OPTIMIZING EDGE AND FOG COMPUTING APPLICATIONS WITH AI AND METAHEURISTIC ALGORITHMS

Edited by
Madhusudhan H S, Punit Gupta,
and Dinesh Kumar Saini

An **Auerbach** Book

CRC Press
Taylor & Francis Group

OPTIMIZING EDGE AND FOG COMPUTING APPLICATIONS WITH AI AND METAHEURISTIC ALGORITHMS

Edited by
Madhusudhan H S, Punit Gupta,
and Dinesh Kumar Saini

An Auerbach Book

CRC Press
Taylor & Francis Group

Optimizing Edge and Fog Computing Applications with AI and Metaheuristic Algorithms

Fog and edge computing are two paradigms that have emerged to address the challenges associated with processing and managing data in the era of the Internet of Things (IoT). Both models involve moving computation and data storage closer to the source of data generation, but they have subtle differences in their architectures and scopes. These differences are one of the subjects covered in *Optimizing Edge and Fog Computing Applications with AI and Metaheuristic Algorithms*. Other subjects covered in the book include:

- Designing machine learning (ML) algorithms that are aware of the resource constraints at the edge and fog layers ensures efficient use of computational resources
- Resource-aware models using ML and deep leaning models that can adapt their complexity based on available resources and balancing the load, allowing for better scalability
- Implementing secure ML algorithms and models to prevent adversarial attacks and ensure data privacy
- Securing the communication channels between edge devices, fog nodes, and the cloud to protect model updates and inferences
- Kubernetes container orchestration for fog computing

- Federated learning that enables model training across multiple edge devices without the need to share raw data

The book discusses how resource optimization in fog and edge computing is crucial for achieving efficient and effective processing of data close to the source. It explains how both fog and edge computing aim to enhance system performance, reduce latency, and improve overall resource utilization. It examines the combination of intelligent algorithms, effective communication protocols, and dynamic management strategies required to adapt to changing conditions and workload demands. The book explains how security in fog and edge computing requires a combination of technological measures, advanced techniques, user awareness, and organizational policies to effectively protect data and systems from evolving security threats. Finally, it looks forward with coverage of ongoing research and development, which are essential for refining optimization techniques and ensuring the scalability and sustainability of fog and edge computing environments.

Madhusudhan H S is an associate professor in the Department of Computer Science and Engineering at Vidyavardhaka College of Engineering, Mysuru, India.

Punit Gupta is an associate professor in the Department of Computer and Communication Engineering at Pandit Deendayal Energy University, Gujarat, India.

Dinesh Kumar Saini is a full professor at the School of Computing and Information Technology, Manipal University, Jaipur, India.

Optimizing Edge and Fog Computing Applications with AI and Metaheuristic Algorithms

Edited by

Madhusudhan H S, Punit Gupta, and Dinesh Kumar Saini

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
AN AUERBACH BOOK

Designed cover image: Shutterstock

First edition published [2026]
by CRC Press
2385 NW Executive Center Drive, Suite 320, Boca Raton FL 33431

and by CRC Press
4 Park Square, Milton Park, Abingdon, Oxon, OX14 4RN

CRC Press is an imprint of Taylor & Francis Group, LLC

ISBN: 9781041003540 (hbk)
ISBN: 9781041005001 (pbk)
ISBN: 9781003610168 (ebk)

DOI: 10.1201/9781003610168

Typeset in Adobe Garamond
by Newgen Publishing UK

Contents

Editors

Madhusudhan H S is an associate professor in the Department of Computer Science and Engineering at Vidyavardhaka College of Engineering, Mysuru, Karnataka, India. He has published research articles in reputed international journals and conferences including SCI-indexed and Scopus-indexed journals. His areas of interest include cloud computing, artificial intelligence, machine learning and computer networks. He holds a doctoral degree from Visvesvaraya Technological University (VTU), Belagavi, Karnataka, India.

Punit Gupta is an assistant professor in the Department of Computer Science and Engineering at Pandit Deendayal Energy University. He has completed post-doctoral research at University College Dublin, Ireland. He has completed his PhD in Computer Science and Engineering from Jaypee University of Information Technology, Solan, India. He is Gold Medalist in M-Tech from Jaypee Institute of Information Technology. He has research experience in Internet-of-Things, cloud computing, and distributed algorithms and has authored more than 80 research papers in reputed journals and international conferences. He is gest editor of *Recent Patent on Computer Science* journal and editorial manager of *Computer Standards & Interfaces* and *Journal of Network and Computer Applications*.

Dinesh Kumar Saini has a PhD in Computer Science, an ME in Software Systems, and an MSc in Technology from one of the premier universities in India, BITS Pilani, Rajasthan, India. Dr. Dinesh Kumar Saini is a full professor in the Department of Computer and Communication Engineering, School of Computing and Information Technology, Manipal University, Jaipur, India. Dr. Dinesh Kumar has vast experience in academics—as a professor, researcher, and administrator—in Indian universities like BITS Pilani and abroad and proven record of accomplishment of leadership skills in higher education and tertiary education sector. He served as dean of the Faculty of Computing and Information Technology at Sohar University in Sultanate of Oman. He has been an associate professor at Sohar University, Oman, since 2008, and an adjunct associate and research fellow at the University of Queensland, Brisbane, Australia, between 2010 and 2015. He has also been the founder program coordinator and head of the Department of Business Information Technology for more than 10 years. He won Emerald Literati Award for 2018 for his article "Modeling human factors influencing herding during evacuation" published in *International Journal of Pervasive Computing and Communications*. Besides his academic credentials aptly measured by different quantifiable metrics (e.g., ResearchGate score 36.36, Google citation count 1858, H-Index: 19 and i10 Index: 42), Dr. Dinesh Kumar believes in the spirit of teamwork and therefore has constantly augmented and consolidated research capacity building at the faculty by encouraging and supporting his fellow colleagues and junior faculty members to publish in journals and participate in conferences. This is evident in several publications that have been done in collaboration with his teammates in the faculty. He has visited the USA, the UK, France, Germany, Austria, UAE, Australia, Russia, Bahrain, and KSA for academic purpose and learned a lot of good practices in the universities of these developed countries.

Contributors

Swathi B. H.
Department of Computer Science and Engineering
Vidyavardhaka College of Engineering
Mysore, Karnataka, India

Deepti Bhat
Department of Computer Science and Engineering
Vidyavardhaka College of Engineering
Mysore, Karnataka, India

Ansh Chauhan
Department of Computer Science and Engineering
Vidyavardhaka College of Engineering
Mysore, Karnataka, India

Kavitha D. N.
Department of Computer Science and Engineering
Vidyavardhaka College of Engineering
Mysore, Karnataka, India

Arbaz Adib Dalwai
School of Computing
National College of Ireland
Dublin, Ireland

Manisha Gupta

Rajasthan College of Engineering for Women
Jaipur, Rajasthan, India

Pratik Gupta
Jaypee University of Engineering and Technology
Guna, Raghogarh-Vijaypur
Madhya Pradesh, India

Punit Gupta
Department of computer science
Pandit Deendayal Energy University,
Gandhinagar, India
and
School of Computing,
National College of Ireland
Dublin, Ireland

Yash Jain
Department of Computer Science and Engineering
Jaypee University of Engineering and Technology
Guna University
Raghogarh-Vijaypur, Madhya Pradesh, India

Shivani Jaswal
School of Computing,
National College of Ireland
Dublin, Ireland

Harshitha Kamal Kannan
Department of Computer Science and Engineering,
Vidyavardhaka College of Engineering
Mysore, Karnataka, India

Gayana J. Kumar
Department of Computer Science and Engineering
Vidyavardhaka College of Engineering
Mysore, Karnataka, India

Nidhi Kushwaha
National Institute Technology
Jamshedpur, India

Priyanshi Mishra
Jaypee University of Engineering and Technology
Guna University
Raghogarh-Vijaypur, Madhya Pradesh, India

Vedavathi N.
Department of Computer Science and Engineering
Vidyavardhaka College of Engineering
Mysore, Karnataka, India

Chiranth Nagaprasanna
Department of Computer Science and Engineering
Vidyavardhaka College of Engineering
Mysore, Karnataka, India

Rajkumar Nagulsamy
School of Computing
National College of Ireland
Dublin, Ireland

Anirudh Negi
School of Computing
DIT University

Dehradun, Uttarakhand, India

Sahil Raj
School of Computing
DIT University
Dehradun, Uttarakhand, India

Pradeep Kumar Rawat
School of Computing
DIT University
Dehradun, Uttarakhand, India

Chirashwi S.
Department of Computer Science and Engineering
Vidyavardhaka College of Engineering
Mysore, Karnataka, India

Bhavna Saini
Central University of Rajasthan
Ajmer, Rajasthan, India

Saanch Sapra
Jaypee University of Information Technology
Waknaghat, Himachal Pradesh, India

Bharat Singh
Indian Institute of Information Technology
Ranchi, Jharkhand, India

Nayana Singh
Jaypee University of Information Technology
Waknaghat, Himachal Pradesh, India

Shristi Sonam
Jaypee University of Information Technology
Waknaghat, Himachal Pradesh, India

Prateek Kumar Soni
Jaypee Institute of Information Technology
Noida, Uttar Pradesh, India

Rohit Verma
School of Computing
National College of Ireland
Dublin, Ireland

Vivek Kumar Verma
Manipal University Jaipur
Jaipur, Rajasthan, India

Chapter 1

Introduction to Resource Optimization in Fog and Edge Computing

Pratik Gupta and Prateek Kumar Soni

DOI: 10.1201/9781003610168-1

1.1 Introduction

Resource optimization in fog and edge computing involves the efficient allocation, management, and utilization of computational, storage, and network resources. It is critical for:

1. Performance Enhancement: Resource optimization in fog and edge computing is critical for real-time applications that require rapid responses, such as autonomous vehicles and healthcare monitoring. By processing data closer to the source, tasks experience reduced latency, ensuring faster decision-making. Optimized task scheduling, load balancing, and prioritization of time-sensitive workloads further

enhance responsiveness. High throughput is achieved by leveraging parallel processing, dynamic resource provisioning, and data aggregation techniques, enabling the system to handle large volumes of data efficiently. Together, these optimizations ensure that real-time applications operate seamlessly with minimal delays and maximum efficiency.

2. Energy Efficiency: Energy efficiency is crucial in fog and edge computing, particularly for battery-powered edge devices. By optimizing resource usage and minimizing unnecessary computations or data transmissions, energy consumption is reduced. Techniques such as task offloading, where computation is shifted to more power-efficient fog nodes, and the use of low-power hardware can extend battery life. Additionally, dynamic resource scaling and energy-aware scheduling ensure that edge devices consume energy only when necessary, balancing performance needs with power efficiency. This optimization helps prolong the lifespan of edge devices and reduces operational costs.
3. Cost Reduction: In fog and edge computing, cost reduction is achieved through efficient resource utilization, where computational, storage, and network resources are allocated and managed optimally. By reducing over-provisioning and underutilization of resources, organizations can minimize infrastructure costs. Techniques like dynamic scaling allow resources to be provisioned on demand, ensuring that only the necessary amount is used at any given time. Task offloading to fog or cloud nodes, when local processing is inefficient, further reduces the cost of maintaining high-performance edge devices. This approach ensures

that resources are used effectively, lowering both capital and operational expenses.

4. Scalability: Scalability in fog and edge computing ensures that the system can accommodate an increasing number of devices and applications without sacrificing performance. By dynamically provisioning resources and distributing workloads across multiple edge and fog nodes, the system adapts to rising demands. Efficient resource management strategies, such as load balancing and elastic scaling, enable seamless expansion, ensuring that new devices or applications are integrated smoothly. This scalability ensures that performance remains consistent even as the network grows, preventing bottlenecks and maintaining high throughput and low latency for all connected devices.
5. Quality of Service (QoS): Meeting the specific requirements of diverse applications, including reliability, latency, and bandwidth. In fog and edge computing, QoS ensures that diverse applications, each with unique performance needs, receive the appropriate level of service. QoS management addresses requirements such as low latency for real-time applications, high reliability for critical systems, and sufficient bandwidth for data-intensive tasks. By prioritizing traffic, allocating resources dynamically, and implementing efficient routing and scheduling, QoS guarantees that applications meet their specific performance criteria. This approach helps maintain seamless operations across various use cases, from healthcare and industrial monitoring to autonomous systems, ensuring that the network can handle

different types of demands without compromising on service quality [1].

Key Challenges

1. Heterogeneity: Fog and edge environments involve diverse devices with varying computational power, storage, and energy constraints.
   Fog and edge environments are characterized by a wide range of devices with differing computational capabilities, storage capacities, and energy limitations. These environments often include a mix of resource-rich devices, such as powerful edge servers, and resource-constrained devices like Internet of Things (IoT) sensors or mobile devices. The heterogeneity in these devices presents challenges in managing workloads efficiently, ensuring data processing is optimized for the available resources, and maintaining system reliability under varying energy constraints [2–5]. As a result, solutions must be adaptive and scalable to accommodate the diverse nature of devices and their specific operational requirements within fog and edge networks.
2. Dynamic Workloads: Real-time applications generate fluctuating workloads, requiring adaptive resource allocation.
   Real-time applications often generate fluctuating workloads that can vary significantly depending on factors such as user demand, environmental changes, or system conditions. These applications, which include streaming, gaming, and autonomous systems, require timely processing of data with minimal latency. To ensure optimal performance, resources like

computational power, storage, and network bandwidth need to be allocated dynamically, adjusting in real-time to meet the demands of these workloads. This adaptability is essential in preventing bottlenecks, maintaining responsiveness, and efficiently using available resources, as the system must quickly respond to both anticipated and unexpected fluctuations in workload intensity.

3. Limited Resources: Edge devices typically have less computational power and storage than cloud data centers.

   Edge devices, which are positioned closer to the data source or end-user, typically have less computational power, storage capacity, and energy resources compared to centralized cloud data centers. These devices are designed to perform localized processing and analysis, often with constraints such as limited processing speed, smaller memory, and restricted power availability. While cloud data centers benefit from robust infrastructure, including powerful servers and vast storage networks, edge devices must operate efficiently within these limitations. As a result, managing resource allocation at the edge requires careful optimization, balancing performance and energy consumption while ensuring that critical tasks can still be performed locally without relying heavily on distant cloud resources.
4. Geographic Distribution: Devices are spread across various locations, complicating resource coordination and task scheduling.

   In fog and edge computing environments, devices are often geographically distributed across diverse locations, ranging from urban centers to remote or rural areas. This widespread distribution complicates the coordination of resources and task scheduling, as it introduces challenges such as network latency, bandwidth limitations, and varying availability of resources. Tasks may need to be assigned to devices based not only on their capabilities but also on their proximity to data sources, user locations, or specific regional requirements. Efficiently managing these distributed resources requires intelligent scheduling and load balancing algorithms that can take into account the geographical spread, ensuring that tasks are processed where they can be completed most effectively, minimizing delays, and optimizing resource usage across the network [6].

5. Security and Privacy: Ensuring secure data transmission and computation without overloading resource-constrained devices.
   Ensuring secure data transmission and computation on resource-constrained devices requires a balance between robust encryption methods and efficient resource management. Lightweight cryptographic algorithms, such as elliptic curve cryptography (ECC) or lightweight symmetric ciphers, can be employed to minimize the computational overhead while maintaining strong security. Additionally, data can be encrypted locally on the device before transmission and decrypted only at the receiving end, ensuring privacy. Optimizing protocols for low-power consumption and leveraging edge computing or distributed processing can further reduce the strain on resource-constrained devices, allowing them to securely handle sensitive data without sacrificing performance or security.

1.2 Strategies for Resource Optimization

1. Task Offloading: Deciding whether to process tasks locally, at the fog layer, or in the cloud based on resource availability and application requirements [1, 7, 8].
   Task offloading is a strategic decision-making process where tasks are dynamically assigned to different computational layers—local devices, fog nodes, or the cloud—based on factors like resource availability, network conditions, and application requirements. Local processing is favored when low latency and quick response times are crucial, while fog computing offers a middle ground by providing computational resources closer to the user, reducing latency without overloading the local device. Offloading to the cloud is ideal for resource-intensive tasks or when large-scale data processing and storage are needed, though it may introduce higher latency due to network transmission. By considering factors such as computational power, energy efficiency, and real-time demands, task offloading ensures efficient resource utilization and optimal system performance.
2. Load Balancing: Distributing workloads across devices to prevent bottlenecks and ensure efficient utilization.
   Load balancing involves distributing workloads across multiple devices or resources to prevent any single device from becoming overwhelmed, thereby avoiding bottlenecks and ensuring efficient utilization of available resources. By intelligently allocating tasks based on factors like processing power, current workload, and network conditions, load balancing helps maintain consistent system performance, minimizes

delays, and maximizes throughput. In systems with diverse computational resources, such as edge devices, fog nodes, and cloud servers, effective load balancing ensures that no single layer or device is overburdened, while optimizing overall efficiency, reducing latency, and improving the responsiveness of applications [9].

3. Energy-Aware Computing: Designing algorithms that minimize energy consumption while maintaining performance.
   Energy-aware computing involves designing algorithms and systems that optimize energy consumption without compromising performance. By intelligently managing computational resources, such as dynamically adjusting processing power, reducing unnecessary operations, or offloading tasks to more energy-efficient layers (e.g., fog or cloud), energy-aware algorithms aim to balance energy efficiency with the required performance levels. This approach is particularly crucial in mobile, IoT, and edge computing environments, where energy resources are limited, and devices must operate for extended periods without frequent recharging. Through techniques like workload distribution, hardware-level optimizations, and adaptive processing, energy-aware computing helps extend device battery life while ensuring that applications continue to function effectively.
4. Caching and Data Placement: Optimizing data storage and retrieval to reduce latency and bandwidth usage.
   Caching and data placement involve strategically storing data in locations that optimize access speed and reduce the demand on network resources. By placing frequently accessed data closer to the user or

at intermediate nodes (such as edge or fog servers), caching minimizes latency and reduces the need for repeated data retrieval from distant storage locations, such as the cloud. Effective data placement further ensures that data is stored where it will be most efficiently accessed, considering factors like usage patterns, resource availability, and network conditions. This approach not only accelerates data retrieval times but also lowers bandwidth usage by decreasing the amount of data transmitted over the network, thereby improving overall system performance and user experience [10, 11].

5. Artificial intelligence (AI) and Machine Learning: Employing predictive models to adapt resource allocation dynamically based on historical and real-time data [10–15].

   AI and machine learning in resource allocation involve using predictive models to dynamically adjust resources based on both historical data and real-time conditions. By analyzing patterns in past usage, system performance, and environmental factors, these models can forecast future demands and optimize resource distribution accordingly. For instance, machine learning algorithms can predict when computational resources will be heavily needed or when energy consumption will spike, allowing systems to preemptively allocate resources or adjust processes to maintain efficiency. This adaptive approach ensures that resources are utilized optimally, reduces waste, improves performance, and enhances user experience by responding intelligently to changing conditions in real time.

1.3 Applications

Smart Cities: Optimizing resources for applications like traffic management, surveillance, and waste management.

Smart cities leverage fog and edge computing to optimize resources for critical applications such as traffic management, surveillance, and waste management. By processing data closer to the source, these technologies reduce latency, enabling real-time decision-making and enhancing system responsiveness. For example, traffic management systems can dynamically adjust signals to alleviate congestion, surveillance systems can analyze footage for immediate threat detection, and waste management can streamline collection schedules based on real-time fill levels of bins. This localized data processing results in improved operational efficiency, reduced resource wastage, and a better quality of urban living.

Healthcare: Enhancing the performance of remote health monitoring and diagnostic systems.

In healthcare, fog and edge computing enhance the performance of remote health monitoring and diagnostic systems by processing data closer to patients and healthcare providers. This reduces latency, enabling real-time analysis of vital signs, early detection of anomalies, and quicker decision-making. For instance, wearable devices can continuously monitor a patient’s health and transmit critical data to nearby edge servers for immediate analysis, alerting medical professionals regarding potential issues. Additionally, diagnostic systems can process and share medical imaging data locally, minimizing delays in diagnosis and treatment.

This localized approach improves patient outcomes, optimizes resource utilization, and supports the delivery of timely and personalized healthcare services.

Industrial IoT (IIoT): Managing resources in industrial automation and predictive maintenance.

In IIoT, fog and edge computing play a crucial role in managing resources for industrial automation and predictive maintenance. By processing data at the edge, these technologies minimize latency and enable real-time monitoring of machinery and systems. For example, sensors on manufacturing equipment can continuously collect performance data and analyze it locally to detect potential faults before they lead to failures, thereby reducing downtime. Similarly, automation systems can make rapid decisions based on localized data to optimize production processes. This approach not only enhances operational efficiency but also extends equipment lifespan and reduces maintenance costs, making industries more resilient and cost-effective.

Autonomous Vehicles: Reducing latency for real-time decision-making and data processing.

In autonomous vehicles, fog and edge computing are essential for reducing latency in real-time decision-making and data processing. These technologies enable vehicles to process vast amounts of sensor data, such as information from cameras, Light Detection and Ranging (LiDAR), and radar, directly on the vehicle or at nearby edge servers. This localized processing ensures rapid responses to dynamic driving conditions, such as detecting obstacles, navigating traffic, or responding to sudden hazards. For example, when a pedestrian crosses unexpectedly, edge computing allows the vehicle to instantly process the sensor data and apply the brakes without relying on distant cloud servers. By minimizing delays, fog and edge computing enhance safety, improve navigation efficiency, and support the reliable operation of autonomous systems in real-world environments.

Resource optimization in fog and edge computing plays a pivotal role in meeting the demands of modern applications. By addressing the challenges and adopting effective strategies, these paradigms can unlock their full potential to support a wide range of use cases.

1.4 Resource Optimization in Fog and Edge Computing: A Detailed Overview

Resource optimization in fog and edge computing is essential to address the unique challenges posed by these decentralized environments, which bring computation, storage, and networking closer to data sources like IoT devices. By minimizing latency, these paradigms enhance application performance, enabling real-time responses and efficient operations. However, the limited computational capacity of edge devices, dynamic network conditions, and the need to balance energy efficiency with performance demand innovative resource optimization techniques. Effective strategies ensure that these systems achieve their full potential to support diverse, latency-sensitive applications while overcoming the inherent complexity of decentralized architectures.

Fog and edge computing shift computation, storage, and networking closer to the data sources, such as IoT devices, to minimize latency and improve application performance.

However, these decentralized environments present unique challenges that demand effective resource optimization techniques.

1. Challenges in Resource Optimization
    a. Heterogeneity

       Heterogeneity is one of the most significant challenges in resource optimization within fog and edge computing environments. These systems consist of diverse devices, each with varying capabilities in computation power, storage capacity, and energy availability. Managing resources effectively in such a mixed ecosystem requires addressing the following aspects:

       Fog and edge environments consist of diverse devices, including sensors, gateways, and edge servers, with varying capabilities in terms of:

       i. Computation power (e.g., IoT sensors vs. edge servers).

          The computational capabilities of devices in fog and edge environments vary widely. For example:

          - **IoT Sensors:** These are lightweight devices designed primarily for data collection and transmission. They have minimal computational capabilities and rely on edge or cloud servers for processing.
          - **Edge Servers:** These are more powerful devices located closer to the data sources. They can handle intensive computation tasks but are still less capable than centralized data centers. The disparity in

computational power necessitates dynamic task allocation strategies, where lightweight devices offload tasks to more capable nodes or edge servers. Balancing these tasks to prevent overloading high-capacity nodes while ensuring efficient operation of low-capacity devices is critical

ii. Storage capacity (e.g., low-capacity edge nodes vs. centralized data centers).

    Storage capabilities differ significantly across devices:

    - **Low-Capacity Edge Nodes:** Many edge nodes, such as gateways and smaller servers, have limited storage capacity, making them suitable for temporary data caching or short-term storage.
    - **Centralized Data Centers:** These have vast storage resources and are ideal for archiving large datasets and performing long-term analytics. Efficient storage management requires optimizing where and how data is stored. For instance, frequently accessed data can be cached at edge nodes to reduce latency, while less critical data can be transmitted to centralized data centers for long-term storage.

iii. Energy availability (e.g., battery-operated devices vs. mains-powered nodes).

