Networking and Collaborative Systems, pp. 637-639, 2013. W442W7307

[10] R. L. Rivest, A. Shamir and Y. Tauman, "How to leak a secret," Advances in Cryptology—ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast, Australia, December 9–13, 2001 Proceedings 7, pp. 552-565, 2001. W442W7307

[11] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan and others, "An empirical analysis of traceability in the monero blockchain," arXiv preprint arXiv:1704.04299, 2017. W442W7307

[12] E. Fujisaki and K. Suzuki, "Traceable ring signature," International Workshop on Public Key Cryptography, pp. 181-200, 2007. W442W7307

[13] G. Wood and others, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum project yellow paper, vol. 151, no. 2014, pp. 1-32, 2014. W442W7307

[14] R. C. Merkle, "Protocols for Public Key Cryptosystems," p. 122, 4 1980. W442W7307

[15] D. R. Morrison, "PATRICIA—Practical Algorithm To Retrieve Information Coded in Alphanumeric," Journal of the ACM (JACM), vol. 15, pp. 514 - 534, 1968. W442W7307

[16] W. Diffie and M. E. Hellman, "New Directions in Cryptography," Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman, pp. 365-390, 2022. W442W7307

[17] R. Griesemer, R. Pike and K. Thompson, The Go Programming Language, 2009. W442W7307

[18] M. Hamburg, H. de Valence, I. Lovecruft and T. Arcieri, Ristretto: Prime-Order Elliptic Curve Groups, 2021. W442W7307

[19] B. Lynn, The PBC (Pairing-Based Cryptography) Library, 2006. W442W7307

Index

About the Editors

Dr. Ovidiu Vermesan holds a PhD degree in microelectronics and a Master of International Business (MIB) degree. He is Chief Scientist at SINTEF Digital, Oslo, Norway. His research interests are intelligent systems integration, mixed-signal embedded electronics, analogue neural networks, edge artificial intelligence and cognitive communication systems. Dr. Vermesan received SINTEF's 2003 award for research excellence for his work on implementing a biometric sensor system. He is currently working on projects addressing nanoelectronics, integrated sensor/actuator systems, communication, cyber-physical systems (CPSs) and the Industrial Internet of Things (IIoT), with applications in green mobility, energy, autonomous systems, and smart cities. He has authored or co-authored over 100 technical articles and conference papers. He is actively involved in the activities of the European partnership for Key Digital Technologies (KDT) Joint Undertaking (JU), now the Chips JU. He has coordinated and managed various national, EU and other international projects related to smart sensor systems, integrated electronics, electromobility and intelligent autonomous systems such as E3Car, POLLUX, CASTOR, IoE, MIRANDELA, IoF2020, AUTOPILOT, AutoDrive, ArchitectECA2030, AI4DI, AI4CSM. Dr. Vermesan actively participates in national, Horizon Europe and other international initiatives by coordinating and managing various projects. He is a member of the Alliance for AI, IoT and Edge Continuum Innovation (AIOTI) board. He is currently the coordinator of the Edge AI Technologies for Optimised Performance Embedded Processing (EdgeAI) project.

Dr, Alain Pagani is Principal Researcher and deputy director of the Augmented Vision research department at the German Research Center for Artificial Intelligence (DFKI). His research interests include Artificial Intelligence, Computer Vision, Image Understanding, and Extended Reality. He is the coordinator of the Network of Excellence dAIEDGE about distributed AI at the edge, and the coordinator of the Horizon Europe project CORTEX about remote cooperation using extended reality. He is lecturer at the

University of Kaiserslautern-Landau, and he published over 100 articles in conferences and journals. His research finds applications in eXtended Reality for tele cooperation (project CORTEX), Artificial Intelligence and Computer Vision for Human-Robot cooperation (project FLUENTLY), Artificial Intelligence and Augmented Reality for analysis of extremely large data (project ExtremeXP).

Dr. Paolo Meloni is currently associate professor at the Department of Electrical and Electronic Engineering (DIEE) at the University of Cagliari. He is author of several international papers and tutor of many bachelor and master student's thesis in Electronic Engineering. His research activity is mainly focused on the development of advanced digital systems, programming of multi-core on-chip architectures, with emphasis on the application-driven design and programming of SoCs and FPGAs.