    Energy constraints are a critical factor in heterogeneous environments:

- **Battery-Operated Devices:** Many IoT sensors and edge devices operate on batteries, which makes energy efficiency a top priority. These devices must balance performance with energy conservation to extend their operational life.
- **Mains-Powered Nodes:** Devices connected to a stable power source, such as edge servers and gateways, can perform more energy-intensive operations without concern for battery depletion. Resource optimization strategies must consider the energy profiles of devices to prioritize energy-efficient operations for battery-powered nodes while maximizing the capabilities of mains-powered devices.

iv. Limited Resources: Unlike cloud data centers, edge devices have constrained resources, such as:

Limited processing power for executing computation-intensive tasks.

Edge devices, such as IoT sensors, gateways, and small servers, often lack the processing capabilities required for computation-intensive tasks. For example:

- **IoT Sensors:** These devices are primarily designed for data collection and are typically equipped with low-power processors to conserve energy.
- **Edge Gateways:** While more powerful than sensors, gateways have limited computational resources compared to cloud

servers. This constraint means that tasks requiring significant computational power, such as AI model training, large-scale analytics, or video processing, must be simplified, distributed across multiple edge devices, or offloaded to nearby cloud nodes. The challenge lies in designing systems that intelligently distribute or offload tasks while maintaining low latency and high reliability.

v. Restricted storage for data caching and persistence.

Edge devices are also limited in their ability to store large volumes of data. For instance:

- **Temporary Caching:** Many edge devices can only cache data for short durations, which is suitable for real-time or near-real-time applications but inadequate for long-term storage.
- **Data Persistence:** The lack of sufficient storage means that historical data or less frequently accessed data must be transferred to cloud servers or centralized data centers for archiving and analysis. Effective resource optimization strategies for storage involve determining what data should be cached at the edge, transmitted to the cloud, or discarded. This requires intelligent algorithms that consider data importance, access frequency, and storage availability to optimize storage usage without compromising performance.

vi. Bandwidth constraints in communicating with other devices or the cloud.

    Bandwidth limitations significantly impact the ability of edge devices to communicate with other devices or the cloud. Key challenges include:

    - **Limited Connectivity:** Many edge environments rely on wireless networks, which may suffer from limited bandwidth, latency, or intermittent connectivity.
    - **Data Transmission Costs:** Transmitting large amounts of data from edge devices to the cloud can be expensive and inefficient, especially for applications requiring high-frequency updates.
    - **Network Congestion:** As the number of connected devices grows, bandwidth constraints can lead to network congestion, resulting in delays and degraded performance. To mitigate these challenges, resource optimization must focus on minimizing data transfer by employing techniques such as data compression, intelligent data filtering, and local processing to reduce the amount of information sent to the cloud.

b. Dynamic Workloads

   IoT applications generate unpredictable, fluctuating workloads. In IoT applications, workloads can fluctuate unpredictably due to varying tasks and events from numerous connected devices. Balancing these workloads dynamically is essential to prevent some devices from being overloaded while others remain underutilized. This requires real-time monitoring of each device's capacity and adjusting task distribution accordingly, ensuring optimal resource use and preventing performance issues. Proper workload balancing improves efficiency, reduces latency, enhances reliability, and optimizes energy consumption, particularly in battery-powered devices, making the entire IoT system more responsive and reliable. Balancing tasks between devices dynamically is crucial to prevent overloading some nodes while underutilizing others [11, 17].

c. Geographic Distribution
   Fog and edge nodes are geographically dispersed, making coordination and resource sharing challenging. As the distance between nodes increases, network latency also rises, leading to delays in data communication. Additionally, data transfer costs tend to increase with distance, as more resources are required to send information across longer networks. These factors make it difficult to manage workloads efficiently and maintain fast, cost-effective communication between geographically dispersed devices in fog and edge computing environments. Network latency and data transfer costs increase with distance.
d. Energy Efficiency
   Many edge devices in IoT systems are energy-constrained, often relying on limited battery power for operation. To ensure sustainable and long-term functionality, it is crucial to implement power-aware algorithms that optimize energy usage. These

algorithms dynamically manage device activity, adjust processing loads, and minimize unnecessary data transmission, helping to conserve energy while maintaining performance. By prioritizing energy efficiency, these algorithms extend the lifespan of devices, reduce the need for frequent recharging or replacement, and contribute to the overall sustainability of the IoT system. Many edge devices are energy-constrained, requiring power-aware algorithms to ensure sustainable operation.

e. Security and Privacy

In IoT systems, data processed at the edge often contains sensitive information, making security and privacy a top priority. However, implementing robust security measures like encryption on resource-constrained edge devices poses a significant challenge, as these devices typically have limited processing power, memory, and energy. Balancing the need for strong encryption with the constraints of edge devices requires efficient, lightweight security algorithms that do not compromise the device’s performance or battery life. This delicate balance is crucial to ensuring data privacy while maintaining the functionality and efficiency of the IoT network. Data processed at the edge often contains sensitive information. Implementing security measures like encryption without overloading resource-constrained devices is a significant challenge.

2. **Optimization Techniques and Strategies**

a. **Task Offloading**

Definition: Task offloading involves transferring tasks from resource-constrained edge devices to more powerful fog nodes or cloud data centers based on resource availability, helping to optimize performance
Objective: The primary objectives of task offloading are to minimize:

- **Latency:** Reducing delays by executing tasks on more powerful nodes closer to data centers.
- **Energy Consumption:** Offloading computationally heavy tasks to devices with higher energy resources, thus preserving battery life of edge devices.

  Approaches to Task Offloading:

  1. **Full Offloading:**

     - **Description:** All tasks are sent to the fog nodes or cloud data centers for execution.
     - **Benefit:** Maximizes computational power and resources available in fog/cloud environments, ensuring tasks are processed quickly without overburdening edge devices.
     - **Drawback:** Increases network traffic and data transfer costs, as the entire task has to be transmitted to remote servers. Latency can be an issue if the nodes are far apart.

       Partial Offloading:

- **Description:** Tasks are divided into parts, with some being executed locally on the edge device and others offloaded to fog/cloud nodes.

  - **Benefit:** Strikes a balance between local processing and offloading, optimizing resource use on edge devices while leveraging cloud/fog resources when needed. It can reduce latency and energy consumption depending on the division of tasks.
  - **Drawback:** The complexity of deciding how to divide the tasks efficiently and managing the synchronization between local and offloaded components.

b. **Load Balancing**

Definition: Load balancing involves distributing workloads across multiple nodes in a system to ensure that no single node becomes overloaded, thus preventing bottlenecks and optimizing resource utilization.

Techniques for Load Balancing:

1. **Round-Robin Scheduling:**

    - **Description:** Tasks are distributed sequentially across nodes in a circular manner. Each node receives the next task in the sequence, ensuring an even distribution of work.
    - **Benefit:** Simple and easy to implement, ensuring that all nodes receive an equal share of tasks over time.
    - **Drawback:** It doesn't consider the current load or capacity of the nodes, which could lead to inefficient resource utilization if some nodes are more powerful or underutilized than others.

2. **Priority-Based Scheduling:**

   - **Description:** Tasks with higher QoS requirements are given preference, ensuring that more critical tasks are processed first.
   - **Benefit:** Guarantees that important tasks with stricter deadlines or higher resource needs are prioritized, ensuring optimal performance for critical applications.
   - **Drawback:** Lower-priority tasks may experience delays, especially if there are numerous high-priority tasks, potentially causing imbalance in workload distribution.

3. **Dynamic Allocation:**

   - **Description:** Real-time monitoring of the system's resources (e.g., CPU, memory, and network bandwidth) allows for adaptive allocation of tasks based on the current load on each node.
   - **Benefit:** Maximizes resource utilization by continuously adjusting the task distribution based on the actual workload and available resources, optimizing performance and reducing latency.
   - **Drawback:** Requires more complex algorithms and overhead for continuous monitoring and decision-making, potentially increasing the system's complexity.

c. **Energy-Aware Optimization**
   Definition:

Energy-aware optimization focuses on reducing energy consumption in a system by applying strategies and techniques that minimize power usage without compromising performance, especially in resource-constrained devices.

Techniques for Energy-Aware Optimization:

1. **Energy-Efficient Scheduling:**

   - **Description:** Tasks are allocated to nodes with lower energy consumption, ensuring that power-hungry devices are avoided for tasks that don’t require high processing power.
   - **Benefit:** Reduces overall energy consumption by selecting nodes that are more power efficient for specific tasks, optimizing the system’s energy usage.
   - **Drawback:** May lead to performance degradation if less powerful nodes are consistently chosen for demanding tasks, as they may not be able to execute them efficiently.

2. **Dynamic Voltage and Frequency Scaling (DVFS):**

   - **Description:** The power levels of processors are adjusted in real-time based on workload demands. When the system is under low load, the processor voltage and frequency are reduced to save energy, and when the workload increases, they are ramped up to meet demand.

    - **Benefit:** Helps to balance power consumption with performance requirements, reducing energy usage during low-demand periods without significantly affecting performance.
    - **Drawback:** Continuous scaling can introduce latency and may cause inefficiencies if the system doesn't scale optimally, particularly in systems with unpredictable workloads.
3. **Sleep/Wake Strategies:**
    - **Description:** Devices that are idle or underutilized are put into low-power states (sleep mode), and only woken up when needed, conserving energy during periods of inactivity.
    - **Benefit:** Significantly reduces energy consumption by ensuring that devices consume minimal power when they are not in use, extending battery life and reducing operational costs.
    - **Drawback:** There may be a delay when waking up devices from sleep mode, which can impact system responsiveness, especially in time-sensitive applications.

    These energy-aware optimization techniques aim to balance the need for power conservation with the system's operational demands, improving overall energy efficiency while maintaining functional performance.

d. **Caching and Data Placement**

Definition:

Caching and data placement involve storing frequently accessed data closer to end-users or devices, reducing latency and improving data retrieval times by avoiding long-distance data transfers.

Approaches for Caching and Data Placement:

1. **Proactive Caching:**
    - **Description:** Future data needs are predicted based on user behavior, and data is preemptively stored at nearby nodes to minimize future access delays.
    - **Benefit:** Reduces latency by ensuring that the most likely data to be accessed is already available locally, improving response times and user experience.
    - **Drawback:** Requires accurate prediction algorithms, which can be complex to implement, and may result in unnecessary caching if predictions are not accurate, wasting storage resources.

2. **Data Replication**
    - **Description:** Multiple copies of data are stored across different nodes to ensure reliability and faster access. If one node fails or is too far away, data can be retrieved from another replica.
    - **Benefit:** Enhances data availability and reliability, ensuring that data is accessible even in case of node failures or high-

demand situations, while also improving access speeds by distributing copies across locations.

  - **Drawback:** Increases storage requirements, as maintaining multiple copies of the same data can be resource-intensive and costly, especially in large-scale systems.

  Both approaches focus on improving data access speed and reducing latency, with proactive caching targeting future data needs and data replication focusing on reliability and availability.

e. **AI and Machine Learning Applications:**

AI and machine learning are increasingly being used in IoT and distributed systems to optimize performance, enhance decision-making, and improve efficiency through intelligent algorithms.

Applications for AI and Machine Learning:

1. **Predictive Task Scheduling:**

  - **Description:** Historical data is analyzed to predict future workloads, helping to optimize the distribution of tasks across nodes and reduce delays or resource overloads.
  - **Benefit:** Enhances system efficiency by anticipating workload peaks and dynamically adjusting task scheduling to ensure resources are allocated effectively in advance.
  - **Drawback:** Requires accurate historical data and sophisticated models to ensure

predictions are reliable; poor predictions can lead to misallocation of resources.

2. **Anomaly Detection:**

    - **Description:** AI models are used to identify unusual patterns in resource usage or system behavior, enabling early detection of potential issues or failures before they affect performance.
    - **Benefit:** Helps maintain system reliability by identifying and addressing issues proactively, preventing system downtimes or crashes.
    - **Drawback:** Requires continuous monitoring and accurate model training to minimize false positives or missed anomalies, which can be complex and resource-intensive.

3. **Reinforcement Learning:**

    - **Description:** The system learns optimal strategies for resource allocation by interacting with its environment and receiving feedback on performance, allowing it to adapt and improve over time.
    - **Benefit:** Enables dynamic, self-optimizing systems that can continuously improve resource allocation based on real-time feedback, making them highly adaptive and efficient.
    - **Drawback:** Training reinforcement learning models can be time-consuming and computationally expensive, and the system

may require a long time to converge to optimal solutions.

These AI and machine learning applications enhance system performance by making intelligent predictions, detecting anomalies early, and continuously optimizing resource management, leading to more efficient and reliable IoT and distributed systems.

f. **Network Optimization**

Definition:

Network optimization involves improving network performance by managing resources like bandwidth, communication paths, and traffic flow to enhance system efficiency, reduce delays, and avoid congestion.

Techniques for Network Optimization:

1. **Bandwidth Management:**

    - **Description:** Prioritizes critical data flows over less important ones to ensure that important data is transmitted with minimal delay and congestion.
    - **Benefit:** Ensures that high-priority tasks, such as real-time communication or critical data transfers, are not delayed by less time-sensitive data, improving overall system responsiveness.
    - **Drawback:** Requires accurate identification of priority data and dynamic adjustment of bandwidth allocation, which can increase complexity and overhead.

2. **Edge-to-Edge Communication:**

- **Description:** Enables direct communication between edge nodes (devices at the edge of the network), reducing the need to send all data through centralized servers or cloud infrastructure.
- **Benefit:** Reduces network traffic and reliance on centralized servers, lowering latency and improving system responsiveness by allowing nodes to interact more efficiently.
- **Drawback:** Requires a robust and secure communication protocol between edge devices, and may not always be feasible if nodes are geographically dispersed or lack sufficient connectivity.

  Both techniques aim to optimize network performance by minimizing congestion, reducing latency, and improving data flow, leading to more efficient and responsive system

1.5 Applications of Resource Optimization in Smart Cities

1. **Traffic Monitoring and Control:**

   - **Intelligent Traffic Lights:** Adaptive signal control systems adjust traffic light timings based on real-time traffic flow, reducing congestion and improving traffic flow.
   - **Smart Parking:** Sensors and apps help drivers find available parking spots, minimizing time

spent searching for parking and reducing congestion.

- **Real-Time Traffic Analytics:** Data collected from vehicles, cameras, and sensors helps in predicting traffic patterns and adjusting traffic signals accordingly.
- **Congestion Management:** Resource optimization strategies like carpooling, road pricing, or alternative route suggestions help reduce traffic bottlenecks.

2. **Public Safety and Surveillance Systems**

- **Smart Surveillance:** AI-powered cameras with facial recognition and real-time monitoring can enhance security and detect suspicious activities, leading to faster response times.
- **Emergency Response Optimization:** Using data analytics to predict and optimize the response times of emergency services (police, fire, and ambulance) for better public safety.
- **Predictive Policing:** Analysis of crime data helps in predicting crime hotspots and deploying law enforcement resources efficiently.
- **Disaster Management:** Resource optimization helps in efficient evacuation planning, resource distribution, and coordination during natural disasters.

3. **Waste and Water Management**

- **Smart Waste Collection:** Sensors placed in bins detect fill levels, optimizing waste collection routes and frequency, saving time and fuel.

- **Waste Sorting and Recycling:** Automation in sorting waste helps in better recycling, reducing landfill usage and improving resource utilization.
- **Water Leak Detection:** Sensors and IoT devices monitor water systems in real-time to detect leaks and prevent water wastage.
- **Water Usage Optimization:** Smart meters and data analytics help in managing water distribution efficiently, reducing wastage, and ensuring equitable distribution.
- **Rainwater Harvesting Systems:** Resource optimization strategies can promote the collection and use of rainwater, ensuring sustainable water supply in urban areas.

Applications of Resource Optimization in Healthcare:

1. **Real-Time Health Monitoring (e.g., Wearable Devices)**

    - **Continuous Monitoring:** Wearable devices track vital signs like heart rate, blood pressure, and oxygen levels in real time, providing constant health data.
    - **Early Detection:** These devices help in detecting anomalies (e.g., irregular heartbeats or blood sugar levels) early, enabling prompt intervention and reducing hospital visits.
    - **Personalized Healthcare:** Data collected from wearables can be analyzed to create personalized health plans and optimize treatment for individuals.

    - **Health Behavior Insights:** Wearables provide feedback on daily physical activity, sleep, and stress levels, helping individuals make healthier lifestyle choices.

2. **Remote Diagnostics and Telemedicine:**

    - **Virtual Consultations:** Telemedicine platforms allow patients to consult doctors remotely, reducing the need for in-person visits and optimizing healthcare access, especially in remote or underserved areas.
    - **Remote Monitoring:** Healthcare professionals can monitor patients’ health remotely through connected devices, offering timely interventions without requiring hospital admission.
    - **Diagnostic Tools:** Telemedicine enables access to diagnostic tests (like blood pressure readings, electrocardiogram, etc.) remotely, improving the speed and efficiency of diagnosis.
    - **Access to Specialists:** Patients in rural or distant areas can connect with specialized doctors via telemedicine, optimizing the use of specialized healthcare resources.

3. **Emergency Response Systems**

    - **Optimized Ambulance Dispatch:** Data analytics and AI optimize the allocation of ambulances based on the location of emergencies, ensuring quicker response times.
    - **Real-Time Communication:** Emergency teams can receive real-time information about the patient’s condition (from wearables, for example),

allowing them to prepare in advance and optimize care upon arrival.

- **Health Data Integration:** Emergency response systems integrate patient data from various sources (e.g., wearables and electronic health records), enabling informed decision-making and resource allocation during critical situations.
- **Efficient Resource Allocation:** By analyzing past emergency trends, healthcare resources (staff, equipment) can be allocated more effectively, ensuring that they are ready for high-demand periods.

Predictive Maintenance of Machinery in IIoT [18–20]:

- **Condition Monitoring:** IIoT sensors continuously monitor the health of machines, detecting vibrations, temperature, and other performance indicators that could signal a potential failure.
- **Data Analytics:** Collected data is analyzed to predict when a machine is likely to fail, allowing for timely maintenance and avoiding costly breakdowns.
- **Minimized Downtime:** By addressing issues before they cause significant damage, predictive maintenance ensures minimal production downtime and reduces the need for expensive emergency repairs.
- **Extended Equipment Life:** Optimizing maintenance schedules based on real-time data helps extend the lifespan of machinery, reducing overall capital expenditure.

Real-Time Monitoring of Production Lines:

- **Process Optimization:** IIoT sensors track every aspect of the production line, including speed, quality, and output, ensuring optimal performance and quickly identifying inefficiencies.
- **Quality Control:** Real-time monitoring allows for the immediate detection of defects or deviations from quality standards, enabling quick adjustments and minimizing waste.
- **Automated Reporting:** IIoT systems automatically generate reports on production status, reducing the need for manual checks and improving decision-making.
- **Resource Allocation:** Data from the production line helps managers allocate labor and resources more efficiently, optimizing operational workflows and reducing bottlenecks.

Energy Management in Industrial Facilities:

- **Smart Energy Meters:** IIoT devices track energy usage across different machinery and departments, providing real-time insights into energy consumption patterns.
- **Energy Efficiency Optimization:** By analyzing energy data, IIoT helps identify areas of energy waste and optimize consumption, reducing costs and environmental impact.
- **Demand Response Systems:** IIoT-enabled systems can adjust energy usage during peak demand times, automatically scaling back operations or switching to alternative energy sources to reduce costs.
- **Predictive Power Consumption:** Using data from production schedules and equipment performance, IIoT

systems predict and optimize energy needs, ensuring that energy is used efficiently without excess.

Real-Time Decision-Making for Navigation in Autonomous Vehicles:

- **Dynamic Route Adjustment:** Autonomous vehicles continuously analyze real-time data from sensors, global positioning system, and traffic updates to make decisions about the best routes, avoiding traffic jams and roadblocks.
- **Adaptability to Changing Conditions:** The vehicle adjusts its route and speed based on changing factors like weather conditions, construction zones, or accidents, optimizing travel time.
- **Smart Traffic Navigation:** Using AI and machine learning algorithms, autonomous vehicles can anticipate and respond to traffic signals, road signs, and pedestrian movements, ensuring smooth and efficient navigation.
- **Efficient Energy Use:** By optimizing speed and acceleration patterns, autonomous vehicles can minimize energy consumption, reducing overall fuel usage or battery drain.

Collision Avoidance and Route Optimization:

- **Sensor Integration:** Sensors such as cameras, LIDAR, and radar continuously scan the environment for potential hazards, including other vehicles, pedestrians, and obstacles, to avoid collisions.
- **Real-Time Risk Assessment:** AI algorithms process the data from these sensors to assess risks and make

decisions that avoid accidents, including braking, steering, or adjusting speed.

- **Optimal Path Selection:** By analyzing traffic flow, road conditions, and potential risks, autonomous vehicles can select the safest and most efficient routes, minimizing time and fuel consumption.
- **Vehicle-to-Vehicle Communication:** Vehicles can share information about their speed, position, and intentions with each other, helping to prevent accidents and optimize traffic flow.

Communication between Vehicles and Infrastructure (V2X):

- **Vehicle-to-Infrastructure Communication:** Autonomous vehicles can communicate with traffic signals, road signs, and other infrastructure, receiving real-time updates on road conditions, signal changes, and traffic patterns to optimize their movement.
- **Coordinated Traffic Flow:** V2X communication helps in synchronizing vehicle movements with traffic management systems, reducing congestion and ensuring smoother traffic flow.
- **Smart Parking and Charging Solutions:** V2X can enable vehicles to find parking spaces or charging stations efficiently, reducing time spent searching and optimizing resource usage.
- **Data Sharing for Better Planning:** Autonomous vehicles can share data with city infrastructure systems, helping urban planners optimize road layouts, signal timings, and public transportation strategies based on real-time traffic patterns.

Personalized Marketing and Recommendations in Retail and Commerce:

- **Data-Driven Insights:** AI and machine learning algorithms analyze customer behavior, purchase history, and preferences to deliver personalized marketing content and product recommendations.
- **Targeted Advertising:** Retailers use customer data to create highly targeted advertising campaigns, ensuring that marketing efforts reach the right audience with relevant products, improving conversion rates.
- **Dynamic Pricing:** Prices are optimized based on customer behavior, demand fluctuations, and competitor pricing, ensuring the most competitive pricing for consumers while maximizing profit.
- **Loyalty Programs:** Personalized offers and promotions based on purchasing patterns encourage customer loyalty and increase repeat sales.

Inventory Management and Supply Chain Optimization:

- **Demand Forecasting:** Predictive analytics help retailers forecast demand accurately, ensuring that the right amount of stock is available at the right time, reducing excess inventory or stockouts.
- **Automated Replenishment:** Using real-time sales data and inventory levels, systems automatically trigger restocking orders, preventing supply shortages and reducing human intervention.
- **Warehouse Management:** IoT and AI-driven systems optimize storage, shelf management, and picking processes in warehouses, reducing time and effort involved in product retrieval.

- **Supply Chain Visibility:** Advanced tracking and analytics provide real-time visibility into the supply chain, helping retailers identify inefficiencies and improve resource allocation, reducing delays and costs.

Automated Checkout Systems:

- **Self-Checkout Stations:** Automated checkout systems allow customers to scan and pay for items without cashier assistance, reducing wait times and improving customer satisfaction.
- **Radio-Frequency Identification (RFID) and IoT Technology:** Items are automatically detected and added to the customer’s cart using RFID tags, making the checkout process seamless and faster.
- **Mobile Payment Solutions:** Customers can use mobile apps to scan and pay for products, further streamlining the checkout process and reducing reliance on traditional payment systems.
- **Fraud Prevention:** AI algorithms analyze transaction data in real time to detect unusual patterns and prevent fraudulent transactions, ensuring secure and efficient checkouts.

3. **Future Directions in Resource Optimization**
   a. **Edge-Cloud Collaboration [12, 21]**
      1. **Seamless Integration of Edge, Fog, and Cloud Resources:**

         - **Distributed Computing:** Edge, fog, and cloud resources work together to distribute computational tasks more efficiently, reducing latency by processing data closer to the source (edge), while leveraging the cloud for heavy computations.

- **Optimized Resource Allocation:** Workloads are dynamically assigned based on the proximity to the data source and resource availability, ensuring that time-sensitive tasks are handled at the edge, while non-urgent tasks are processed in the cloud.
- **Multi-Tier Architecture:** Edge devices handle real-time data collection and processing, fog nodes manage local data storage and intermediate processing, and the cloud handles large-scale data storage, analytics, and long-term processing.
- **Cost Efficiency:** By offloading nonessential or less time-sensitive tasks to the cloud, and keeping latency-sensitive tasks at the edge, this collaboration optimizes the cost of infrastructure, balancing local resource usage and cloud computing expenses.

2. **Real-Time Data Processing and Low Latency:**

   - **Edge for Low-Latency Applications:** Edge devices handle time-sensitive tasks such as real-time sensor data processing, enabling faster decision-making in applications like autonomous vehicles, industrial IoT, and smart cities.
   - **Fog for Localized Data Filtering:** Fog nodes help by filtering and aggregating data from edge devices before sending it to the

cloud, reducing the volume of data transmitted and improving overall system responsiveness.

- **Cloud for Heavy Data Analytics:** The cloud can manage more resource-intensive tasks like big data analytics, machine learning model training, and long-term data storage, while edge and fog nodes handle immediate processing needs.

3. **Dynamic Load Balancing and Scalability**

- **Real-Time Load Distribution:** Resource optimization algorithms dynamically allocate tasks between edge, fog, and cloud based on current load and resource availability, ensuring that no single tier is overwhelmed and resources are used efficiently.
- **Scalable Infrastructure:** As demand fluctuates, the system can scale cloud resources up or down automatically, ensuring that performance and cost are continuously optimized.
- **Elasticity of Resources:** Edge devices and fog nodes can act as intermediary processors that help balance the load between the cloud and edge, adapting in real time to changing demands for both performance and storage.

4. **Data Privacy and Security**

- **Data Localization at the Edge:** Sensitive data can be processed at the edge or fog level, minimizing the need to transmit data to the cloud and enhancing privacy and security for applications like healthcare, finance, and government.

  - **Distributed Security Measures:** Edge, fog, and cloud resources collaborate in implementing security protocols, ensuring that data is protected at all points in the data pipeline (local processing, transit, and cloud storage).
  - **Access Control and Authentication:** Using edge and fog nodes to handle authentication and authorization can reduce the risk of unauthorized access by enforcing policies closer to the data source.

5. **Energy Efficiency and Sustainability:**

  - **Energy-Aware Resource Management:** Edge and fog nodes are often more energy-efficient for localized processing, reducing the need for continuous cloud data transfers and minimizing energy consumption.
  - **Smart Energy Allocation:** The collaboration between edge, fog, and cloud can be optimized to use energy resources in the most sustainable way, considering the energy costs of local devices and cloud-based data centers.
  - **Reduced Data Transmission Costs:** By processing data locally at the edge or fog levels, the need for large-scale data transmission to the cloud is reduced, leading to lower network energy consumption and cost.

6. **AI-Driven Optimization and Automation:**

    - **Autonomous Decision-Making:** AI algorithms can autonomously decide when to offload tasks to the edge or cloud based on current system performance, data urgency, and resource availability, improving overall efficiency.
    - **Self-Optimizing Networks:** Edge-cloud collaboration systems can use machine learning to predict resource usage patterns, automatically optimizing task distribution and energy consumption across all three layers.
    - **Predictive Maintenance:** AI can be used to predict when hardware in edge or fog nodes may fail, allowing for proactive resource allocation to prevent system downtime.

b. **Federated Learning**

1. **Training AI Models across Distributed Edge Devices:**

    - **Decentralized Model Training:** Federated learning allows AI models to be trained on distributed datasets located on edge devices, avoiding the need to centralize data in a single server.
    - **Efficient Use of Local Data:** Each edge device trains the model locally using its own data, contributing to a shared global model without transferring raw data to the cloud.

- **Collaborative Learning:** Multiple devices collaborate to improve the model’s accuracy by aggregating updates, leveraging diverse datasets across distributed systems.

2. **Enhancing Privacy:**

   - **Data Locality:** Sensitive data, such as medical records or personal user information, remains on the edge devices, significantly reducing the risk of data breaches.
   - **Secure Aggregation:** Federated learning employs techniques like homomorphic encryption and differential privacy to securely aggregate model updates, ensuring that no individual device’s data can be reconstructed.
   - **Compliance with Privacy Regulations:** Federated learning aligns with privacy laws such as General Data Protection Regulation (GDPR) and Health Insurance Portability and Accountability Act (HIPAA) by keeping personal data decentralized and within users’ control.

3. **Reducing Bandwidth Usage:**

   - **Model Update Transmission:** Only model updates (e.g., gradients or weights) are transmitted to a central server for aggregation, instead of raw data, reducing the volume of data exchanged over the network.

- **Edge Device Optimization:** By training locally, federated learning minimizes the need for frequent data uploads and downloads, conserving network bandwidth and reducing latency.
- **Adaptive Synchronization:** Communication between edge devices and the central server can be scheduled intelligently to further optimize bandwidth usage, such as during off-peak hours or in batch updates.

4. **Improving Resource Efficiency:**

   - **Scalable Learning Framework:** Federated learning enables scalable training by utilizing the idle computational power of edge devices, reducing the reliance on centralized cloud infrastructure.
   - **Energy Efficiency:** By processing data locally, edge devices consume less energy compared to continuously transmitting data to the cloud for centralized training.
   - **Task Prioritization:** Federated learning frameworks can prioritize resource allocation for devices with greater availability or processing power, ensuring optimal use of distributed resources.

5. **Applications in Real-World Scenarios:**

   - **Healthcare:** Federated learning facilitates collaborative model training across hospitals or devices without sharing sensitive patient data, enabling advancements in medical research and diagnosis.

- **Smartphones and IoT:** Personalization features, such as predictive text or recommendation systems, are trained locally on users' devices, enhancing user experience while preserving privacy.
- **Autonomous Vehicles:** Edge devices in autonomous vehicles use federated learning to collaboratively improve navigation, object detection, and decision-making algorithms, while keeping location and sensor data private.
- **Finance:** Federated learning allows financial institutions to collaboratively develop fraud detection and risk assessment models without sharing proprietary customer data.

c. **6G and Beyond [22, 23]**

1. **Faster Communication and Enhanced Connectivity:**

   - **Ultra-High Data Speeds:** 6G is expected to deliver data speeds up to 100 times faster than 5G, enabling seamless communication for data-intensive applications such as augmented reality, virtual reality, and holographic telepresence.
   - **Low Latency:** With latencies as low as 1 millisecond or less, 6G will support real-time applications like autonomous vehicles, remote surgeries, and industrial automation with near-instant responsiveness.

- **Global Coverage:** Integration of terrestrial and satellite networks in 6G will provide ubiquitous connectivity, even in remote and underserved areas, ensuring optimal resource usage across the globe.

2. **Advanced Resource Allocation:**

    - **AI-Driven Network Management:** AI will dynamically allocate network resources based on real-time demand, optimizing bandwidth usage and ensuring high-quality service delivery.
    - **Network Slicing:** 6G networks will support highly customizable network slices tailored for specific applications, such as low-latency gaming or high-reliability industrial automation, optimizing the use of network resources.
    - **Spectrum Optimization:** Leveraging higher frequency bands (e.g., terahertz frequencies), 6G will enhance spectrum efficiency, enabling better utilization of available bandwidth and reducing congestion.

3. **Edge-Cloud Integration:**

    - **Enhanced Edge Computing:** With 6G's ultra-fast speeds, edge devices can process and share data locally with minimal latency, reducing reliance on centralized cloud infrastructure.

- **Collaborative Resource Sharing:** 6G will enable seamless collaboration between edge, cloud, and fog computing systems, ensuring that computing tasks are distributed efficiently based on resource availability and application requirements.
- **Smart Energy Allocation:** Optimized energy usage across edge and cloud systems will ensure sustainable resource utilization while maintaining high performance.

4. **Support for Emerging Applications:**

   - **Extended Reality (XR):** 6G will provide the ultra-fast and low-latency connectivity needed to support immersive experiences in XR, enhancing applications in gaming, education, and training.
   - **Smart Cities and IoT:** The massive machine-type communication capabilities of 6G will optimize resource usage in IoT ecosystems, including smart grids, traffic systems, and public safety networks.
   - **Holographic Communication:** Holographic telepresence will become viable with 6G’s ability to handle vast amounts of data in real time, revolutionizing remote collaboration and communication.

5. **Sustainability and Green Technology:**

   - **Energy-Efficient Networks:** 6G will focus on designing energy-efficient communication protocols and hardware to minimize the carbon footprint of wireless technologies.

  - **Dynamic Power Allocation:** AI-driven systems will adjust power usage dynamically based on network load, reducing energy consumption during low-traffic periods.
  - **Integration with Renewable Energy Sources:** 6G infrastructure will likely integrate renewable energy sources, contributing to the sustainability goals of next-generation networks.

6. **Enhanced Security and Privacy:**

  - **Quantum Communication Integration:** 6G is expected to incorporate quantum communication technologies for ultra-secure data transmission, protecting sensitive information in critical applications.
  - **Decentralized Security Models:** Advanced encryption and blockchain-based technologies will ensure secure communication and resource allocation in 6G networks.
  - **Privacy-Preserving Mechanisms:** Data sharing and processing in 6G will leverage techniques like federated learning and edge-based data analytics to enhance privacy and security.

d. **Blockchain Integration**

1. **Decentralized Resource Management:**

- **Distributed Resource Allocation:** Blockchain enables decentralized systems where edge devices manage resources collaboratively without relying on a central authority, ensuring transparency and trust.
- **Dynamic Task Distribution:** Smart contracts can automate the allocation of computational tasks among edge devices based on their availability and capacity, optimizing the use of distributed resources.
- **Efficient Resource Sharing:** Blockchain facilitates peer-to-peer resource sharing, allowing devices to exchange storage, processing power, and bandwidth dynamically and securely.
- **Global Scalability:** Decentralized systems powered by blockchain can scale seamlessly, making them ideal for applications in IoT, where devices are widely distributed.

2. **Secure Transactions among Edge Devices**

   - **Immutable Ledger:** Blockchain ensures that all transactions between edge devices are recorded on an immutable ledger, providing a reliable and tamper-proof record of resource usage and exchanges.
   - **Trustless Collaboration:** Devices can interact and share resources without needing prior trust, as blockchain enforces trust through consensus mechanisms and cryptographic verification.

- **Authentication and Authorization:** Blockchain can be used to securely authenticate and authorize edge devices, preventing unauthorized access and enhancing network security.
- **Secure Payments:** Devices can use cryptocurrency or token-based systems for automated microtransactions, enabling seamless payment for resources like computing power, storage, or energy.

3. **Enhanced Privacy and Data Integrity**

   - **Decentralized Data Management:** Data collected by edge devices remains decentralized, reducing the risk of data breaches and enhancing user privacy.
   - **Data Provenance:** Blockchain provides a transparent history of data generation and usage, ensuring that data integrity is maintained and its origin is verifiable.
   - **Privacy-Preserving Mechanisms:** Combining blockchain with advanced cryptographic techniques like zero-knowledge proofs allows transactions and data exchanges to remain private while still being verifiable.

4. **Energy and Cost Efficiency**

   - **Incentivized Participation:** Blockchain incentivizes devices to contribute resources by rewarding them with tokens or credits, encouraging efficient use of idle resources.

- **Optimized Consensus Mechanisms:** New blockchain technologies (e.g., proof-of-stake or proof-of-authority) reduce the energy costs associated with traditional consensus methods like proof-of-work, making them more suitable for edge networks.
- **Cost-Effective Management:** By eliminating intermediaries in resource transactions, blockchain reduces operational costs, making resource optimization more affordable.

5. **Applications in IoT and Edge Computing:**

    - **Smart Grids:** Blockchain facilitates the secure and decentralized exchange of energy among smart grid participants, optimizing energy distribution and usage.
    - **Healthcare:** Blockchain ensures secure sharing and management of sensitive healthcare data among IoT devices, enhancing patient privacy and data reliability.
    - **Supply Chain Optimization:** Blockchain tracks the movement of goods in real time, ensuring transparency and optimizing logistics resources across distributed networks.
    - **Autonomous Vehicles:** Vehicles can securely exchange data and resources with other vehicles and infrastructure, enabling efficient traffic management and navigation.

e. **Quantum Computing**

1. **Quantum Algorithms for Faster Optimization:**

   - **Solving Complex Optimization Problems:** Quantum algorithms, such as the **Quantum Approximate Optimization Algorithm (QAOA)**, can solve combinatorial and resource allocation problems exponentially faster than classical algorithms.
   - **Real-Time Decision-Making:** Quantum computing enables rapid processing of vast amounts of data, making it ideal for dynamic and time-sensitive resource optimization, such as traffic management or supply chain logistics.
   - **Parallel Problem Solving:** Quantum superposition allows multiple solutions to be evaluated simultaneously, accelerating the identification of optimal configurations for resource allocation.

2. **Enhanced Efficiency in Resource Management:**

   - **Improved Scheduling:** Quantum algorithms can optimize resource scheduling, such as allocating computing power or bandwidth in data centers or edge networks, with unmatched precision.
   - **Reduced Energy Consumption:** By identifying optimal solutions faster, quantum computing minimizes

computational overhead, reducing the energy required for processing large-scale optimization tasks.

- **Supply Chain Optimization:** Quantum computing can model and optimize supply chains more efficiently by analyzing all variables simultaneously, leading to reduced costs and delays.

3. **Handling Large-Scale Data**

   - **Quantum Machine Learning:** Integrating quantum computing with machine learning models enables faster training and prediction for resource optimization tasks, such as predictive maintenance or anomaly detection.
   - **Scalable Optimization:** Quantum systems can handle the optimization of extremely large datasets, which are common in IoT, autonomous systems, and smart cities.
   - **Advanced Simulations:** Quantum computers excel at simulating complex systems, such as climate models or energy grids, allowing for precise optimization of resources within these systems [24–26].

4. **Applications in Various Domains:**

   - **Smart Cities:** Quantum algorithms can optimize urban infrastructure, such as traffic flow, energy distribution, and waste management, in real time, enhancing efficiency.

- **Healthcare:** Quantum computing can optimize the allocation of medical resources, such as hospital beds, equipment, and staff, especially during emergencies or pandemics.
- **Telecommunications:** Quantum computing can optimize network routing, bandwidth allocation, and spectrum management to enhance performance and reduce latency.
- **Finance:** Portfolio optimization, risk assessment, and fraud detection can be performed faster and more accurately with quantum algorithms.

5. **Addressing Challenges in Quantum Resource Optimization**

- **Noise and Error Correction:** While quantum systems are prone to errors, advancements in quantum error correction are improving the reliability of quantum optimization solutions.
- **Hybrid Quantum-Classical Models** Combining quantum computing with classical systems allows organizations to leverage the strengths of both, ensuring practical and scalable resource optimization.
- **Scalability of Quantum Hardware:** As quantum hardware matures, its capability to handle larger and more complex optimization problems will grow, making it a game-changer for resource management.

6. **Potential Transformative Impacts**

- **Disruptive Advancements:** Quantum computing’s ability to solve previously intractable problems will redefine how industries approach resource optimization, unlocking new levels of efficiency and innovation.
- **Cross-Industry Benefits:** From logistics to energy, healthcare, and beyond, quantum-powered optimization will revolutionize resource allocation strategies across all major sectors.
- **Ethical Resource Allocation:** Quantum computing can model fair and equitable resource distribution strategies, addressing global challenges like energy scarcity and resource inequality.

1.6 Conclusion

Resource optimization lies at the heart of fog and edge computing, serving as the key to meeting the challenges posed by latency-sensitive, resource-intensive applications. By bringing computation closer to the data source, these paradigms reduce latency, improve real-time responsiveness, and enhance overall performance. Advanced strategies such as task offloading, load balancing, and energy-efficient computation ensure that resources are utilized effectively, minimizing wastage and maximizing output.

Emerging technologies like AI, blockchain, quantum computing, and federated learning further elevate the potential of resource optimization. AI-driven systems enable dynamic, predictive resource allocation, while blockchain ensures secure and transparent management in decentralized environments. Quantum algorithms promise breakthroughs in solving complex optimization problems, and federated learning combines privacy with efficiency by training models across distributed devices without sharing raw data.

The integration of next-generation wireless technologies, such as 6G, strengthens resource optimization by enabling seamless connectivity, faster data transmission, and intelligent edge-cloud collaboration. Moreover, the emphasis on energy efficiency and sustainability ensures that fog and edge systems not only perform better but also contribute to a greener technological future.

By leveraging these strategies and innovations, fog and edge computing can achieve unparalleled efficiency, scalability, and sustainability, making them indispensable in driving the future of technology and modern applications.

References

1. Soni, P. K., and Dhurwe, H. "Challenges and Open Issues in Cloud Computing Services." *Advanced Computing Techniques for Optimization in Cloud*, 1st ed., CRC Press LLC, 2024. Routledge Tylor and Francis Group. www.routledge.com/Advanced-Computing-Techniques-for-Optimization-in-Cloud/Madhusudhan-Gupta-Rawat/p/book/9781032600079⏎
2. Rawat, P. S., and Soni, P. K. "Resource Management in Cloud Using Nature-Inspired Algorithm." *Advanced Computing Techniques for Optimization in Cloud*, 1st ed., CRC Press LLC, 2024. Routledge Tylor and Francis Group. www.routledge.com/Advanced-Computing-Techniques-for-

Optimization-in-Cloud/Madhusudhan-Gupta-Rawat/p/book/9781032600079⏎

3. Rawat, P. S., and Soni, P. K. “Efficient Virtual Machine Allocation Techniques Based on Hybrid Approach.” *Advanced Computing Techniques for Optimization in Cloud,* 1st ed., CRC Press LLC, 2024. Routledge Tylor and Francis Group. www.routledge.com/Advanced-Computing-Techniques-for-Optimization-in-Cloud/Madhusudhan-Gupta-Rawat/p/book/9781032600079
4. Rawat, P. S., and Soni, P. K. “Roles of Soft Computing Methodologies in Service-Oriented Computing.” *Soft Computing Principles and Integration for Real-Time Service-Oriented Computing*, 1st ed., Auerbach Publications, 2024, p. 83. www.routledge.com/Soft-Computing-Principles-and-Integration-for-Real-Time-Service-Oriented/Gupta-Kumar-Saini-Zia/p/book/9781032551883
5. Rawat, P. S., and Soni, P. K. “Smart IoT System for Agricultural Production Improvement and Machine Learning-Based Prediction.” *Soft Computing Principles and Integration for Real-Time Service-Oriented Computing*, 1st ed., Auerbach Publications, 2024, p. 174. www.routledge.com/Soft-Computing-Principles-and-Integration-for-Real-Time-Service-Oriented/Gupta-Kumar-Saini-Zia/p/book/9781032551883⏎
6. Rawat, P. S., et al. “Landslide Monitoring Using IoT Systems with Cloud Platform.” *10th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)*, 2023. IEEE. https://ieeexplore.ieee.org/document/10434461⏎
7. Gupta, R., et al. “Resource Optimization in Fog Computing for IoT Applications.” *IEEE Transactions on Industrial*

Informatics, vol. 17, no. 3, pp. 2018–2030, 2021. https://ieeexplore.ieee.org/document/9053761⏎

8. Soni, P. K., and Dhurwe, H. “Introduction to Next Generation Optimization in Cloud Computing Services.” *Advanced Computing Techniques for Optimization in Cloud,* 1st ed., CRC Press LLC, 2024. Routledge Tylor and Francis Group. www.routledge.com/Advanced-Computing-Techniques-for-Optimization-in-Cloud/Madhusudhan-Gupta-Rawat/p/book/9781032600079⏎
9. Rawat, P. S., Soni, P. K., & Gupta, P. (2024, May). Performance Analysis of Intelligent Surveillance System in a Fog Computing Environment. In *International Conference on Data & Information Sciences* (pp. 425–435). Singapore: Springer Nature Singapore.⏎
10. Shi, W., and Dustdar, S. “The Promise of Edge Computing.” *Computer,* vol. 49, no. 5, pp. 78–81, 2016. https://ieeexplore.ieee.org/document/7457397⏎
11. Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. “Fog Computing and Its Role in the Internet of Things.” *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, 2012, pp. 13–16. https://dl.acm.org/doi/10.1145/2342509.2342513⏎
12. Mustapha, S. D. S., and Gupta, P. “DBSCAN Inspired Task Scheduling Algorithm for Cloud Infrastructure.” *Internet of Things and Cyber-Physical Systems,* vol. 4, pp. 32–39, 2024.⏎
13. Gupta, P., Rawat, P. S., Kumar Saini, D., Vidyarthi, A., and Alharbi, M. “Neural Network Inspired Differential Evolution Based Task Scheduling for Cloud Infrastructure.” *Alexandria Engineering Journal,* vol. 73, pp. 217–230, 2023.

14. Madhusudhan, H. S., Gupta, P., Saini, D. K., and Tan, Z. “Dynamic Virtual Machine Allocation in Cloud Computing Using Elephant Herd Optimization Scheme.” *Journal of Circuits, Systems and Computers*, vol. 32, no. 11, Art. 2350188, 2023.
15. Rawat, P. S., Gaur, S., Barthwal, V., Gupta, P., Ghosh, D., Gupta, D., and Rodrigues, J. J. C. “Efficient Virtual Machine Placement in Cloud Computing Environment Using BSO-ANN Based Hybrid Technique.” *Alexandria Engineering Journal,* vol. 110, pp. 145–152, 2025.⏎
16. Mao, Y., You, C., Zhang, J., Huang, K., and Letaief, K. B. “A Survey on Mobile Edge Computing: The Communication Perspective.” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017. https://ieeexplore.ieee.org/document/7929965
17. Deng, R., Lu, R., Lai, C., Luan, T. H., and Liang, H. “Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption.” *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1171–1181, 2016. https://ieeexplore.ieee.org/document/7558165⏎
18. HS, M., and Gupta, P. “Federated Learning Inspired Antlion Based Orchestration for Edge Computing Environment.” *PLoS One*, vol. 19, no. 6, Art. e0304067, 2024.⏎
19. Gupta, P., Anand, A., Agarwal, P., and McArdle, G. “Neural Network Inspired Efficient Scalable Task Scheduling for Cloud Infrastructure.” *Internet of Things and Cyber-Physical Systems,* vol. 4, pp. 268–279, 2024.
20. Morabito, R., Petrolo, R., Loscrí, V., and Mitton, N. “LEGIoT: A Lightweight Edge Gateway for the Internet of Things.” *Future Generation Computer Systems*, vol. 81, pp. 1–15, 2018.

[www.sciencedirect.com/science/article/pii/S0167739X17313727⏎](www.sciencedirect.com/science/article/pii/S0167739X17313727)

21. Bittencourt, L. F., et al. “The Internet of Things, Fog and Cloud Continuum: Integration and Challenges.” *Internet of Things*, vol. 3, pp. 134–155, 2018. [www.sciencedirect.com/science/article/pii/S2542660518301197⏎](www.sciencedirect.com/science/article/pii/S2542660518301197)
22. Satyanarayanan, M., et al. “The Role of Edge Computing in 5G.” *IEEE Consumer Electronics Magazine*, vol. 9, no. 1, pp. 24–29, 2020. [https://ieeexplore.ieee.org/document/8955088⏎](https://ieeexplore.ieee.org/document/8955088)
23. Dash, S. K., Mohapatra, S., and Pattnaik, P. K. “A Survey on Applications of Wireless Sensor Network Using Cloud Computing.” *International Journal of Computer Science and Engineering Survey*, vol. 2, no. 1, pp. 37–54, 2011. [https://aircconline.com/ijcses/V2N1/2111ijcses04.pdf⏎](https://aircconline.com/ijcses/V2N1/2111ijcses04.pdf)
24. Chiang, M., and Zhang, T. “Fog and IoT: An Overview of Research Opportunities.” *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, 2016. [https://ieeexplore.ieee.org/document/7553031⏎](https://ieeexplore.ieee.org/document/7553031)
25. Premsankar, G., Di Francesco, M., and Taleb, T. “Edge Computing for the Internet of Things: A Case Study.” *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1275–1284, 2018. https://ieeexplore.ieee.org/document/8017561
26. Gupta, H., Vahid Dastjerdi, A., Ghosh, S. K., and Buyya, R. “iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in the Internet of Things, Edge and Fog Computing Environments.” *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017. [https://onlinelibrary.wiley.com/doi/full/10.1002/spe.2509⏎](https://onlinelibrary.wiley.com/doi/full/10.1002/spe.2509)

Chapter 2

Artificial Intelligence Inspired Scheduling in Edge Computing

Kavitha D. N., Gayana J. Kumar, and Vedavathi N.

DOI: 10.1201/9781003610168-2

2.1 Introduction

Cloud computing, which is comparatively distant from Internet of Things (IoT) devices, has struggled in recent years to accommodate applications with stringent requirements for mobility or delay, like real-time object detection and real-time video streaming [1, 2]. These shortcomings in cloud computing can now be addressed using the edge computing (EC) paradigm. Mobile users create tasks in EC, which are then transferred to edge nodes (edge servers) for processing.

Efficient task scheduling has drawn a lot of interest since nodes are placed at the periphery of networks with constrained processing power. Significant energy consumption reductions and an improvement in service level agreement violations (SLAV) in EC can be accomplished through effective job scheduling.

Researchers have improved and suggested optimization techniques based on linear, dynamic, greedy, and other programming techniques in order to create a better scheduling approach. Deep reinforcement learning (DRL)-based optimization techniques have received increased research interest recently due to the advancement of deep learning. DRL-based algorithms can obtain a more effective scheduling approach when compared to conventional optimization techniques. However, DRL-based algorithms continue to confront two significant obstacles because of the intricate EC contexts found in the real world [3, 4].

First, in order to save energy or satisfy processing requirements, certain nodes in real-world EC scenarios can be dynamically turned on or turned off. Additionally, there is a dynamic shift in the quantity of active and pending tasks. However, the dynamic rise or fall of nodes and jobs is too much for the DRL-based algorithms that are currently in use. Thus, a significant difficulty is how to efficiently depict the dynamic environment in order to adjust to the fluctuation of nodes and tasks and enhance the scalability of the algorithm.

Second, genuine EC scenarios involve many diverse nodes and tasks, whereas existing DRL algorithms presume an environment with comparatively minor state sets and action sets. This necessitates handling a lot of data. Additionally, every task or node has its own properties, such as tasks with resource requests, allocations, etc., and nodes with resource abilities, etc.

These lead to dimensional disasters and make it challenging to store all the data directly. Thus, another significant difficulty is how to efficiently store all environmental data, conserve computational power, and make quick choices.

Techniques for data compression and dimensionality reduction are required to address the aforementioned issues. The issue of dimensional disasters can be resolved and the state of the environment can be effectively represented by reducing and connecting a high-dimensional state space to a low-dimensional vector space while also minimizing the impact on the relationships between the original data.

To decrease the state dimension of DRL and dynamically represent the EC environment, representation learning is presented and enhanced in this study. Similar to vertices and edges in graphs, relationships and entities are the primary building blocks of representation learning. Furthermore, relations are typically understood to be translations acting on the entities' low-dimensional representations.

The entities and relations in EC are initially established with two levels prior to using representation learning as shown in Figure 2.1 [5].

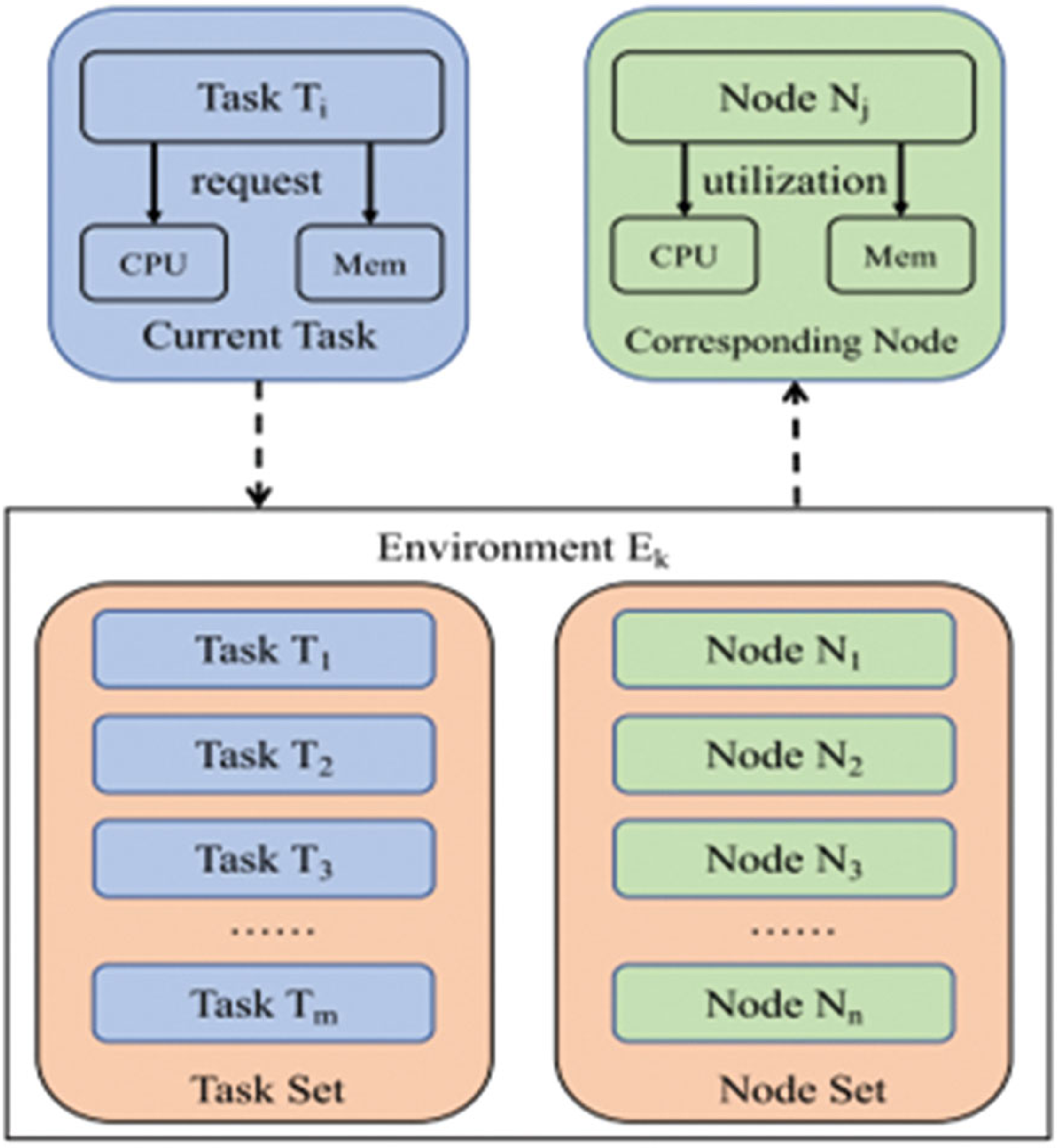


Figure 2.1 Relationship between tasks and nodes (two levels).⏎

The first-level entities are the tasks and nodes themselves, which are propertyless. Regarding the first-level entities, resource needs describe the relationships among tasks and their properties, whereas resource usage defines the relationships between nodes and their properties. Next, the environment—which comprises the collection of tasks and nodes—and the tasks and nodes with matching qualities are designated as the second-level entities.

When a task enters a certain environment, the relationship among the second-level entities is identified as the scheduling decision, that is, the relationship is extracted to obtain the matching node. This approach is highly scalable because of representation learning, which also makes it simple to construct various entities and relations for newly arriving tasks and devices in EC as well as for device-to-device communications. Vectors and distances between vectors can then be used to represent the entities and relationships, respectively.

A new model for representation TransEC, which translates the EC tasks, nodes, and environment to various vector subspaces, is presented using the entities and relations previously mentioned.

In particular, the tasks and nodes are initially embedded into various sub-spaces using embedding layers. Then, the potential features of the environment are retrieved using linear transformations and convolution neural networks, and the knowledge is mapped to a different sub-space. Lastly, our TransEC model is used to extract the relationships between second-level entities.

Scheduling decisions in a dynamic and large-scale EC environment can be resolved by DRL with the aid of the TransEC model, which effectively reduces the state dimension. The Deep Q-Learning (DQL) algorithm is selected among many DRL algorithms due to its ability to facilitate quick decision-making.

Additionally, the TransEC-DQL algorithm's training frequency is adjusted to account for both the state information surrounding this time slot as well as the state knowledge of the present time slot, which involves the distributions of task and node resource situations. Also double Q-learning is used to enhance DQL performance even more [6]. Many similar solutions using hybrid solutions are proposed in references [7–12].

Literature survey:

The paper "Workflow Makespan Minimization for Partially Connected Edge Network: A Deep Reinforcement Learning-Based Approach" [13] takes on the problem of improving workflow execution in partly connected edge networks by combining critical path analysis, dynamic task sorting, and a unique reinforcement learning-based workflow embedding (RLWE) scheme. By modeling the scheduling process as a Markov decision process, the technique coordinates task placement and multipath routing to reduce makespan while prioritizing path quality and congestion avoidance using a disjoint subpath selection mechanism. While RLWE outperforms traditional methods such as DPE, FixDoc, and HEFT in terms of reducing computational delays and optimizing resource usage across a wide range of network configurations and workflows, the study highlights challenges such as task scheduling and multipath routing complexity, scalability concerns for hybrid cloud–edge systems, and limited real-time dynamic congestion management. Despite these limitations, experimental data show that RLWE is successful, with considerable makespan reductions and enhanced resource usage.

The paper "Deep Reinforcement Learning-Based Workload Scheduling for Edge Computing" [14] proposes a new technique to optimize workload scheduling in dynamic EC systems with resource limits and significant unpredictability. It uses a multi-tier architecture and Deep Q-Networks (DQN) to balance workloads, improve service time, and reduce the number of unsuccessful tasks. The technique expresses the scheduling issue as a Markov Decision Process (MDP), which includes state and action spaces as well as a reward function aimed at reducing delays. The DQN model improves resource allocation decision-making by combining experience replay with neural networks. Simulations on EdgeCloudSim indicated that the technique outperformed alternatives such as Deep Deterministic Policy Gradient (DDPG) and Proximal Policy Optimization (PPO), notably in terms of service time, virtual machine usage, and task failure rates, while preserving efficiency under high device density. However, the study identifies shortcomings, such as inadequate scalability to large-scale or hybrid systems, insufficient modeling of real-time environmental dynamics, and a lack of attention on energy efficiency. Despite these constraints, the findings show that the suggested technique for scheduling edge computing workloads is resilient and effective.

The article “Resource Scheduling in Edge Computing: A Survey” [15] examines advances in resource management within EC settings to meet the rising needs of IoT applications. It investigates three major topics: compute offloading, resource allocation, and resource supply in both centralized and distributed techniques. The study divides scheduling models into three categories: things–edge, things–edge–cloud, and edge–edge collaborations. It investigates binary and partial offloading; resource allocation strategies for computing, communication, and storage; and dynamic provisioning methods such as task assignment and edge resource placement. While emphasizing the efficacy of these technologies in enhancing quality of service (QoS) and quality of experience, the study also finds limitations such as inadequate real-time adaptation, poor energy efficiency, and scalability issues for big networks. By combining performance indicators such as latency, energy usage, and cost, the report emphasizes the necessity for further research into dynamic, scalable, and energy-efficient frameworks.

The study “Representation and Reinforcement Learning for Job Scheduling in Edge Computing” [5] provides a new framework that combines representation learning with DRL to address dynamic job scheduling in EC. This approach addresses difficulties such as dynamic node changes and dimensional catastrophes by using a TransEC model to describe entities (tasks and nodes) and their interactions in a compressed vector space, as well as a TransEC-DQL algorithm to make scheduling choices. Key approaches include embedding task and node characteristics in vector subspaces, using convolutional neural networks to describe the environment, and using a double Q-learning-based DRL strategy for scalable, dynamic decision-making. Experimental results from real-world datasets show that the methodology outperforms baseline methods in terms of decreasing energy usage by 18.04% and SLAV by 9.94%. However, the study reveals opportunities for enhanced network architectures to boost feature extraction and scalability even further.

The paper “Multiagent Meta-Reinforcement Learning for Optimized Task Scheduling in Heterogeneous Edge Computing Systems” [16] focuses on the challenges of efficient computation task scheduling in mobile–edge computing (MEC) environments with resource constraints, spectrum congestion, and nonstationarity. Using a multiagent MDP paradigm, the paper defines the problem as a noncooperative stochastic game and presents a multiagent proximal policy optimization (PPO) technique for stationary systems. To increase flexibility and learning efficiency in nonstationary systems, it presents the multiagent meta-PPO method based on meta-learning. The primary approaches include using approximation for state representation, using local observation-based policy optimization, and incorporating meta-learning to improve convergence. The results of the study demonstrate considerable performance gains in terms of job execution latency, payment, and queuing delay. The intrinsic complexity of multiagent coordination in dynamic situations and delayed learning adaption in conventional approaches have been identified as problems. The study demonstrates how the proposed framework outperforms baselines in both stationary and nonstationary environments.

The survey paper “Machine and Deep Learning for Resource Allocation in Multi-Access Edge Computing (MEC): A Survey” [17] presents a detailed review of the use of machine learning (ML and DL approaches for resource allocation in MEC systems. The paper focuses on three major topics: task offloading, task scheduling, and joint resource allocation, emphasizing the benefits of ML/DL-enabled mechanisms over traditional heuristic and optimization approaches. Methodologies include using reinforcement learning for adaptive task offloading, convolutional neural networks for feature extraction, and federated learning for privacy-preserving model training. The research highlights major problems such as computational complexity, adaptation to dynamic situations, and efficient integration of ML/DL models in resource-constrained MEC devices. The survey results show advances in latency reduction, energy efficiency, and overall quality of service. However, there are still shortcomings in areas like scalability, real-time flexibility, and MEC system heterogeneity. Future objectives include improving model optimization, establishing hybrid techniques, and utilizing new paradigms such as quantum computing for better resource allocation.

The research paper “Latency-Aware Container Scheduling in Edge Cluster Upgrades: A Deep Reinforcement Learning Approach” presents a latency-aware container scheduling technique that optimizes task allocation during edge cluster upgrades in MEC. The technique handles issues such as resource restrictions, geographically scattered nodes, and the requirement for adaptive scheduling during dynamic upgrades. Using a policy gradient-based reinforcement learning (RL) algorithm, the technique includes task and node properties, including geographic distribution, via a self-attention-based feature extraction mechanism. The approach reduces task delay by around 30% as compared to baseline methods. It addresses holes/gaps in existing techniques by taking into account node upgrade status, location information, and long-term optimization. Evaluations using simulated and real-world data demonstrate higher performance in connectivity, computation, and download latency. However, adoption into systems such as Kubernetes poses integration issues. While the method requires extensive training data and computational cost, it is scalable, efficient, and capable of real-time execution, making it a reliable option for IoT services in edge contexts [18].

The paper “Improved Double Deep Q Network-Based Task Scheduling Algorithm in Edge Computing for Makespan Optimization” presents a task scheduling technique that uses an upgraded Double Deep Q Network (DDQN) to optimize makespan in EC settings. It tackles issues such as work scheduling in dynamic and diverse edge contexts, the constraints of static scheduling algorithms, and the inefficiency of classical Q-learning models owing to state explosion. The process entails splitting the computation of goal Q-values and action selection over two networks, creating a novel reward function, and improving experience replay using a control unit to maximize data consumption. An enhanced Particle Swarm Optimization (PSO) technique is utilized to pre-train the evaluation network, provide starting solutions, and shorten convergence time. The results of experiments show that the algorithm outperforms competing techniques, such as First-Come, First-Served (FCFS), Shortest Job First (SJF), Particle Swarm Optimization (PSO), Simulated Annealing-PSO (SA-PSO), DDQN, and classic DDQN, in terms of makespan and load balancing across a variety of workloads and machine configurations. While successful, obstacles include the need for significant training data and the computing needs of reinforcement learning. The study indicates that the suggested technique considerably enhances task scheduling performance in EC and recommends more research into workflow scheduling [19].

The research paper “Edge Computing Sleep Mode Task Scheduling Based on Deep Reinforcement Learning” offers a task scheduling technique for EC systems called Proximal Policy Optimization Task Scheduling based on Sleep Mode (PPO-TSSM), which addresses the difficulties of idle energy consumption and resource limits. The suggested method maximizes job completion time, energy usage, and meeting deadlines. Using a sleep-mode-based edge architecture, the study models task scheduling as an MDP and applies the

PPO algorithm, which incorporates both policy and value functions. The system constantly switches servers between active and sleep modes, activating them as tasks come. Experimental results show that PPO-TSSM decreases energy consumption by up to 74.76% when compared to systems without sleep mode, while increasing minimum job completion time by just 5.33%. In addition, it reduces total work costs by 39.45% as compared to other baseline techniques. While sleep mode efficiently reduces idle energy usage, extra system wake-up times and container reconfigurations cause modest delays. Future work will enhance the system by including dynamic network changes and explicit resource needs. This solution demonstrates sleep mode's potential for lowering operational costs and enhancing scheduling efficiency in EC [20].

The research paper "Dependency-Aware Application Assigning and Scheduling in Edge Computing" [21] explores dependency-aware program assignment and scheduling in EC settings. It identifies issues such as task interdependence, limited and varied edge resources, and uneven spatial–temporal distribution of requests. The study models applications as directed acyclic graphs (DAGs) and presents Daas, a unique approach for simultaneously optimizing task assignment and scheduling in an online setting, solving both NP-hard challenges. Daas uses a heuristic technique to assess job priority, allocating and scheduling activities iteratively in order to optimize resource use and fulfill application deadlines. Key findings show that Daas allows up to 20% more applications to achieve deadlines than baseline approaches, with a maximum improvement of 38% in large-scale cases. Methodological limitations include a failure to account for storage limits, real-time unpredictability in computation times, and simplifications of edge server and base station connections. Drawbacks include the solution’s heuristic character, which does not ensure global optimality, as well as computing cost from iterative optimization. Despite its shortcomings, Daas offers a useful and efficient framework for managing dependent applications in dynamic EC settings.

The paper “Deep Reinforcement Learning for Task Scheduling in Intelligent Building Edge Network” addresses the shortcomings of standard cloud-based job scheduling, which suffers with high latency and inefficient resource use in intelligent building edge networks. To overcome these issues, the authors suggest a DDQN-based task scheduling algorithm with an empirical replay mechanism. This strategy increases scheduling success rates, speeds convergence, and decreases edge server overhead. The technique employs a complete framework that includes a user workload model that processes tasks such as as DAGs to account for dependencies, an environment model that replicates real-time resource conditions, and a DRL-based decision-making agent for dynamic optimization. Experimental findings show that the algorithm outperforms previous techniques, with greater scheduling success rates, lower latency, and better resource usage even as task load increases. However, the method takes significant training, incurs computing expense in large-scale applications, and relies on precise job parameter prediction for optimal deployment. Overall, this research greatly increases task scheduling for intelligent building systems by striking a compromise between performance, efficiency, and energy savings in EC environments [22].

The paper “Deep Reinforcement Learning Based Task Scheduling in Edge Computing Networks” focuses on reducing task scheduling delays in cloud-edge networks, specifically the nondeterministic polynomial time (NP)-hard Task Scheduling for Delay Optimization issue. The authors offer the cloud–edge collaboration scheduling algorithm, which is based on the Asynchronous Advantage Actor-Critic (CECS-A3C) DRL architecture. This approach dynamically distributes resources and optimizes task offloading decisions, taking into account edge and cloud resource limits. Unlike existing systems that frequently rely on single-edge nodes or utilize wasteful experience replay mechanisms, CECS-A3C uses asynchronous training with parallel agents, which eliminates the requirement for replay and improves scalability. Tasks are planned using real-time status changes to maximize cumulative rewards depending on task delay sensitivity. Experimental results reveal that CECS-A3C decreases task delays by 28.3% and 46.1% when compared to DQN and RL-G algorithms, respectively, while also attaining faster convergence and improved scalability under various task loads. However, its dependence on centralized decision-making may limit scalability in highly distributed systems, and real-world implementation issues include unexpected network dynamics and the ability to collect reliable state information. Overall, the proposed framework improves scheduling efficiency, resulting in timely task execution and optimal resource usage in EC networks [23].

The paper “Deep Reinforcement Learning-Based Approach for Online Service Placement and Computation Resource Allocation in Edge Computing” provides an innovative strategy for optimizing service placement and resource allocation in 5G-enabled EC systems, with the goal of minimizing task delay. It handles issues such as dynamic and unexpected task arrivals, limited edge resources, service migration overhead, and the interdependence of placement and resource decisions. The problem is expressed as an MDP with hybrid discrete-continuous action spaces, in which latency components (switching, serving, and offloading) impact the reward function. The proposed PDQN combines two neural networks—one for optimum resource allocation and the other for assessing hybrid action Q-values—and employs techniques such as double-network topologies and experience replay to improve stability. Extensive simulations show that the methodology outperforms baseline approaches in lowering latency across a variety of system characteristics, including CPU capacity, service diversity, and storage limits. While the study findings seem encouraging, there are several limitations, such as scaling issues, computing cost from reinforcement learning, and simplified assumptions that may restrict real-world application. Nonetheless, it demonstrates the utility of DRL in solving challenging optimization issues in EC systems [24].

The paper “Deep Reinforcement Learning Algorithms for Low Latency Edge Computing Systems” investigates the use of DRL models—Q-Learning, Double Q-Learning, DQN, and DDQN—to improve resource allocation and task scheduling in EC settings, with an emphasis on latency reduction. It solves issues such as task scheduling complexity caused by diverse resource needs, high latency from extensive task queues, and the requirement for intelligent decision-making in dynamic, low-latency environments. Using MDP to represent the problem, the study demonstrates the improved performance of DDQN, which separates action selection and assessment to increase stability and decrease Q-value overestimations. Neural networks and approaches such as experience replay are being used to improve training and decision-making. While Q-Learning and DQN experience instability and overestimation, DDQN delivers consistent, efficient resource allocation with a lower average latency (0.97–1.03 seconds) throughout simulations. The results show that DDQN can give greater rewards and more consistent outcomes, making it a viable option for intelligent EC systems. However, the paper identifies limits such as scalability issues and the computational complexity of DRL models, recommending the future inclusion of advanced algorithms such as dueling DQN for more improvement. Finally, the article finds that DRL models, particularly DDQN, greatly improve the performance of edge system [25].

The paper “Decentralized Scheduling for Concurrent Tasks in Mobile Edge Computing via Deep Reinforcement Learning” provides a decentralized task scheduling technique for MEC that reduces latency and lost job ratios in dynamic, resource-constrained contexts. The paper tackles issues such as communication costs in centralized techniques, problems with concurrent job execution, and heterogeneity edge networks. The proposed technique treats task offloading as a delay-aware optimization problem that is solved using DRL. Techniques such as Double DQN, Dueling DQN, Prioritized Replay Memory, and Recurrent Neural Networks (RNNs) are used to improve flexibility and convergence. The Distributed Optimization and Scheduling Algorithm (DOSA) algorithm divides the DRL network into user devices and servers, enabling each device to make autonomous offloading decisions with little computing complexity. Simulation findings show that DOSA greatly decreases task delay and drops task ratios when compared to baseline methods (such as greedy, random, and centralized DRL approaches). However, disadvantages include large training costs, difficulty in achieving convergence in decentralized environments, and a lack of consideration for real-world complications such as task dependency graphs and varied server caches. The identified needs include expanding the model to handle divisible tasks and using edge service caches to improve scheduling performance. Despite these constraints, the findings demonstrate DOSA's efficiency in balancing scalability and performance through rapid learning and adaptability to changing MEC settings[26].

The paper "A Resource-efficient Task Scheduling System using Reinforcement Learning" describes a novel RL-based task scheduling system that optimizes resource efficiency in multi-core CPU scenarios. The suggested methodology overcomes the limits of classic heuristic-based and hardcoded scheduling algorithms by adapting to dynamic computing environments, assuring equivalent or enhanced performance while drastically decreasing resource utilization. The RL model approaches the scheduling problem as an MDP, with states representing system configurations, actions allocating jobs to workers, and rewards punishing workload imbalance and excessive data transmission costs. The scheduler uses a deep Q-learning algorithm to learn rules that effectively generalize to previously encountered task graphs, displaying flexibility across a wide range of configurations. The experimental findings reveal that the RL-based scheduler reduces active workers to 7–8 (from 40 in baseline approaches) while maintaining comparable runtimes, demonstrating considerable advantages in efficiency and scalability. However, the technique has disadvantages, including high initial training costs and gaps, such as a lack of support for distributed systems and GPU-specific scheduling. Future work attempts to solve these limitations, making the architecture more applicable to dispersed and heterogeneous computing settings [27].

The paper titled “A Request Scheduling Optimization Mechanism Based on Deep Q-Learning in Edge Computing Environments” proposes a DRL-based (DQN) technique for optimizing the scheduling of user requests represented as DAGs in edge computing settings. The technique seeks to reduce long-term average system latency and increase job completion rates by treating the scheduling problem as an MDP. The system describes states as resource utilization and task queue statuses, actions as scheduling decisions, and incentives as penalties for delays and missed deadlines. The approach uses a deep Q-network, experience replay, and an ε-greedy policy to learn optimum scheduling strategies over time. Experimental results show that the DQN mechanism greatly lowers delay and enhances task completion rates when compared to a random-based technique, especially at high request arrival rates. Despite initial training instability and difficulty in high-dimensional state spaces, the mechanism achieves near-perfect job completion. However, shortcomings such as a lack of distributed edge intelligence and computing cost in real-time settings persist. Future studies will look at distributed scheduling solutions to improve decision-making in large-scale edge networks [28].

The paper “Deep Reinforcement Learning Based Delay-Aware Task Offloading for UAV-Assisted Edge Computing” describes a unique technique for maximizing job offloading in multi-access edge computing (MEC) environments that employs UAV relays. The proposed DDPG-based Task Offload Policy (DTOP) for mobile terminals reduces energy usage while meeting delay limitations by utilizing DRL to make effective offloading decisions. The system represents tasks as an MDP with continuous state and action spaces, and it includes a reward function that penalizes energy waste and task timeouts. A task equalization method guarantees that Unmanned Aerial Vehicle (UAVs) use resources in a balanced manner. Simulation findings show that Dynamic Task Offloading Protocol (DTOP) dramatically decreases energy usage and task delays when compared to baseline algorithms, while providing fair resource allocation using Jain's fairness index. Despite the initial training complexity and scalability issues, the system stabilizes quickly. The paper concludes with potential future paths, such as real-time implementation, scalability improvements, and security considerations [29].

The paper "Task Allocation in Industrial Edge Networks with Particle Swarm Optimization and Deep Reinforcement Learning" explores job allocation in industrial edge networks, focusing on energy usage reduction. The authors present an enhanced integer linear programming (ILP) model for efficient multi-workflow task allocation, which improves on previous heuristics. They provide Particle Swarm Optimization (PSO) and DRL as alternatives that are tested for scalability and efficiency. The primary issues include tackling the NP-hard nature of the task allocation problem, preserving QoS, and dealing with the computational inefficiencies of ILP in big networks. While PSO and DRL have shorter execution times than ILP, they sacrifice accuracy for scalability, with PSO excelling in smaller issue sizes and DRL doing better in bigger circumstances. Methodologically, ILP optimizes energy usage for workloads and network connections within resource and capacity limits. PSO approximates the answer with a meta-heuristic technique, whereas DRL uses proximal policy optimization (PPO) for iterative learning based on task-node mapping. The results demonstrate that PSO achieves a median optimality gap of 7.7% but struggles with bigger issue sizes, whereas DRL provides a lower upper bound for larger problems and executes quicker, completing in less than 1 second. Gaps include PSO's inability to handle large-scale jobs efficiently and the lack of attention for heterogeneity in network settings. Future research should focus on hybrid models that combine PSO accuracy with DRL scalability [30].

The paper "A Deep Reinforcement Learning-Based Hybrid Algorithm for Efficient Resource Scheduling in Edge Computing Environment" offers a hybrid approach that combines DQN and genetic algorithms (GA) to solve resource scheduling problems in EC settings. It aims to reduce application execution time by using DQN to produce the initial population for GA, improving convergence speed and optimization efficiency. Subtask dependence, the unpredictability of GA's beginning population, and the inefficiency of classical optimization methods in large-scale, distributed settings are all potential issues. Methodologically, the technique represents the scheduling issue as an MDP, using DQN for learning optimum task assignments and GA for further refining. Experiments on real-world workflows (Montage_25, Sipht_60, and Epigenomics_46) show that DQN_GA outperforms known techniques such as Heterogeneous Earliest Finish Time (HEFT), Predict Earliest Finish Time (PEFT), and Genetic Algorithm (GA) with random initialization, improving makespan by 5.16% over PEFT and 4.91% over GA. The findings show faster convergence and greater optimization quality, however there are certain shortcomings, such as restricted multi-objective and dynamic case handling. Future research might investigate multi-objective extensions and adaptability to dynamic EC settings [31].

The paper "Multi-Resource Interleaving for Task Scheduling in Cloud-Edge System by Deep Reinforcement Learning" introduces DeepMIC, a new DRL-based multi-resource interleaving strategy for improving task scheduling in cloud–edge computing systems. It solves the problem of optimizing multi-resource queuing inside a single edge node to reduce weighted-sum latency. DeepMIC employs a PPO DRL algorithm to dynamically interleave jobs across diverse resources, increasing resource utilization and lowering latency. The dynamic nature of cloud–edge systems creates challenges such as managing task interdependence, resource rivalry, and high computational complexity. One significant disadvantage is that the approach's efficacy in multi-objective situations or severe network states requires additional investigation. Methodologically, the problem is defined as a hypergraph matching problem, which allows for efficient task grouping for interleaved execution. Experiments reveal DeepMIC's superiority over previous approaches, with up to 1.67 times higher resource usage and a 30.9% reduction in average reaction time, demonstrating its potential to improve the performance of cloud–edge systems. Future research should focus on dynamic network circumstances and multi-objective optimization [32].

2.2 Conclusion

To address work scheduling in EC, we would suggest combining representation algorithms with reinforcement learning methods in this chapter. The task scheduling issue is first developed, along with the definition of EC's entities and relationships. The relation computation model is then suggested when the entities' EC embeddings have been defined. Later, a learning method based on DQL has been presented to train the embedding vectors and make the decisions about task scheduling. Hence, we can combine reinforcement learning and representation learning for task scheduling in EC is this chapter.

References

1. Ananthanarayanan, G., et al., 2017. Real-time video analytics: The killer app for edge computing. *Computer*, 50(10), pp. 58–67.⏎
2. Liu, L., Li, H. and Gruteser, M., 2019. Edge assisted real-time object detection for mobile augmented reality. In *Proceedings of 25th Annual International Conference on Mobile Computing and Networking* (pp. 1–16).⏎
3. Mao, H., Alizadeh, M., Menache, I. and Kandula, S., 2016. Resource management with deep reinforcement learning. In *Proceedings of 15th ACM Workshop on Hot Topics in Networks* (pp. 50–56).⏎
4. Tan, L. T. and Hu, R. Q., 2018, November. Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning. *IEEE Transactions on Vehicular Technology,* 67(11), pp. 10,190–10,203.⏎
5. Hasselt, H. V., 2010. Double Q-learning. In *Proc. Int. Conf. Neural Inf. Process. Syst.* (pp. 2613–2621).⏎
6. Tang, Z., Jia, W., Zhou, X., Yang, W. and You, Y., 2020. Representation and reinforcement learning for task scheduling in edge computing. *IEEE Transactions on Big Data*, 8(3), pp. 795–808.⏎
7. Mustapha, S. D. S. and Gupta, P., 2024. DBSCAN inspired task scheduling algorithm for cloud infrastructure.

Internet of Things and Cyber-Physical Systems, 4, pp. 32–39.⏎

8. Gupta, P., Rawat, P. S., Kumar Saini, D., Vidyarthi, A. and Alharbi, M., 2023. Neural network inspired differential evolution based task scheduling for cloud infrastructure. *Alexandria Engineering Journal,* 73, pp. 217–230.
9. Madhusudhan, H. S., Gupta, P., Saini, D. K. and Tan, Z., 2023. Dynamic virtual machine allocation in cloud computing using elephant herd optimization scheme. *Journal of Circuits, Systems and Computers*, 32(11), Art. 2350188.
10. Rawat, P. S., Gaur, S., Barthwal, V., Gupta, P., Ghosh, D., Gupta, D. and Rodrigues, J. J. C., 2025. Efficient virtual machine placement in cloud computing environment using BSO-ANN based hybrid technique. *Alexandria Engineering Journal,* 110, pp. 145–152.
11. HS, M. and Gupta, P., 2024. Federated learning inspired Antlion based orchestration for Edge computing environment. *PLoS One*, 19(6), Art. e0304067.
12. Gupta, P., Anand, A., Agarwal, P. and McArdle, G., 2024. Neural network inspired efficient scalable task scheduling for cloud infrastructure. *Internet of Things and Cyber-Physical Systems,* 4, pp. 268–279.⏎
13. Zhu, K., Zhang, Z., Sun, F. and Shen, B., 2022. Workflow makespan minimization for partially connected edge network: A deep reinforcement learning-based approach. *IEEE Open Journal of the Communications Society,* 3, pp. 518–529.⏎
14. Zheng, T., Wan, J., Zhang, J. and Jiang, C., 2022. Deep reinforcement learning-based workload scheduling for edge computing. *Journal of Cloud Computing*, 11(1), p. 3.⏎

15. Luo, Q., Hu, S., Li, C., Li, G. and Shi, W., 2021. Resource scheduling in edge computing: A survey. *IEEE Communications Surveys & Tutorials*, 23(4), pp. 2131–2165.⏎
16. Niu, L., Chen, X., Zhang, N., Zhu, Y., Yin, R., Wu, C. and Cao, Y., 2023. Multiagent meta-reinforcement learning for optimized task scheduling in heterogeneous edge computing systems. *IEEE Internet of Things Journal*, 10(12), pp. 10519–10531.⏎
17. Djigal, H., Xu, J., Liu, L. and Zhang, Y., 2022. Machine and deep learning for resource allocation in multi-access edge computing: A survey. *IEEE Communications Surveys & Tutorials,* 24(4), pp. 2449–2494.⏎
18. Cui, H., Tang, Z., Lou, J., Jia, W. and Zhao, W., 2024. Latency-aware container scheduling in edge cluster upgrades: A deep reinforcement learning approach. *IEEE Transactions on Services Computing,* 17(5), pp. 2530–2543.⏎
19. Zeng, L., Liu, Q., Shen, S. and Liu, X., 2023. Improved double deep Q network-based task scheduling algorithm in edge computing for Makespan optimization. *Tsinghua Science and Technology*, 29(3), pp. 806–817.⏎
20. Tao, S., Zeng, F., Tao, Y., Xia, P. and Liu, T., 2023, August. Edge computing sleep mode task scheduling based on deep reinforcement learning. In *2023 9th International Conference on Big Data Computing and Communications (BigCom)* (pp. 231–239). IEEE.⏎
21. Liao, H., Li, X., Guo, D., Kang, W. and Li, J., 2021. Dependency-aware application assigning and scheduling in edge computing. *IEEE Internet of Things Journal*, 9(6), pp. 4451–4463.⏎

22. Chen, Y., Wang, Y., Zhang, Z., Fu, Q., Wang, H. and Lu, Y., 2022, November. Deep reinforcement learning for task scheduling in intelligent building edge network. In *2022 Tenth International Conference on Advanced Cloud and Big Data (CBD)* (pp. 312–317). IEEE.⏎
23. Qi, F. A. N., Zhuo, L. and Xin, C., 2020, August. Deep reinforcement learning based task scheduling in edge computing networks. In *2020 IEEE/CIC International Conference on Communications in China (ICCC)* (pp. 835–840). IEEE.⏎
24. Liu, T., Ni, S., Li, X., Zhu, Y., Kong, L. and Yang, Y., 2022. Deep reinforcement learning based approach for online service placement and computation resource allocation in edge computing. *IEEE Transactions on Mobile Computing*, 22(7), pp. 3870–3881.⏎
25. Kumaran, K. and Sasikala, E., 2023, March. Deep reinforcement learning algorithms for low latency edge computing systems. In *2023 3rd International conference on Artificial Intelligence and Signal Processing (AISP)* (pp. 1–5). IEEE.⏎
26. Fan, Y., Ge, J., Zhang, S., Wu, J. and Luo, B., 2023. Decentralized scheduling for concurrent tasks in mobile edge computing via deep reinforcement learning. *IEEE Transactions on Mobile Computing*, 23(4), pp. 2765–2779.⏎
27. Morchdi, C., Chiu, C. H., Zhou, Y. and Huang, T. W., 2024, January. A resource-efficient task scheduling system using reinforcement learning. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)* (pp. 89–95). IEEE.⏎
28. Zhang, Y., Li, R., Zhao, Y. and Li, R., 2021, May. A request scheduling optimization mechanism based on deep Q-

learning in edge computing environments. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (pp. 1–2). IEEE.⏎

29. Huang, M., Sun, K., Hou, Y., Ye, Z., Wan, Y. and He, H., 2022, December. Deep reinforcement learning based delay-aware task offloading for UAV-assisted edge computing. In (pp. 1–8).⏎
30. Buschmann, P., Shorim, M. H., Helm, M. Proceedings of the 2022 5th International Conference on Algorithms, Computing and Artificial Intelligence, Bröring, A. and Carle, G., 2022, November. Task allocation in industrial edge networks with particle swarm optimization and deep reinforcement learning. In Proceedings of the 12th International Conference on the Internet of Things (pp. 239–247).⏎
31. Xue, F., Hai, Q., Dong, T., Cui, Z. and Gong, Y., 2022. A deep reinforcement learning based hybrid algorithm for efficient resource scheduling in edge computing environment. *Information Sciences,* 608, pp. 362–374.⏎
32. Pei, X., Sun, P., Hu, Y., Li, D., Tian, L. and Li, Z., 2024. Multi-resource interleaving for task scheduling in cloud-edge system by deep reinforcement learning. *Future Generation Computer Systems*, 160(November), pp. 522–536.⏎

Chapter 3

Supervised Machine Learning for Load Balancing in Fog Environments

Priyanshi Mishra and Yash Jain

DOI: 10.1201/9781003610168-3

3.1 Introduction

Fog computing has emerged as a transformative paradigm that bridges the gap between centralized cloud services and edge devices, such as Internet of Things (IoT) sensors, mobile devices, and industrial systems. Unlike traditional cloud computing, which centralizes computation and storage, fog computing decentralizes these processes, bringing them closer to data sources. This architectural shift offers significant benefits, including reduced latency, enhanced data security, and improved system efficiency. However, it also introduces unique challenges, particularly in managing workload distribution across heterogeneous and dynamic fog nodes.

Effective load balancing is critical in fog environments to optimize resource utilization, minimize delays, and prevent system failures. Supervised machine learning (ML) provides powerful tools to address these challenges by leveraging historical data to predict and adapt to workload variations in real time. This chapter explores the integration of supervised ML techniques in load balancing for fog computing, discussing their methodologies, applications, and future directions. This exploration includes an expanded focus on real-world applications, advanced ML techniques, and the challenges that researchers and practitioners face today.

3.2 Fog Computing: An Overview

Fog computing extends cloud capabilities by positioning computational resources closer to the edge, reducing the distance data needs to travel. This approach addresses several key issues in modern distributed systems:

1. **Reduced Latency:** Fog computing processes data locally, significantly reducing response times. This capability is vital for latency-sensitive applications such as autonomous vehicles, healthcare monitoring, and real-time video streaming. For instance, autonomous cars rely on instant decision-making to ensure passenger safety—a requirement that centralized cloud systems cannot meet [1].

A case study involving autonomous vehicles highlights that processing sensor data within milliseconds can prevent accidents and improve traffic efficiency. Traditional cloud solutions fail to meet such stringent latency demands, showcasing the necessity of localized fog computing [2].

1. **Enhanced Security:** By processing sensitive data locally, fog computing minimizes risks associated with data transmission, such as interception or unauthorized access. This is particularly crucial for industries handling sensitive information, such as healthcare and finance [3].

Moreover, compliance with data sovereignty laws, which require sensitive data to remain within specific geographic boundaries, becomes more manageable with fog computing. Enhanced security fosters trust in applications like remote health monitoring and financial transactions [4].

1. **Scalability:** Fog computing's distributed nature supports the scalability required for the rapidly growing number of IoT devices. The architecture allows horizontal scaling, accommodating diverse workloads and increasing the system's overall capacity [5].

Scalability enables seamless management of diverse and complex tasks, such as those in smart grids and industrial IoT systems, where millions of sensors interact in real time [6].

Despite these advantages, fog computing introduces challenges such as heterogeneous resource management, dynamic workloads, and real-time processing demands. Load balancing plays a pivotal role in addressing these challenges to ensure consistent performance and reliability.

3.3 Supervised Machine Learning (SML) Basics

SML is a type of ML where the model is trained on labeled data, meaning that each input in the training dataset has a corresponding output or label. This type of learning is called “supervised” because the algorithm learns from the training data that includes both the input features and the correct output, essentially guiding the model to make predictions. [7–9]

Core Concepts of Supervised Learning

1. **Labeled Data**: In supervised learning, the dataset is made up of pairs of input data and corresponding output labels. For example, in a classification task, the input could be an image, and the output label could be the object in the image (such as “cat” or “dog”). In a regression task, the input might be historical data, and the output could be a continuous value, such as a stock price or temperature.
2. **Training**: During training, an ML model learns a mapping from input features to the correct output labels. The model tries to minimize the difference between its predictions and the actual output by adjusting its internal parameters. This process is done iteratively, using optimization techniques like gradient descent, where the model fine-tunes itself by reducing the error over multiple iterations.
3. **Testing**: After training, the model is evaluated using a testing dataset, which contains labeled data that it has not seen during training. This helps assess how well the model generalizes to new, unseen data. The performance of the model is measured using different

metrics, depending on whether the task is classification or regression.

Supervised Learning Algorithms

Supervised learning encompasses several types of algorithms, each suited for different tasks, such as classification or regression.

1. **Classification Algorithms**: These algorithms are used when the output label is a categorical value. The goal is to assign an input to one of several predefined categories. Examples of classification algorithms include:

    - **Decision Trees**: A tree-like structure where each node represents a decision based on a feature, and each branch represents a possible outcome. The model makes predictions by traversing the tree from root to leaf.
    - **Support Vector Machines (SVM)**: SVMs aim to find the hyperplane that best separates the data points of different classes. It tries to maximize the margin between the closest data points of each class, known as the support vectors.
    - **K-Nearest Neighbors (KNN)**: KNN is a simple algorithm where the model classifies an input by looking at the “K” closest labeled data points and assigning the most common label among them.
    - **Logistic Regression**: Despite the name, logistic regression is a classification algorithm used for binary classification tasks. It calculates the probability of the input belonging to a particular

class based on a linear combination of input features.

2. **Regression Algorithms**: These algorithms are used when the output is a continuous value. The model tries to predict a numeric value based on input features. Examples of regression algorithms include:

    - **Linear Regression**: One of the simplest algorithms, linear regression models the relationship between input features and output as a straight line. It predicts a continuous value by finding the best-fit line through the data points.
    - **Decision Trees for Regression**: Similar to decision trees for classification, but in regression, each leaf represents a predicted continuous value, not a class label.
    - **Support Vector Regression (SVR)**: Similar to SVM, but SVR is designed for regression tasks. It tries to find a hyperplane that best fits the data points within a certain margin of tolerance.
    - **Random Forests**: A collection of decision trees, each built on random subsets of the training data. In regression, the prediction is made by averaging the outputs of all the trees in the forest.

Evaluation Metrics in Supervised Learning

To assess the performance of a model, various evaluation metrics are used. These metrics help determine how well the model generalizes to unseen data.

1. **For Classification Tasks**

    - **Accuracy**: The fraction of correctly predicted labels out of the total number of predictions. It is commonly used but can be misleading if the classes are imbalanced.

- **Precision**: The fraction of true positive predictions (correctly predicted positive class) out of all positive predictions (true positives + false positives). It measures the accuracy of positive predictions.
- **Recall (Sensitivity)**: The fraction of true positives out of all actual positives (true positives + false negatives). It measures how well the model captures all positive cases.
- **F1 Score**: The harmonic mean of precision and recall. It is useful when you need a balance between precision and recall, especially in imbalanced datasets.
- **Confusion Matrix**: A table used to evaluate the performance of a classification model. It summarizes the correct and incorrect predictions for each class.

2. **For Regression Tasks**

- **Mean Absolute Error (MAE)**: The average of the absolute differences between predicted and actual values. It gives a sense of the average error in the predictions.
- **Mean Squared Error (MSE)**: Similar to MAE but squares the differences before averaging them. This gives more weight to larger errors, making MSE more sensitive to outliers.
- **Root Mean Squared Error (RMSE)**: The square root of the MSE, which brings the error measure back to the original scale of the data.

- **R-squared**: A measure of how well the model explains the variance in the data. It indicates the proportion of the variance in the dependent variable that is predictable from the independent variables.

Overfitting and Underfitting

Two common challenges in supervised learning are overfitting and underfitting.

- **Overfitting**: Occurs when the model learns too much from the training data, including noise and irrelevant patterns. As a result, it performs very well on the training data but poorly on unseen test data. Overfitting can be mitigated by using regularization techniques, cross-validation, and ensuring that the model is not overly complex for the available data.
- **Underfitting**: Happens when the model is too simple to capture the underlying patterns in the data. It leads to poor performance on both the training and test data. Underfitting can be addressed by increasing model complexity, using more relevant features, or reducing regularization.

Training, Validation, and Test Datasets

The typical workflow in supervised learning involves splitting the data into three sets:

- **Training Dataset**: Used to train the model, allowing it to learn from the input–output pairs.
- **Validation Dataset**: Used to tune hyperparameters and validate the model’s performance during the training phase. This helps in model selection.

- **Test Dataset**: Used to evaluate the final model’s performance after training is complete. It provides an unbiased estimate of the model’s generalization ability.

Overview of Load Balancing

Load balancing is a crucial concept in distributed systems and networks that involves distributing incoming network traffic or computational tasks across multiple resources, such as servers, network links, or processing units. The goal of load balancing is to optimize the utilization of resources, improve the response time, and ensure that no single resource is overwhelmed with too much work. In the context of computing, load balancing is often used to enhance the availability, reliability, and performance of systems, particularly in large-scale distributed environments where resources are spread across multiple nodes or data centers.

The concept of load balancing becomes particularly important in scenarios where high availability and fault tolerance are critical. These scenarios include cloud computing, web servers, content delivery networks (CDNs), and even fog computing systems, where distributed resources must work together to provide seamless service to end users. Load balancing allows these systems to scale efficiently by ensuring that workloads are shared evenly and that no single node or resource becomes a bottleneck.

Types of Load Balancing

1. **Network Load Balancing**: In network load balancing, the aim is to distribute incoming network traffic evenly across a group of servers or devices to ensure optimal network performance and prevent overload on a single device. This type of load balancing is most commonly used in web servers, application servers, and even in

telecommunications systems. By distributing requests based on various algorithms, network load balancers help reduce the possibility of a single server or network link becoming overwhelmed, which can lead to service disruptions or slow response times.

2. **Task Load Balancing**: Task load balancing, also known as computational or application-level load balancing, refers to the allocation of computational tasks to different resources or nodes in a system. This type of load balancing is typically applied in scenarios like parallel computing, where a large number of small tasks need to be distributed across a set of processors. The goal is to minimize idle time on the processors and achieve a more efficient computation by ensuring that each processor or computing node is assigned a task that can be processed in parallel with others. Task load balancing can be found in distributed computing environments like high-performance computing systems or cloud-based environments.
3. **Global Load Balancing**: Global load balancing refers to the process of distributing traffic or computational tasks across multiple data centers or geographical regions. It helps ensure that users access the most responsive server or application, depending on their location, load conditions, and availability. Global load balancing can leverage geolocation-based routing, which directs users to the closest data center or the one with the lowest load. It is typically used in large-scale web applications, CDNs, and enterprise systems that operate in multiple regions.
4. **Local Load Balancing**: Local load balancing is concerned with distributing traffic or tasks across resources within a single data center or server farm. It

aims to optimize resource utilization within the localized infrastructure and ensure that no single server becomes a bottleneck. Local load balancing can be handled by software or hardware appliances and is typically used in situations where the traffic or tasks are concentrated within a specific physical or logical location.

Common Load Balancing Algorithms

Several algorithms are commonly used for load balancing, each with its own advantages and trade-offs. These algorithms are designed to determine how incoming requests or tasks should be distributed to various resources. Some of the most widely used load balancing algorithms include:

1. **Round Robin**: Round robin is one of the simplest and most widely used load balancing algorithms. It works by assigning requests or tasks to the next server in a cyclic manner, regardless of the server's current load. The simplicity of this approach makes it easy to implement, but it may not always achieve optimal performance, especially in systems where servers have different processing capacities or where traffic patterns are not uniform.
2. **Least Connections**: The Least Connections algorithm directs new incoming requests to the server that has the fewest active connections or tasks. This approach is beneficial when server load is highly variable, as it aims to balance the number of concurrent connections across the available servers. This method can help prevent servers from becoming overloaded with too

many tasks, but it requires that the load balancer has real-time access to server connection counts.

3. **Weighted Round Robin**: Weighted round robin is an extension of the basic round robin algorithm that takes server capacities into account. In this algorithm, each server is assigned a weight based on its processing power, and the load balancer distributes requests according to these weights. For example, a server with a higher weight will receive more requests than a server with a lower weight. This approach ensures that more powerful servers handle a greater portion of the traffic, leading to a more balanced and efficient system.
4. **Least Response Time**: In the Least Response Time algorithm, new requests are sent to the server with the shortest response time. This method is particularly useful when servers are highly variable in performance, as it ensures that users are directed to the fastest and most responsive resources. However, this algorithm may not always be effective if there is high variability in server response times due to factors such as network congestion or external system dependencies.
5. **Hash-Based Load Balancing**: Hash-based load balancing involves distributing requests based on a hash function, typically applied to a specific parameter in the request, such as the IP address or session ID. This approach ensures that requests with the same hash value are sent to the same server, which can be helpful for session persistence or maintaining stateful interactions. While it can lead to uneven distribution if the data is skewed, it is effective for systems that require session affinity or where a particular server needs to handle specific requests consistently.

6. **Dynamic Load Balancing**: Dynamic load balancing involves adjusting the distribution of requests or tasks based on real-time feedback, such as server load, resource availability, or network conditions. This type of load balancing is adaptive and continuously responds to changing conditions within the system, making it more suitable for environments with fluctuating or unpredictable workloads. Dynamic load balancing typically requires more sophisticated algorithms and systems that can monitor and analyze system performance in real time.

Challenges of Load Balancing

While load balancing offers significant benefits, it also comes with its own set of challenges. These challenges arise from the complexity of managing distributed systems, maintaining high availability, and ensuring optimal resource utilization.

1. **Scalability**: Load balancing must be able to scale effectively as the number of resources or requests grows. Systems need to be designed to handle an increasing number of servers or tasks without introducing performance bottlenecks or delays in request handling.
2. **Fault Tolerance**: Load balancers must be able to handle failures gracefully. When a server or resource goes down, the load balancer should automatically reroute traffic to available servers, ensuring minimal disruption to the end user. Implementing fault tolerance requires monitoring health and performance metrics of all servers in real time.

3. **Traffic Distribution**: Distributing traffic efficiently is a critical challenge, especially in heterogeneous systems where resources may have different processing capabilities. Load balancing algorithms must take into account the varying capabilities of different servers to avoid overloading less powerful resources.
4. **Latency and Response Time**: The speed and responsiveness of load balancing decisions can significantly impact the overall system performance. Load balancers must make decisions quickly to ensure low latency and avoid delays in processing requests. High-latency decisions can lead to performance degradation and a poor user experience.
5. **Security**: Security is another challenge in load balancing, particularly in cloud-based or web applications. Load balancers must ensure that they are protected from threats such as Distributed Denial of Service attacks, which could overwhelm the load balancing system itself. Ensuring that traffic is routed to secure and trusted resources is critical for maintaining system integrity.

Load Balancing in Fog Computing

Load balancing refers to the even distribution of computational tasks across available resources. In fog environments, load balancing must account for:

- **Heterogeneous Resources:** Fog nodes vary in computational power, memory, and network bandwidth [7].
- **Dynamic Workloads:** IoT devices generate fluctuating workloads, requiring adaptive strategies [8].

Static Load Balancing

Static techniques assign tasks based on predefined rules, such as:

- **Round Robin:** Tasks are distributed cyclically among nodes [9].
- **Weighted Round Robin:** Nodes with higher capacities receive more tasks [10].

While simple and easy to implement, static methods lack flexibility to adapt to real-time changes, potentially leading to inefficiencies under variable workloads. For example, in a healthcare monitoring network, static allocation might overload nodes monitoring critical patients, jeopardizing timely data processing [11].

Dynamic Load Balancing

Dynamic methods consider real-time metrics like CPU usage, memory, and network latency. Examples include:

- **Least Connection Method:** Allocates tasks to nodes with the fewest active connections [12].
- **Least Response Time Method:** Prioritizes nodes with the shortest response times [13].

These methods require continuous monitoring and computational overhead but significantly improve adaptability and efficiency. For instance, real-time traffic management in smart cities can benefit from these methods by dynamically redirecting resources to congested areas [14].

Predictive Analytics and ML Integration

Dynamic methods can be enhanced by integrating predictive analytics through supervised ML. These models can forecast workload spikes and anticipate resource requirements, enabling proactive task redistribution [15].

Supervised Machine Learning for Load Balancing

Supervised ML involves training models on labeled datasets to make predictions. In fog environments, such models can predict workload patterns and resource availability, enabling proactive task allocation.

Implementation Steps

1. **Data Collection:** Gather metrics such as CPU usage, memory consumption, and network latency from fog nodes. This data forms the foundation for training ML models. Techniques like IoT data logging and real-time monitoring frameworks (e.g., MQTT) can facilitate comprehensive data collection [16].
2. **Feature Engineering:** Identify and preprocess key features, such as workload intensity and network conditions, ensuring that models focus on critical parameters. Advanced feature selection methods like principal component analysis (PCA) can reduce dimensionality while preserving essential data characteristics [17].
3. **Model Selection**

    - Regression models (e.g., linear regression) predict workloads [18].
    - Classification models (e.g., decision trees) categorize tasks based on resource requirements [19].

      For example, Random Forest models have been shown to outperform simpler algorithms in workload prediction scenarios due to their ability to handle complex interactions [20–24].

4. **Model Training and Evaluation:** Train models on historical data, evaluating them using metrics like accuracy, precision, and recall. Cross-validation ensures robust performance. Hyperparameter tuning using techniques like grid search can optimize model accuracy [21, 31–34].
5. **Deployment and Continuous Learning:** Deploy trained models in real-time environments, continuously updating them to adapt to evolving workloads. Lightweight frameworks like TensorFlow Lite enable efficient deployment on resource-constrained fog nodes [22].

Advanced Techniques

- **Ensemble Learning:** Combines multiple models for improved accuracy and robustness (e.g., Random Forests) [23].
- **Deep Learning:** LSTMs and CNNs capture temporal and spatial patterns, enhancing prediction capabilities in complex environments [24].
- **Reinforcement Learning:** Complements supervised learning by optimizing load balancing policies based on feedback mechanisms [25].

Supervised Learning Models for Load Balancing

Supervised learning models are a subset of ML techniques that leverage labeled datasets to train algorithms, enabling them to make predictions or decisions based on input features. In the context of load balancing, supervised learning can be used to predict optimal resource allocation, task distribution, or traffic management strategies based on historical data. The ability to learn from past patterns and adapt to new, unseen data is particularly beneficial for load balancing in complex, dynamic environments such as cloud computing, fog computing, and web applications, where network traffic and workloads can fluctuate in real time.

How Supervised Learning Enhances Load Balancing

Traditional load balancing techniques, such as round robin, least connections, and random distribution, use fixed rules or simple heuristics to allocate resources. While these methods are effective in some contexts, they often fail to account for complex patterns and changing conditions, such as varying server performance, fluctuating network traffic, and the dynamic availability of resources. Supervised learning models, on the other hand, can be trained on historical data to recognize patterns and make data-driven decisions, providing several advantages:

1. **Adaptability**: Supervised learning models can adapt to changing traffic conditions and server loads. By learning from past data, these models can identify trends and predict future resource demands, allowing them to dynamically adjust load balancing decisions in real time.
2. **Precision**: Unlike static methods, supervised learning models can identify subtle patterns that may not be obvious to rule-based systems. For example, models can recognize correlations between server performance and incoming traffic patterns, leading to more efficient resource allocation.
3. **Predictive Capabilities**: Supervised learning models can predict when specific resources (e.g., servers and network links) will become overloaded, enabling proactive load balancing decisions that prevent

performance degradation or system failures before they occur.

4. **Optimization**: By continuously learning from new data, supervised learning models can optimize load balancing over time, making better decisions as more training data becomes available, leading to improved overall system performance.

Key Steps for Implementing Supervised Learning in Load Balancing

To apply supervised learning to load balancing, the following steps are typically involved:

1. **Data Collection and Preprocessing**: The first step in applying supervised learning to load balancing is to collect relevant data from the system. This data typically includes metrics such as:

    - Server load (e.g., CPU, memory, and disk utilization)
    - Network traffic volume
    - Response time and latency
    - Server health and availability
    - Historical load balancing decisions (e.g., resource allocation decisions and task assignments)

      The data is then preprocessed to ensure it is clean, structured, and ready for training. This may involve removing duplicates, handling missing values, and normalizing the data to ensure consistency and accuracy.

2. **Feature Selection**: Selecting the right features is crucial for the success of supervised learning models.

In the context of load balancing, important features might include:

- Current server load (e.g., CPU usage and memory consumption)
- Incoming request rate (e.g., number of incoming network packets per second)
- Previous server performance data (e.g., average response time and task completion time)
- Resource utilization patterns (e.g., memory usage over time)
- Network conditions (e.g., bandwidth usage and packet loss rate)

  Choosing the appropriate features helps the model learn patterns that are most relevant to optimizing load balancing decisions.

3. **Model Training**: Once the data is prepared and features are selected, the next step is to train a supervised learning model. The model learns from the labeled data, where the input features (e.g., server load and network traffic) are mapped to the corresponding output labels (e.g., which server should handle the next request). Some of the commonly used supervised learning algorithms for load balancing include:

   - **Linear Regression**: This algorithm can be used for predicting continuous values, such as the expected load on a server based on historical data. It models the relationship between input features and server load to provide predictions for optimal resource allocation.

- **Decision Trees**: Decision trees split data into subsets based on feature values and can be used to make load balancing decisions based on factors like server load and network conditions. They are easy to interpret and can handle both numerical and categorical data.
- **Random Forests**: Random forests are an ensemble method that builds multiple decision trees and combines their predictions. They are more robust than individual decision trees and can handle complex relationships between features.
- **Support Vector Machines**: SVM can be used for classification tasks in load balancing, such as determining which server should handle an incoming request. It finds the optimal boundary (hyperplane) that separates data points from different classes, ensuring the best allocation decisions.
- **K-Nearest Neighbors**: KNN can be used to classify new requests based on their similarity to historical data. For load balancing, it can predict which server is likely to be the best match for handling a new request, based on past requests with similar features.

4. **Model Evaluation and Testing**: After training the model, it is important to evaluate its performance using a separate testing dataset. The testing data should not overlap with the training data to ensure that the model generalizes well to new, unseen instances. Various metrics are used to evaluate the performance of supervised learning models, depending on the type of task:

    - For **regression** tasks (e.g., predicting server load), metrics such as MSE or R-squared are commonly used.
    - For **classification** tasks (e.g., choosing which server to handle a request), metrics like accuracy, precision, recall, and F1 score are often used.

    By evaluating the model's performance, we can identify areas for improvement and fine-tune the model, such as adjusting hyperparameters or incorporating additional features.

5. **Deployment and Real-Time Predictions**: Once the model is trained and evaluated, it is deployed in the load balancing system for real-time predictions. The model continuously receives data from the system (e.g., server load and traffic patterns) and makes predictions about which resources should handle incoming tasks. These predictions are used to inform load balancing decisions, ensuring that the system remains efficient and responsive under varying conditions.

The model can be updated periodically with new data to improve its performance and adapt to changes in the system over time. This continuous learning approach ensures that the load balancing mechanism evolves as the system scales or undergoes changes in usage patterns.

Benefits of Using Supervised Learning for Load Balancing

1. **Dynamic and Adaptive Load Balancing**: Supervised learning models can adjust to dynamic changes in traffic patterns, server performance, and network conditions. By learning from historical data, these models can predict future load scenarios and adjust

resource allocation accordingly, ensuring that resources are efficiently utilized.

2. **Improved Resource Utilization**: By using supervised learning to make data-driven decisions, load balancing can be more efficient, reducing the chances of overloading specific resources. Models can predict which servers are underutilized and shift load accordingly, maximizing the overall efficiency of the system.
3. **Proactive Fault Management**: Supervised learning models can help predict potential server failures or performance degradation based on historical data. This proactive approach enables the system to reroute traffic before a failure occurs, reducing downtime and improving system reliability.
4. **Scalability**: Supervised learning models can scale to handle increasing amounts of data, making them suitable for large-scale distributed systems. As the system grows, the model can learn from new data, adapting to the increasing complexity and providing accurate load balancing decisions.
5. **Optimized Response Times**: By accurately predicting the load distribution and traffic patterns, supervised learning models can reduce response times for users. By ensuring that the most capable resources are handling incoming requests, the system can provide quicker responses, improving the overall user experience.

3.4 Challenges and Considerations

While supervised learning provides many benefits for load balancing, there are several challenges to consider:

1. **Data Quality and Availability**: The performance of supervised learning models depends on the quality and quantity of historical data. If the data is noisy, incomplete, or not representative of real-world conditions, the model's predictions may be inaccurate, leading to poor load balancing decisions.
2. **Model Complexity**: Supervised learning models, especially complex ones like deep learning models or ensemble methods, may require significant computational resources for training and deployment. This can introduce overhead, especially in real-time systems where low latency is critical.
3. **Overfitting**: Overfitting occurs when the model learns to perform well on the training data but fails to generalize to new, unseen data. Regularization techniques and cross-validation are necessary to prevent overfitting and ensure the model's robustness.
4. **Dynamic Environment**: Load balancing in dynamic environments, such as cloud or fog computing, presents a challenge for supervised learning models, as the system's state can change rapidly. To address this, models need to be retrained periodically or use techniques like online learning to adapt to new data without retraining from scratch.

3.5 Mathematical Formulations for Load Balancing

To better understand the principles of load balancing, consider:

1. **Load Estimation Formula:**
   Li=α×CPUi+β×Memi+γ×NetiL_i = \α \times CPU_i + \β \times Mem_i + \γ \times Net_i
   where:
   - LiL_i: Load on node ii
   - CPUiCPU_i: CPU utilization of node ii
   - MemiMem_i: Memory usage of node ii
   - NetiNet_i: Network bandwidth usage of node ii
   - α,β,γ Weight coefficients.
2. **Optimization Problem:**
   min∑i=1N(LiCi)\min \sum_{i=1}^{N} \left(\frac{L_i}{C_i}\right)
   Subject to:
   - ∑i=1NLi≤Total Load Capacity \sum_{i=1}^{N} L_i \leq \text{Total Load Capacity}
   - Li≥0L_i \geq 0

These formulas help for designing algorithms that could optimize resource usage while preventing overloading.

3.6 Applications and Case Studies

1. **Smart Cities:** Predictive ML models optimize traffic flow and resource allocation, improving urban infrastructure efficiency. For instance, congestion prediction algorithms dynamically adjust traffic light timings to reduce delays [26].
2. **Healthcare:** Models forecast workload spikes in wearable device networks, ensuring timely processing of patient data. In critical care scenarios, such models

prioritize resource allocation to nodes handling emergency data streams [27].

3. **Industrial IoT:** Supervisory control systems dynamically allocate tasks across machines, reducing downtime and boosting productivity. Predictive maintenance models use sensor data to preemptively address potential failures, minimizing production disruptions [28].
4. **Agriculture:** IoT-enabled smart farms use fog nodes to monitor soil conditions and weather patterns. ML models predict irrigation needs, ensuring efficient water use and optimal crop growth [29].
5. **Disaster Management:** In disaster-prone areas, fog networks facilitate real-time monitoring and communication. Predictive ML models detect early warning signs, enabling proactive measures to mitigate risks [30].

3.7 Challenges and Future Directions

3.7.1 Challenges

- **Data Quality and Volume:** High-quality, labeled datasets are essential for training effective models. Addressing noisy or incomplete data remains a significant challenge [31].
- **Real-time Adaptation:** Ensuring low-latency predictions in dynamic environments [32].
- **Resource Constraints:** Balancing ML computational overhead with limited fog node resources [33].
- **Ethical Concerns:** Privacy and data security issues in ML-driven systems require robust governance

frameworks [34].

3.7.2 Future Research Directions

1. **Federated Reinforcement Learning:** Integrating federated learning with reinforcement strategies can enable decentralized, privacy-preserving decision-making in fog networks [35].
2. **Blockchain Integration:** Using blockchain for secure and transparent data sharing among fog nodes can enhance trust and system reliability [36].
3. **Multi-objective Optimization:** Incorporate trade-offs between latency, energy efficiency, and QoS in load balancing algorithms for holistic improvements [37].
4. **AI-Driven Resilience Models:** Develop AI systems that predict and adapt to node failures, ensuring uninterrupted service delivery [38].
5. **Green Computing:** Prioritize energy-efficient algorithms to reduce the environmental footprint of large-scale fog networks [39].

3.8 Conclusion

Supervised ML offers robust solutions to the complex load balancing challenges in fog environments. By leveraging predictive analytics and adaptive strategies, these models enhance resource utilization, system reliability, and user experience. As fog computing evolves, integrating advanced ML techniques will play a pivotal role in addressing future challenges and unlocking the full potential of distributed computing systems.

References

1. Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). Fog Computing and its role in the Internet of Things. *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*.⏎
2. Madsen, H., Albeanu, G., Popentiu-Vladicescu, F., & Zhang, Z. (2013). Reliability in the Utility Computing era: Towards reliable Fog Computing. *International Conference on Systems, Signals, and Image Processing*.⏎
3. Breiman, L. (2001). *Random Forests*. Machine Learning.⏎
4. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.⏎
5. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12, 2825–2830.*⏎
6. Abawajy, J., & Hassan, M. M. (2019). *Machine Learning for Fog Computing: Principles and Applications*. Springer. Vol. 1, pp. 1–23.⏎
7. Singh, S., & Chana, I. (2016). QoS-aware autonomic resource management in Cloud Computing: A systematic review. *ACM Computing Surveys*, (3), 1–46.⏎
8. Li, W., & Liu, J. (2015). Reinforcement learning for dynamic resource optimization in Fog Computing. *Journal of Parallel and Distributed Computing*, 25, 1–23.⏎
9. Alsadie, D. (2024). Artificial intelligence techniques for securing Fog Computing environments: Trends, challenges, and future directions. *IEEE Access*, 12, 151598–151648.⏎
10. Mastorakis, G., Mavromoustakis, C. X., Batalla, J. M., & Pallis, E. (Eds.). (2020). *Convergence of artificial*

intelligence and the Internet of Things. Cham, Switzerland: Springer International Publishing, pp. 1–9.⏎

11. Islam, M. S. U., Kumar, A., & Hu, Y. C. (2021). Context-aware scheduling in Fog computing: A survey, taxonomy, challenges and future directions. *Journal of Network and Computer Applications,* 180, 103008.⏎
12. Lin, Z., Lu, L., Shuai, J., Zhao, H., & Shahidinejad, A. (2023). An efficient and autonomous planning scheme for deploying IoT services in fog computing: A metaheuristic-based approach. *IEEE Transactions on Computational Social Systems,* 11(1), 1415–1429.⏎
13. Heilig, L., Lalla-Ruiz, E., Voß, S., & Buyya, R. (2018). Metaheuristics in cloud computing. *Software: Practice and Experience*, 48, 1729–1733.⏎

14 Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R. H., Morrow, M. J., & Polakos, P. A. (2017). A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 20, 416–464.⏎

15. Gupta, H., Vahid, D., & Goudar, R. H. (2019). Resource allocation challenges in Fog Computing: A review. *Journal of Cloud Computing*, 12, 1–23.⏎
16. Chollet, F. (2018). *Deep Learning with Python*. Manning Publications, Vol. 1, pp. 1–55.⏎
17. Real, Esteban et al. (2020). AutoML-Zero: Evolving machine learning algorithms from scratch. *International Conference on Machine Learning*, 1, 1–9.⏎
18. Aggarwal, C. C. (2018). *Neural Networks and Deep Learning*. Cambridge International Law Journal.⏎
19. Khan, L. U., Yaqoob, I., Tran, N. H., Kazmi, S. M. A., Dang, T. N., & Hong, C. S. (2020, Oct). Edge-Computing-Enabled Smart Cities: A Comprehensive Survey. *IEEE Internet of*

Things Journal, 7(10), 10200–10232. doi: 10.1109/JIOT.2020.2987070.⏎

20.Raafat, H. M., Hossain, M. S., Essa, E., Elmougy, S., Tolba, A. S., Muhammad, G., & Ghoneim, A. (2017). Fog intelligence for real-time IoT sensor data analytics. *IEEE Access*, 5, 24062–24069.⏎

21.Xiao, Y., & Krunz, M. (2021). AdaptiveFog: A modelling and optimization framework for fog computing in intelligent transportation systems. *IEEE Transactions on Mobile Computing*, 21(12), 4187–4200.⏎

22. Mchergui, A., Hajlaoui, R., Moulahi, T., Alabdulatif, A., & Lorenz, P. (2024). Steam computing paradigm: Cross-layer solutions over cloud, fog, and edge computing. *IET Wireless Sensor Systems*, 14(5), 157–180.⏎

23. He, J., Wei, J., Chen, K., Tang, Z., Zhou, Y., & Zhang, Y. (2017). Multitier fog computing with large-scale IoT data analytics for smart cities. *IEEE Internet of Things Journal*, 5(2), 677–686.⏎

24. Alam, S., Shuaib, M., Ahmad, S., Jayakody, D. N. K., Muthanna, A., Bharany, S., & Elgendy, I. A. (2022). Blockchain-based solutions supporting reliable healthcare for fog computing and Internet of medical things (IoMT) integration. *Sustainability*, 14(22), 15312.⏎

25. Vankayalapati, R. K. (2020). Green Cloud Computing: Strategies for Building Sustainable Data Center Ecosystems. *SSRN 5079773*.⏎

26.Alpaydin, E. (2020). *Introduction to Machine Learning* (4th ed.). MIT Press, Vol. 1, no. 1, p. 712.⏎

27.Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer, Vol. 1, no. 1, p. 778.⏎

28.Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press, Vol. 1, no. 1, p. 219.⏎

29. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). Springer.⏎
30. Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4), 427–437.⏎
31. Mustapha, S. D. S., & Gupta, P. (2024). DBSCAN inspired task scheduling algorithm for cloud infrastructure. *Internet of Things and Cyber-Physical Systems,* 4, 32–39.⏎
32. Gupta, P., Rawat, P. S., kumar Saini, D., Vidyarthi, A., & Alharbi, M. (2023). Neural network inspired differential evolution based task scheduling for cloud infrastructure. *Alexandria Engineering Journal,* 73, 217–230.⏎
33. Madhusudhan, H. S., Gupta, P., Saini, D. K., & Tan, Z. (2023). Dynamic virtual machine allocation in cloud computing using elephant herd optimization scheme. *Journal of Circuits, Systems and Computers*, 32(11), 2350188.⏎
34. Rawat, P. S., Gaur, S., Barthwal, V., Gupta, P., Ghosh, D., Gupta, D., & Rodrigues, J. J. C. (2025). Efficient virtual machine placement in cloud computing environment using BSO-ANN based hybrid technique. *Alexandria Engineering Journal,* 110, 145–152.⏎

Chapter 4

Blockchain-Based Secure Data Sharing System in Fog–Edge System

Harshitha Kamal Kannan

DOI: 10.1201/9781003610168-4

4.1 Introduction

With the rise of technology in all the domains such as healthcare, industry and agriculture sector with internet of things (IoT), data generation has become huge. It is important to store the data in an efficient manner and to build a network with low latency, high bandwidth specifications and high security. The traditional system fails to provide all the requirements of the specifications [1]. To overcome these challenges, we adopt blockchain-based fog–edge computing systems. This is a critical issue as the data is decentralized and shared among multiple devices and edge nodes and ensuring integrity, access control and confidentiality is a challenge.

Fog computing provides a decentralized architecture which extends to cloud by inculcating the storage, data processing and analytics close to the source of data such as edge devices and IoT devices instead of transmitting the data to the cloud through the gateways [2]. Intermediate fog layer exists between the cloud and the devices to bring out improvements in efficiency and to reduce latency and bandwidth constraints connected with the network, thereby allowing better usage of resources and faster decision-making. The fog computing functions as follows: after the devices generate the data, the nearby fog nodes analyze and filter the data. Next, if any real-time action is required, fog executes it based on the defined rules. Critical data and long-term data are transmitted to the cloud for in-depth analysis, complex computations and storage [3].

Edge computing on the other hand processes and stores the data at the network's edge where the data is generated and consumed. This enables faster decision-making and quicker real-time analysis, thereby reducing the latency. It also leads to lower bandwidth consumption. Edge computing involves devices like gateways, edge servers, IoT sensors and mobile devices. Only the critical data and the data that needs more processing like advanced analytics and deep learning models are sent to the cloud [4]. The future of edge computing involves AI-powered edge analytics involving real-time image recognition and predictive analytics. Edge computing as services allows businesses to leverage edge computing resources without major infrastructure investments. Edge computing integrated with blockchain forms a tamper-proof network and enhances data authentication and integration.

Blockchain provides high security, transparency and decentralization, making it an important component in the digital economy. It provides distributed ledger which is digital by nature. It involves hash functions and records the hash function of the previous block. It also maintains the timestamp of all the transactions. After a transaction is initiated, a peer-to-peer network between nodes is initiated. Then, the Proof of Work or Proof of Stake is used to verify the consensus mechanism [5]. Upon, the validation it is grouped with the blocks in the blockchain network. All the transactions are permanent and immutable, ensuring consistency and transparency.

Hence, combining the three technologies gives an efficient solution to the modern data problems.

4.2 Review of Literature

The authors in [6] propose lightweight shared validation conventions for IoT frameworks utilizing actual unclonable capabilities. Two conventions are presented: one for secure correspondence between two IoT gadgets and the other for correspondence between an IoT gadget and a server. The proposed plot gives different safety efforts for various ongoing applications regarding calculation, energy and other continuous boundaries. For large-scale IoT deployments, verifying IoT clients poses significant technical challenges. In any case, blockchain technology gives a decentralized trust model where each IoT client can be verified without depending on the focal power [7].

Elliptic bend cryptography (ECC) has ended up being an effective method; furthermore, it is efficient for verification in different applications, particularly in asset-obliged conditions like IoT and remote sensor networks. In [8], the authors proposed a lightweight and secure authentication strategy utilizing ECC explicitly intended for the client passage (U-GW) IoT network model. This technique is pivotal for guaranteeing secure correspondence in asset-compelled conditions run of the mill of IoT gadgets.

This approach uses a conveyed record to keep up with network data safely while guaranteeing common authentication among vehicles and road side units (RSUs) in vehicular haze figuring climate. The proposed conspire results uncover that it improves security highlights as well as decreases computational and communicational overheads, making it a practical choice for genuine world applications [9, 10].

In [11], the authors proposed a two-factor RSA-based authentication convention for multiserver conditions. The exhaustive examination and reproduction results prove its adequacy against various security threats while keeping up with low computational overheads.

The proposed convention utilizes a blend of passwords and shrewd cards, alongside the RSA cryptosystem [12]. According to the authors it addresses known security issues, upholds meeting key arrangement, and guarantees common validation between the client and the application server. The accuracy of common verification and the newness of the meeting key are additionally approved utilizing the Boycott rationale model [13].

All things considered, the outstanding constraints of their work include recollecting the passwords, dealing with the conveyance of brilliant cards in enormous scope frameworks. ECC has been proved to be an effective method; besides, robust reply for confirmation across various applications, particularly in asset-obliged conditions like IoT and remote sensor networks [14, 15].

In [16], the authors have proposed a book validation and key-trade conspire that influences blockchain innovation and ECC. This approach uses a conveyed record to keep up with network data safely while guaranteeing common authentication among vehicles and road side units (RSUs) in vehicular haze figuring climate. The proposed conspire results uncover that it improves security highlights as well as decreases computational and communicational overheads, making it a practical choice for genuine world applications [17, 18].

They actually address known security dangers, upholds meeting key arrangement, and guarantees common validation between the client and the application server [19]. The accuracy of common verification and the newness of the meeting key are additionally approved utilizing the Boycott rationale model. All things considered, the outstanding constraint of their work is recollecting the passwords, dealing with the conveyance of brilliant cards in enormous scope frameworks with numerous servers.

In [20], the authors proposed a lightweight and security safeguarding two-factor confirmation conspire explicitly intended for Internet of Things (IoT) gadgets. The proposed conspire is demonstrated to be efficient against different types of assaults, including physical and cloning assaults. However, the prominent impediment is that the conspire depends largely on the viability of Physical Unclonable Functions (PUFs) as a subsequent confirmation factor. If the PUFs do not perform dependably because of ecological factors or assembling irregularities, the general security of the validation cycle may be compromised [21].

4.3 Methodology

Blockchain-based secure data sharing systems enhance security, efficiency and privacy in data exchange nodes.

Traditional cloud-based approaches face challenges such as high latency, vulnerability and inefficient resource utilization, which are overcome by blockchain approaches. This section gives an overview of systems deployed using blockchain in fog–edge computing.

Blockchain can be used with IoT applications. The suggested framework encompasses various stages, such as device configuration, enrollment, verification, key exchange and preservation with data retrieval stage. As the data generates from the IoT devices, it must be safeguard and the integrity and security of data transactions has to be preserved. Merkle tree-based data integrity is deployed for this purpose. Elliptic curve Diffie–Hellman, which is a hybrid encryption method, is utilized for data encryption and decryption. A case study of electronic healthcare record (EHR) of a hospital secured with the help of blockchain is considered.

As the number of cardiovascular diseases is increasing, an IoT wearable device is attached to the patient to continuously monitor the electrocardiogram (ECG). It collects the data and transmits heart rate and rhythm data for the cloud, which is centralized, for real-time analysis by various doctors. The system uses fog-assisted EHR which uses blockchain technology that integrates all kinds of IoT sensors and systems to monitor and analyze the collected data. It processes the ECG data at the edge of the network. Figure 4.1 shows the architecture of the system. The entities involved are hospital, fog gateway, fog node, distributed blockchain-based storage and medical specialist.

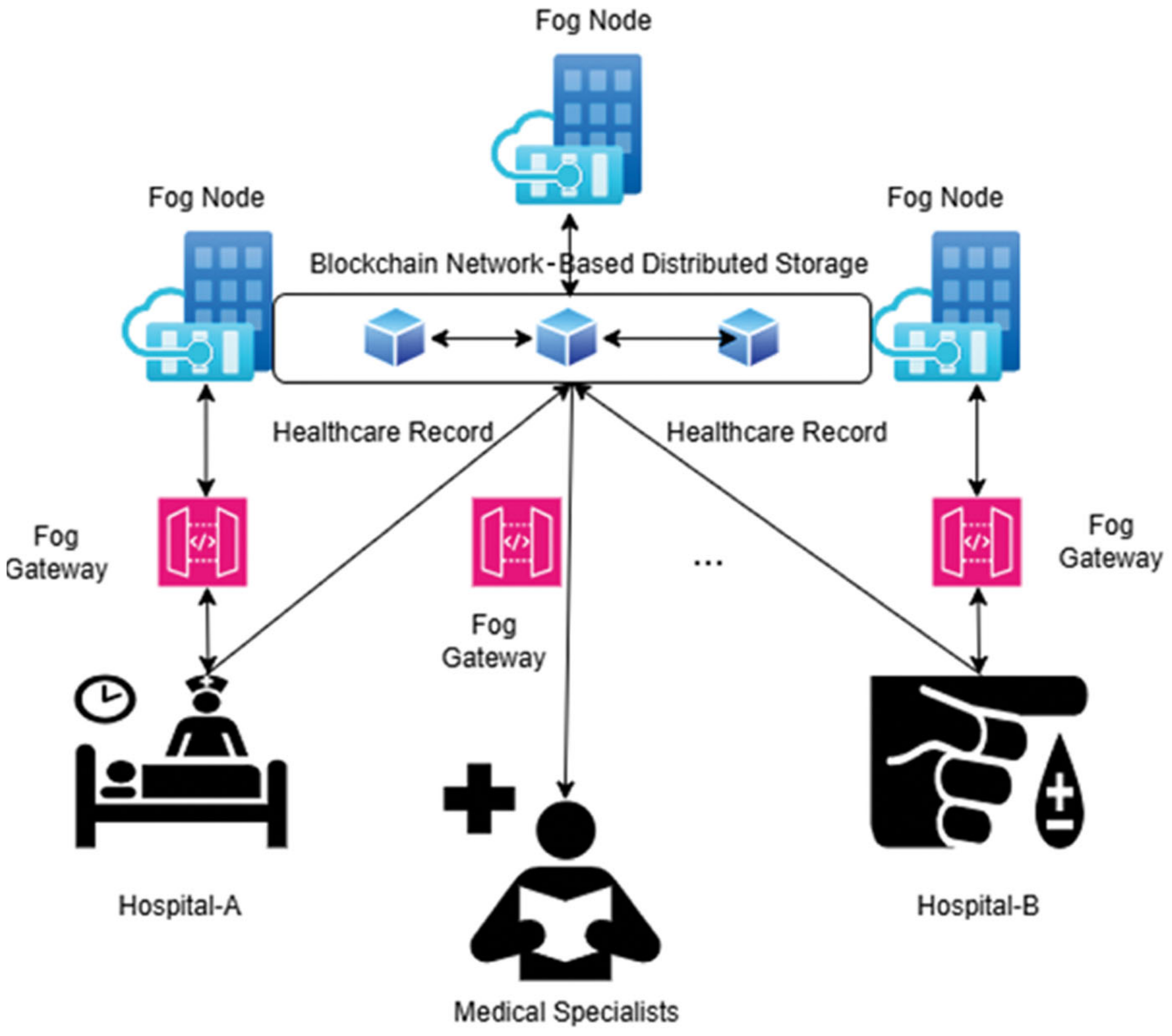


Figure 4.1 Blockchain-enabled fog-assisted smart healthcare system.⏎

Model generates the threats such as data integrity attacks in fog-enabled systems to protect the intruders from false readings and altering vital signs which can mislead the medical decisions. It also prevents data confidentiality attack to protect patient's privacy data and also prevents unauthorized access to the patient's data, and the authentication is given only to doctors, healthcare devices and nurses.

The IoT nodes form a network to communicate the data for sharing among each other in a network. Two software agents are deployed at the fog: one is used to form and monitor the network and the other is used to establish communication among the devices. Before adding a new device to the network, approval is taken from the connected network and then the device is added.

The IoT node generates the data and stores it in the form of the block. The blockchain network contains the hash value of the previous block and the current block; SHA 256 is used for hashing. Each block contains the timestamp of when it is added, and the block sizes are predefined. Proof of Work, which checks whether the user is trying to interfere with the block, needs to be checked by the IoT device and must be verified before a block is added to the blockchain.

SAN is responsible for the peer-to-peer networking of IoT devices, which is meant to support data sharing. To ensure the security, the following steps are followed: a group key is generated, AES is used to share securely and secure hash algorithm (SHA) is used for finding the hash value. Figure 4.2 shows the architecture of the system.

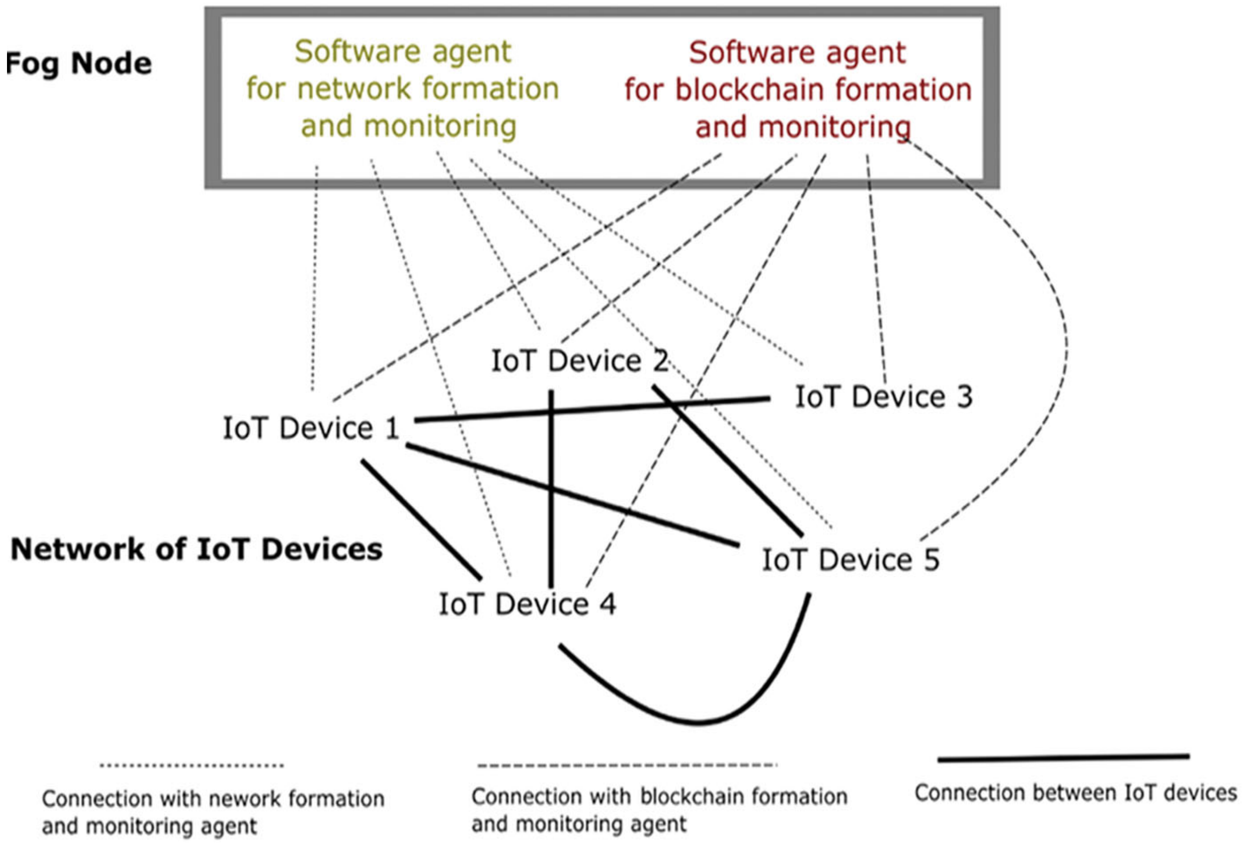

Fog Node
Software agent for network formation and monitoring
Software agent for blockchain formation and monitoring
IoT Device 1
IoT Device 2
IoT Device 3
IoT Device 4
IoT Device 5
Network of IoT Devices
Connection with nework formation and monitoring agent
Connection with blockchain formation and monitoring agent
Connection between IoT devices

Figure 4.2 System model.⏎

A blockchain-based information-centric network computing in cloud, edge and fog services can be designed based on decentralized and secure data sharing environment which increases the efficiency of cloud services. The defined architecture has three layers, namely, an application layer which provides the interface to the end users, a blockchain layer that provides data security and a network layer that provides routing services.

This system provides improved security, efficiency and transparency when compared to the traditional cloud system. The integration of the whole system depends on the smart contracts, which also provides the automation of the services. In fog computing, most of the work is carried out on the edge devices which focus on computation of data storage. Usually, edge devices are often limited in power and memory.

The impact of blockchain on information-centric storage which includes the IoT devices, cloud computing provides an overall security of the systems. Another impact is that it gives rise to business models, which helps to monetize the data that generates revenue for organizations and individuals.

Some of the limitations of these networks are that they use large number of blocks for transactions which cause the system to slow down and cause latency. Another limitation is that interoperability between different systems leads to security concerns.

Next, the blockchain-based edge computation can also be used in the industrial networks. Industry 4.0 requires edge-based distributed resources for maintaining the real-time industrial processes. It uses multi-access edge computing and 5G technology to provide high performance to the localized applications.

The proposed work consists of three parts: IoT edge network, fog networks and cloud networks integrated with the blockchain services. The case study discusses the use of smart building construction that involves various contractors and sub-contractors. The builder further allocates various tasks of the contractors. The main requirement of the use case is that it requires low latency services, trusted data sharing, optimized scalability, secure process monitoring, secure offloading and authenticated access control.

The various elements used here are different types of IoT nodes available at the local device level. Next, there are IoT clusters which include sub-contractors who provide services according to the agreement. The edge nodes have a greater number of resources when compared to other nodes of the network. The system uses a lightweight blockchain model for data sharing. The fog nodes have more capability for processing data when compared to other nodes. It makes use of centralized cloud for storage. Figure 4.3 shows the architecture of the system.

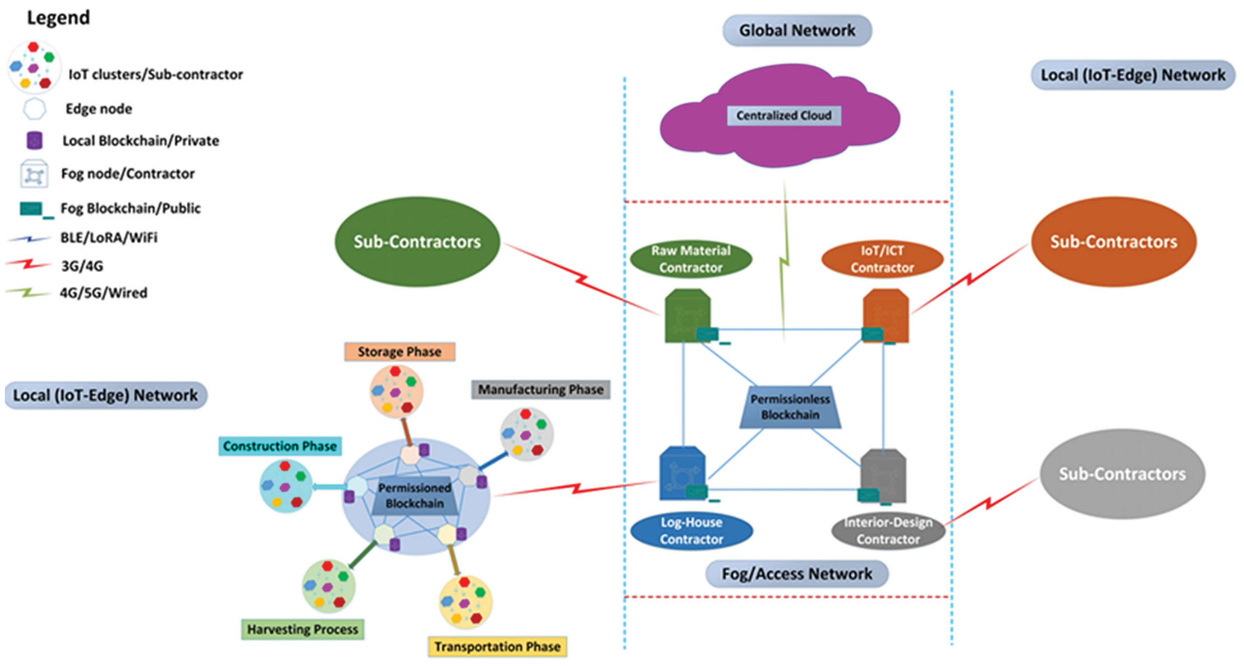


Figure 4.3 Architecture of smart building use case.⏎

Another system uses Hyperledger fabric to track the hash values of medical data. These values are calculated by using the SHA256 algorithm to define the authentication of data. Hyperledger fabric is based on blockchain systems which open-source and use plug and play components. This Hyperledger fabric along with consensus upholds the company data in protection mode. This also supports peer-to-peer network and maintains a common ledger to record transactions.

The architecture comprises three layers: smart contracts, consensus and a distributed ledger. The fog layer acts as a connecting layer between the end user and cloud layer and supports distributed computing to link peripheral nodes to the cloud. The fog layer processes the data at the data sourcing nodes to reduce the bandwidth. It is also connected with the Hyperledger fabric. Figure 4.4 shows the architecture of the system. The hash value is computed in the cloud and compared with the values of the ledger.

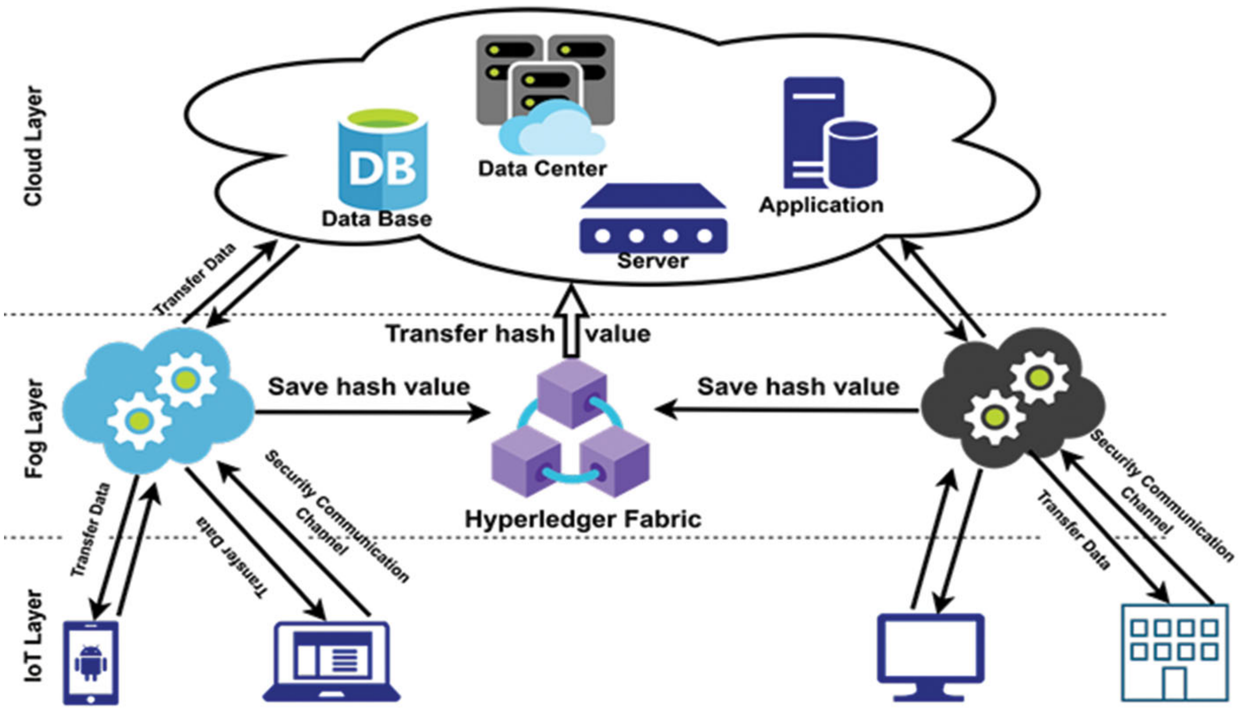


Figure 4.4 Integration of fog computing with Hyperledger.⏎

The flow of the system is as follows: first, the fog collects the data from the IoT devices. Next, the hash value is computed at the fog and compared with the Hyperledger along with which the original data is transmitted to the cloud. Next, the cloud computes the hash value of the data and compares with the value of Hyperledger. If both values match, then no tampering of data has occurred.

The model exhibits a storage overhead involving additional costs because of generation of private keys.

The next concern is about enabling the fog resources access to the services. Considering the Bitcoin system, a blockchain system is developed to maintain the distributed database that records all the blocks using data structures to prevent the data from tampering.

A smart contract is a software that facilitates the agreement between untrusted parties as they do not rely on the third party to operate. When smart contracts are used with fog resources, they provide decentralized authentication and granting resources for each node. The subscriber accesses the resource by decrypting the access key.

The smart contract is dynamically deployed for resource granting. Several Go-Ethereum clients together form a blockchain and transfer the devices into Ethereum nodes. Figure 4.5 shows the structure of the contract built on this concept. A caller sends a few digital coins to the provider to fulfill the agreement, then the secret code is generated automatically, which is then encrypted with the caller's address. The executor's signature is taken when the contract performance is confirmed.

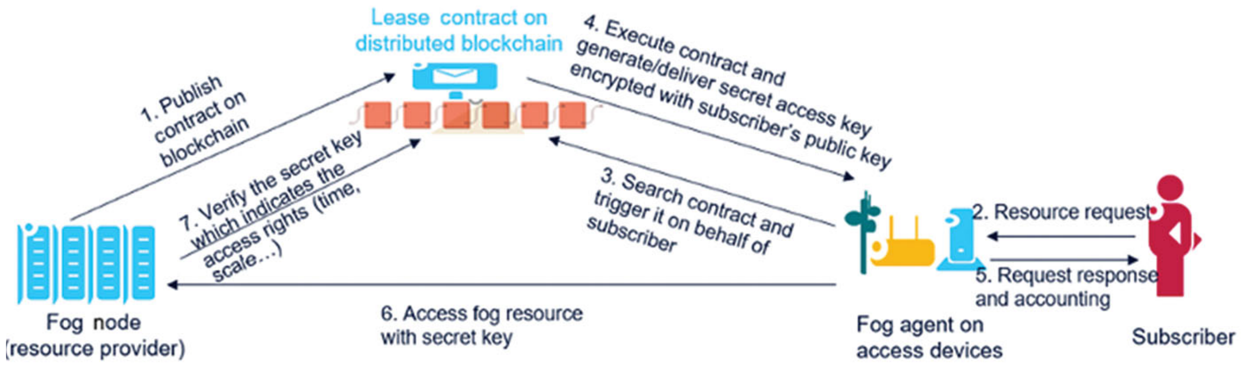

Lease contract on distributed blockchain
1. Publish contract on blockchain
4. Execute contract and generate/deliver secret access key encrypted with subscriber's public key
7. Verify the secret key which indicates the access rights (time, scale...)
3. Search contract and trigger it on behalf of subscriber
2. Resource request
5. Request response and accounting
6. Access fog resource with secret key
Fog node (resource provider)
Fog agent on access devices
Subscriber

Figure 4.5 Fog agent deployed on access device.⏎

The next system describes fake new detection in the mobile edge computing. The system collects data from social media platforms over various phones and detects fake data. This system also uses Natural Language Procession (NLP) approaches to build specific number of words.

The system is built on three layers: an edge layer to process the social media news, and to transfer to the next layer using Transmission Control Protocol (TCP). The next layer is the server layer which receives the messages from the source layer. The third layer is the edge layer and offloading which uses NLP techniques to determine the fake data. The technique used here removes the punctuation in order to recognize the words. Lastly, it is sent to the cloud which has the pre-trained BERT (Bidirectional Encoder Representations from Transformers) model to detect fake news and original news. The architecture is shown in Figure 4.6.

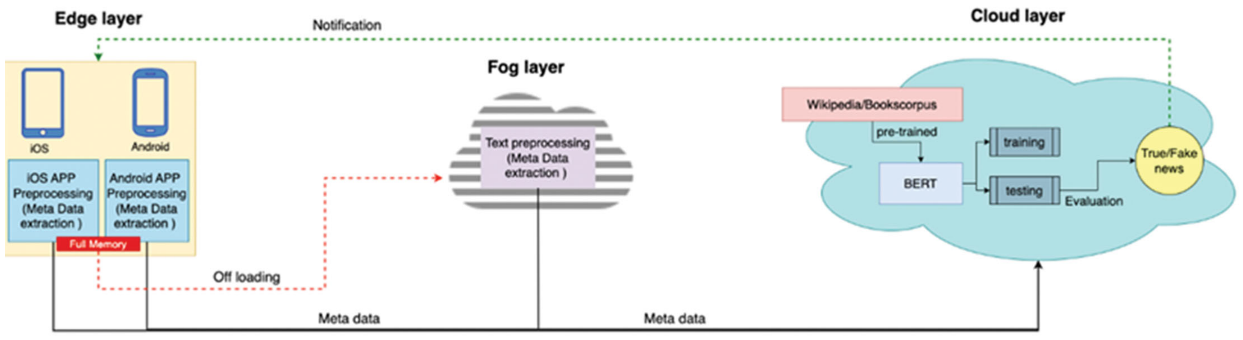


Figure 4.6 EdgeFNF system architecture.⏎

A Fog Chain model can be deployed for personal health records which integrate IoT and blockchain. Fog computing binds the blockchain layer and IoT layer. The latency and network congestion issues in IoT and the Internet of Health Things (IoHT) are overcome in this system. The real-time data processing is enabled by FogChain. This model upgrades the IoHT and blockchain integration which reduces the latency at the edge. The model also focuses on securing the personal health record safer storage. Figure 4.7 shows the model.

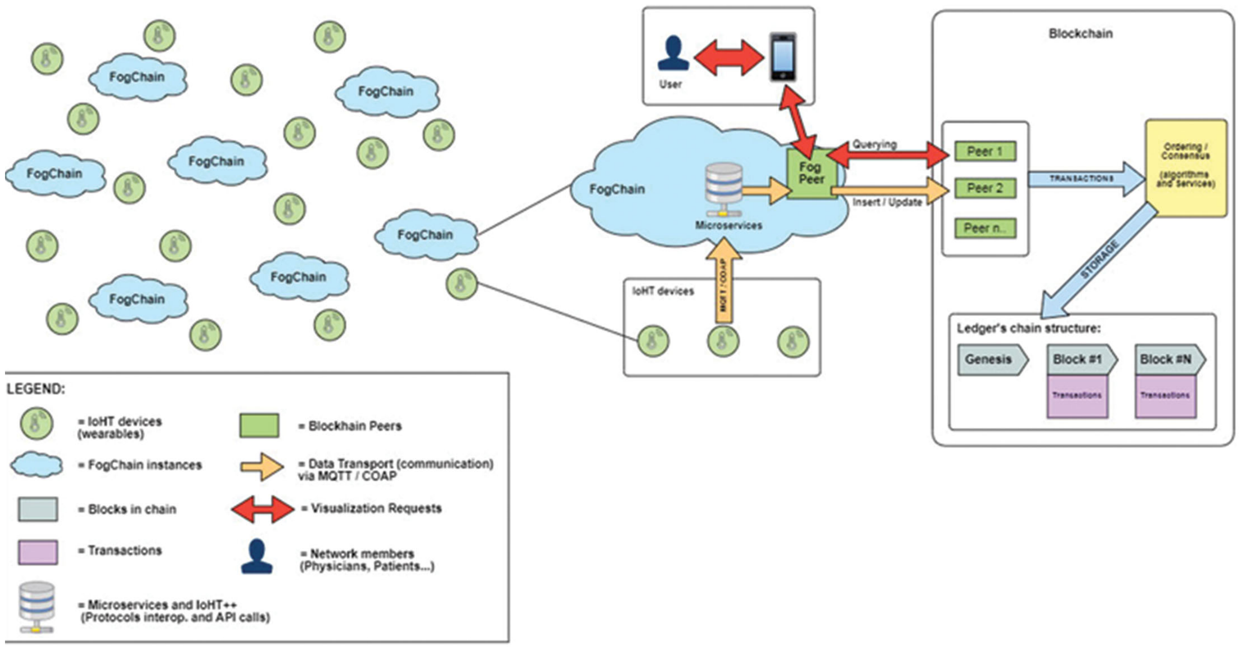


Figure 4.7 FogChain visualization.⏎

The four main components of the architecture are as follows: The first layer is the IoHT layer which provides interaction among the devices and supporting protocols for interoperability. It also supports Wireless sensor network (WSNs) which has low storage capacity. The communication protocols also include Hypertext Transfer Protocol (HTTP), Constrained Application Protocol (CoAP) and Message Queuing Telemetry Transport (MQTT).

The second layer in the architecture is the fog layer, which has the technology for scaling solutions for computing and is able to providescalability. It deals with validating and filtering the data and also deals with communication protocols, which constitute the microservices of the model. It acts as an entry point and uses Apache Kafka for establishing publisher subscriber framework.

The third layer is the blockchain peers that form a consortium network for securing data. It mainly supports data storage and follows standard data formats. It supports blockchain inside the fog instances to overcome the overloading of CPU storage capacity. The consensus is a multiple-step approach that notifies the ledger updates once the process is complete.

The last layer is smart contracts which contain the protocols and programs stored in the blockchain, which has the ability to track the agreements among the parties. These smart contracts are executed in the background automatically. It also triggers the alerts or notification at the patient’s end.

The next discussion is about heterogeneous, interoperable and distributed architecture (HIDRA) for fog and edge computing environments. It is designed for resource management in fog computing environments. The architecture involves a group of heterogeneous devices which cover an area such as a building. Each cluster runs on a blockchain service. Each node monitors its own resources such as CPU usage. Other nodes can also exchange their usage and establish communication among themselves.

The architecture contains three layers: the first one is the device layer, which contains the IoT devices, and the second layer is the distributed fog layer, which comes close to IoT devices. This layer is close to the edge to facilitate the devices with scalability and reliability. These nodes are connected by blockchain technology and also establish peer-to-peer network. The use of smart contracts ensures that every node is equally involved in the network. Also, fog uses operating system (OS) virtualization to automate the deployment of applications.

The network maintains a distributed registry to store the logs of all the applications that were running. It includes the IP address and port with the resource specification needed to run it. The system involves a monitor and a manager. The monitor checks the state of each node in the cluster and alerts the manager in case of policy discrepancies, while the manager checks the working of virtualization in the system. The system architecture is shown in Figure 4.8.

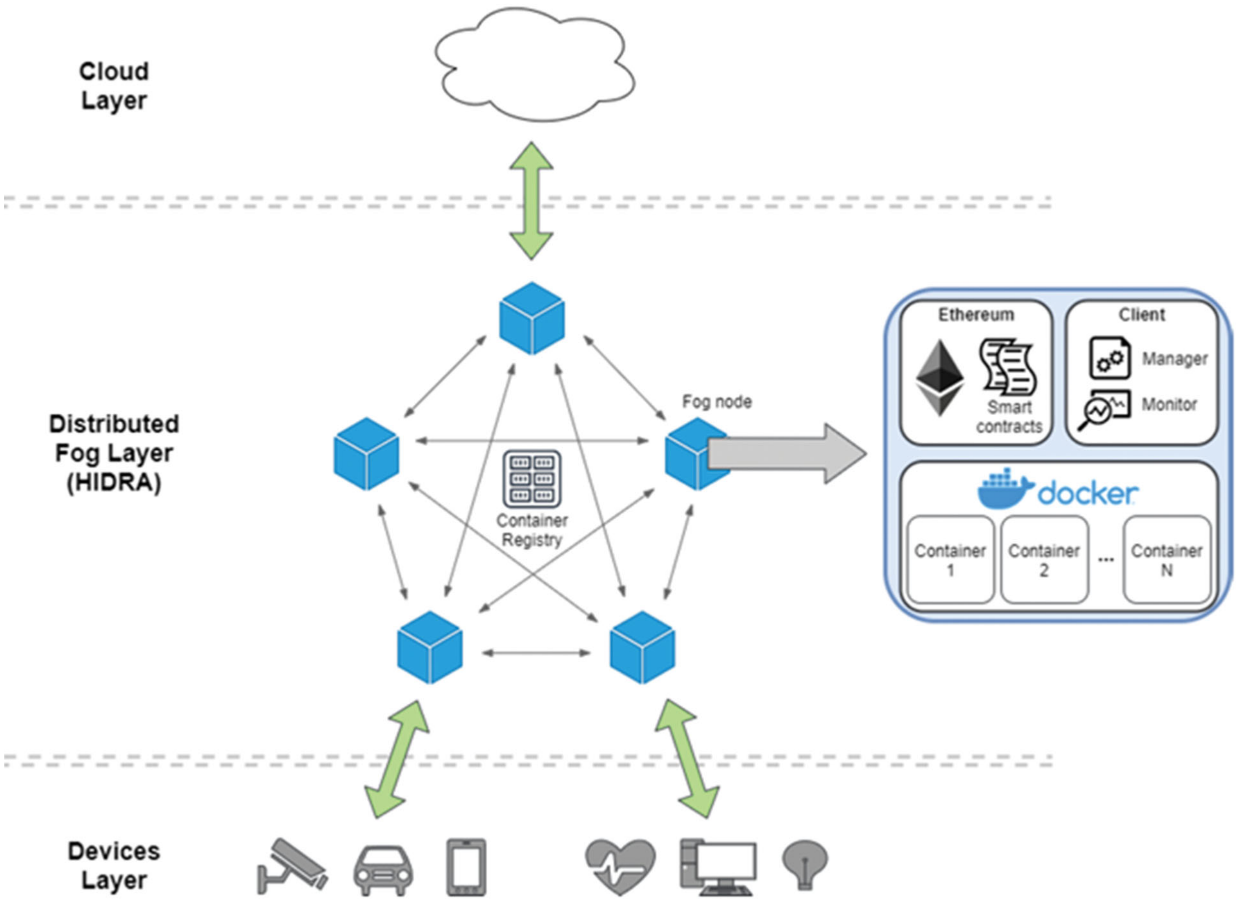


Figure 4.8 System model using HIDRA.⏎

The next system model depicts the blockchain chain system with fault tolerance for IoT devices with fog computing. The architecture contains six entities: The first one is the data owner, who defines the access policy and manages the registration of devices in the network.

The second entity is the user who requests certain requests from the IoT network who is also responsible for encryption and obtains the authorized token if the transaction is valid through smart contracts.

The third entity is the blockchain to store the policies of the network and manage the fog nodes. All activities are managed in a decentralized manner in the network.

The fourth entity includes the fog nodes which provide the fog services and virtualization to reduce the latency and to improve the quality of services provided by the system.

The fifth entity includes the IoT devices, where each device is mapped to fog nodes to check for fault tolerance, with limited processing capacity.

Lastly, the cloud represents the sixth entity which aggregates the data from the fog nodes.

The system works as follows: first, the data owner associates with the blockchain by providing the key authentication and converts the encryption to smart contracts policies. Next, the IoT devices get registered in the network. Next, a user process is submitted to generate a request, which is redirected to blockchain to check for the validity of the request. The system contains the load balancing mechanism to use the resources in an efficient way. Load balancing is carried out by using min-min algorithm. The system prevents attacks such as Man-in-the-Middle, replay and DDoS attack. The overall architecture of the system is shown in Figure 4.9.

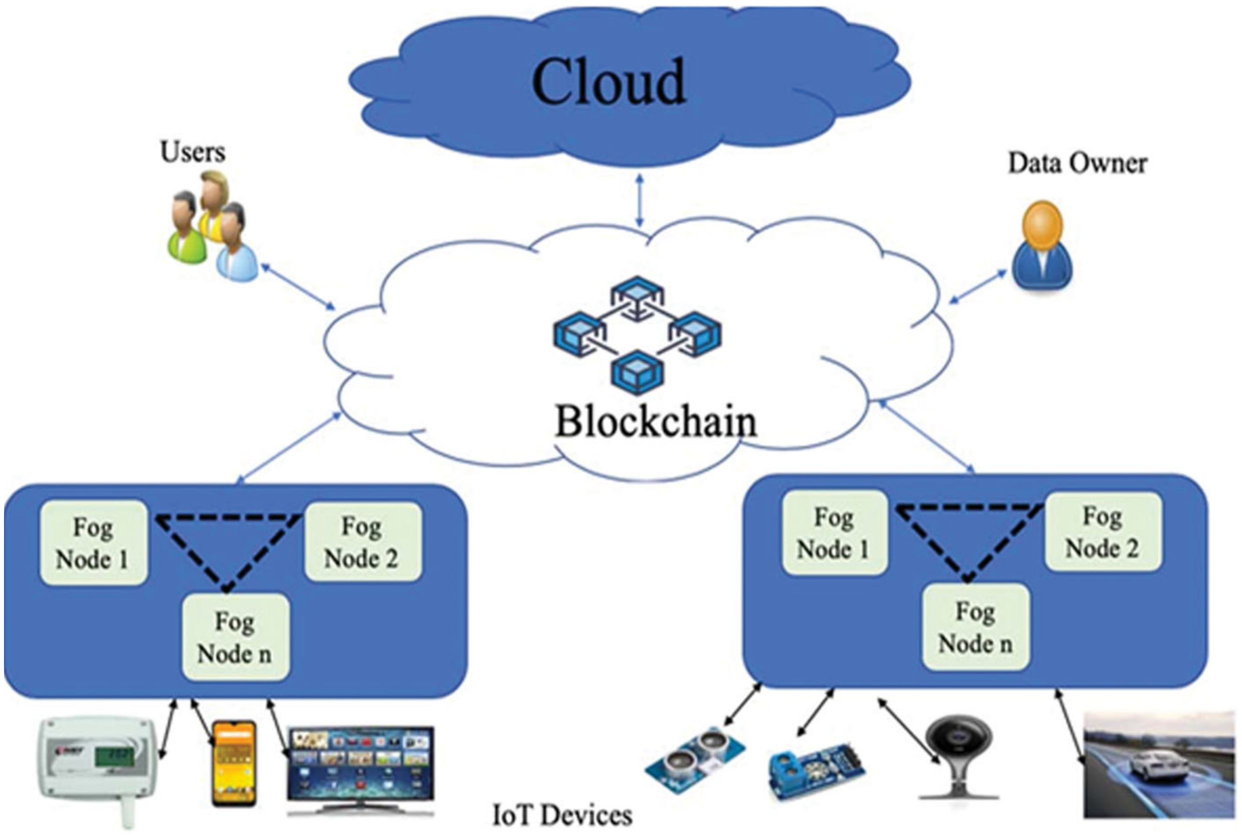


Figure 4.9 System architecture with fault tolerance.⏎

4.4 Conclusion

The integration of blockchain with fog–edge computing gives a comprehensive approach to obtain decentralized and efficient data sharing in the IoT environments. Traditional cloud systems exhibit limitations such as security vulnerabilities, centralized points of failure, bandwidth limitations and latency. These blockchain-enabled systems show a scope of improvement in all the drawbacks of traditional systems. In this chapter, we investigate how blockchain innovates with a decentralized approach and gives a designed answer for securing the information. We inspect architectural frameworks, consensus algorithms, security mechanisms and smart contracts functionalities for real-world applications. Blockchain strengths data security in Fog-environments, enabling trusted and tamper-proof data exchanges among the IoT nodes.

References

1. Hemant Kumar Apat, and Bibhudatta Sahoo, “A Blockchain Assisted Fog Computing for Secure Distributed Storage System for IoT Applications”, *Journal of Industrial Information Integration*, 42, 100–739, 2024.⏎
2. Debasis Mohapatra, Sourav Kumar Bhoi, Kalyan Kumar Jena, Soumya Ranjan Nayak, and Akansha Singh, “A Blockchain Security Scheme to Support Fog-Based Internet of Things”, *Microprocessors and Microsystems*, 89, 104455, 2022.⏎
3. Pranav Zambre, Mitali Panchal, and Ankit Chouhan, “Blockchain Unleashed: Empowering Information-centric Network Computing in Cloud, Fog and Edge Service”, In *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pp. 1–6, IEEE, 2023.⏎
4. Manish Kumar Gupta, and Rajendra Kumar Dwivedi, “Blockchain-Based Secure and Efficient Scheme for Medical Data”, *EAI Endorsed Transactions*, 10(5), 2023.⏎
5. Gang Liu, Jinsong Wu, and Ting Wang, “Blockchain-Enabled Fog Resource Access and Granting, Intelligent and Converged Networks. *Intelligent and Converged Networks*, 2(2), 108–114, 2021.⏎
6. M.N. Aman, K.C. Chua, and B. Sikdar, “Mutual Authentication in IoT Systems Using Physical Unclonable Functions”, *IEEE Internet of Things Journal*, 4(5), 1327–1340, 2017.⏎

7. Sawsan Alzubi, and Feras M. Awaysheh, “EdgeFNF: Toward Real-Time Fake News Detection on Mobile Edge Computing”, In *2022 Seventh International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 1–3. IEEE, 2022.⏎
8. C. Patel, “*Secure Lightweight Authentication for Multi User IoT Environment*”, arXiv preprint arXiv:2207.10353, 2022.⏎
9. André Henrique Mayer, Vinicius Facco Rodrigues, Cristiano André Da Costa, Rodrigo Da Rosa Righi, Alex Roehrs, and Rodolfo Stoffel Antunes, “*FogChain: A Fog Computing Architecture Integrating Blockchain and Internet of Things for Personal Health Records*”, *IEEE Access*, 9, 122723–122737, 2021.⏎
10. Carlos Nunez-Gomez, Carmen Carrion, and Blanca Caminero, “HIDRA: A Distributed Blockchain-Based Architecture for Fog/Edge Computing Environments”, *IEEE Access*, 9, 75231–75251, 2021.⏎
11. Ruhul Amin, S.K. Hafizul Islam, Muhammad Khurram Khan, Arijit Karati, Debasis Giri, and Saru Kumari, “A Two-Factor RSA-Based Robust Authentication System for Multiserver Environments”, *Security and Communication Networks*, 2017(1), 5989151, 2017.⏎
12. Oussama Mounnan, Abdelkrim El Mouatasium, Otman Manad, Tarik Hidar, Anas Abou El Kalam, Noureddine Idboufker, “Privacy-Aware and Authentication Based on Blockchain with Fault Tolerance for IoT Enabled Fog Computing”, In *2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 347–352. IEEE, 2020.⏎
13. Shyamalendu Kandar, and Abhipsho Ghosh, “Smart Card Based Remote User Authentication Scheme in Multi-

server Environment using Chebyshev Chaotic Map", *Wireless Personal Communications*, 133(4), 2657–2685, 2024, 10.1007/s11277-024-10895-w.⏎

14. Shuwan Sun, Weixin Bian, Dong Xie, Deqin Xu, and Yi Huang, "Lightweight and Privacy-Preserving Multi-server Authentication Scheme Based on PUF and Biometrics", *Journal of Intelligent & Fuzzy Systems*, 45(1), 911–928, 2023, 10.3233/JIFS-221354.⏎
15. Y. Liu, J. Zhang, and J. Zhan, "Privacy Protection for Fog Computing and the Internet of Things Data Based on Blockchain", *Cluster Computing*, 2020.⏎
16. P.K. Sharma, and J.H. Park, "Blockchain-Based Secure Mist Computing Network Architecture for Intelligent Transportation Systems", *IEEE Transactions on Intelligent Transportation Systems*, 22(8), 5168–5177, 2020.⏎
17. S. Oyewobi, S. Misra, and C. Lal, "Mobility as a Service (MaaS) Enabled Fog Computing: An Innovative Solution for Smart City Infrastructure", *Journal of Ambient Intelligence and Humanized Computing*, 10(1), 1–17, 2019.⏎
18. T.A. Rahoof, and V.R. Deepthi, "HealthChain: A Secure Scalable Health Care Data Management System Using BC", In Proceedings of International Conference on Distributed Computing and Internet Technology, 2020, pp. 380–391.⏎
19. G. Li, M. Dong, L.T. Yang, K. Ota, J. Wu, and J. Li, "Preserving Edge Knowledge Sharing among IoT Services: A BC-Based Approach", *IEEE Transactions on Emerging Topics in Computational Intelligence,* 4(5), 653–665, Oct. 2020.⏎
20. P. Gope, and B. Sikdar, "Lightweight and Privacy-Preserving Two-factor Authentication Scheme for IoT

Devices”, *IEEE Internet of Things Journal*, 6(1), 580–589., 2018.⏎

21. G. Srivastava, J. Crichigno, and S. Dhar, “A Light and Secure Healthcare Blockchain for IoT Medical Devices”, *2019 IEEE Canadian conference of electrical and computer engineering (CCECE)* (pp. 1–5). IEEE., 2019.⏎

Chapter 5

Securing IoT System Using ML Models

Arbaz Adib Dalwai, Shivani Jaswal, and Rohit Verma

DOI: 10.1201/9781003610168-5

5.1 Introduction

The Internet of Things (IoT) can be described as one of the revolutionary technologies as it connects things, data and cloud to improve the operational efficiency and integration. But this extended environment has brought about severe weaknesses in IoT environments, exposing them to various types of cyber threats. Out of these, distributed denial of service (DDoS) attacks are rather unique because they flood the resources of the network, making the services nonfunctional. Such threats not only affect the IoT-dependent operations or progressive control but also undermine the Cloud platforms [1] that facilitate such devices and operations. The common security solutions like firewalls and intrusion detection systems (IDS) [2] have been working in reducing many vulnerabilities. However, the growing advancement in attacks also requires stronger and smarter security in the context of IoT. As traditional pattern matching has its drawbacks in analysing and classifying frequently changing attack scenarios in real time and within the context of the IoT architecture's limited bandwidth and computational capabilities, it is necessary to switch to a better and far more resilient system to attacks. These limitations motivated the necessity of building a hybrid system which would combine the strengths of advanced machine learning [3] (ML) models with techniques of systems like IDS and Security Information and Event Management (SIEM) in order to improve the detection and differentiation of DDoS [4] attacks in IoT networks. By integrating the rule-based and ML-based methods into the cloud environment the system's overall detection capabilities can be increased into a large extent.

To encounter the challenges posed by the ever-evolving threats landscape in a dynamic cloud environment, this research investigates the following question:

> How can a cloud-based real-time security monitoring system using SIEM, IDS and machine learning models adaptively and effectively analyse and classify DDoS attack on IoT devices?

The foremost objective of this research is to designing and implementing a hybrid security solution that would integrate the rule-based detection with ML models for enhancing the adaptability of the system in identifying the attack patterns accurately and ultimately securing the IoT devices. The research will then assess the system's performance by testing it under simulations, which will replicate the real-world environments to determine whether it is able to detect and classify the threats in real time.

Figure 5.1 illustrates the integral components of the proposed system.

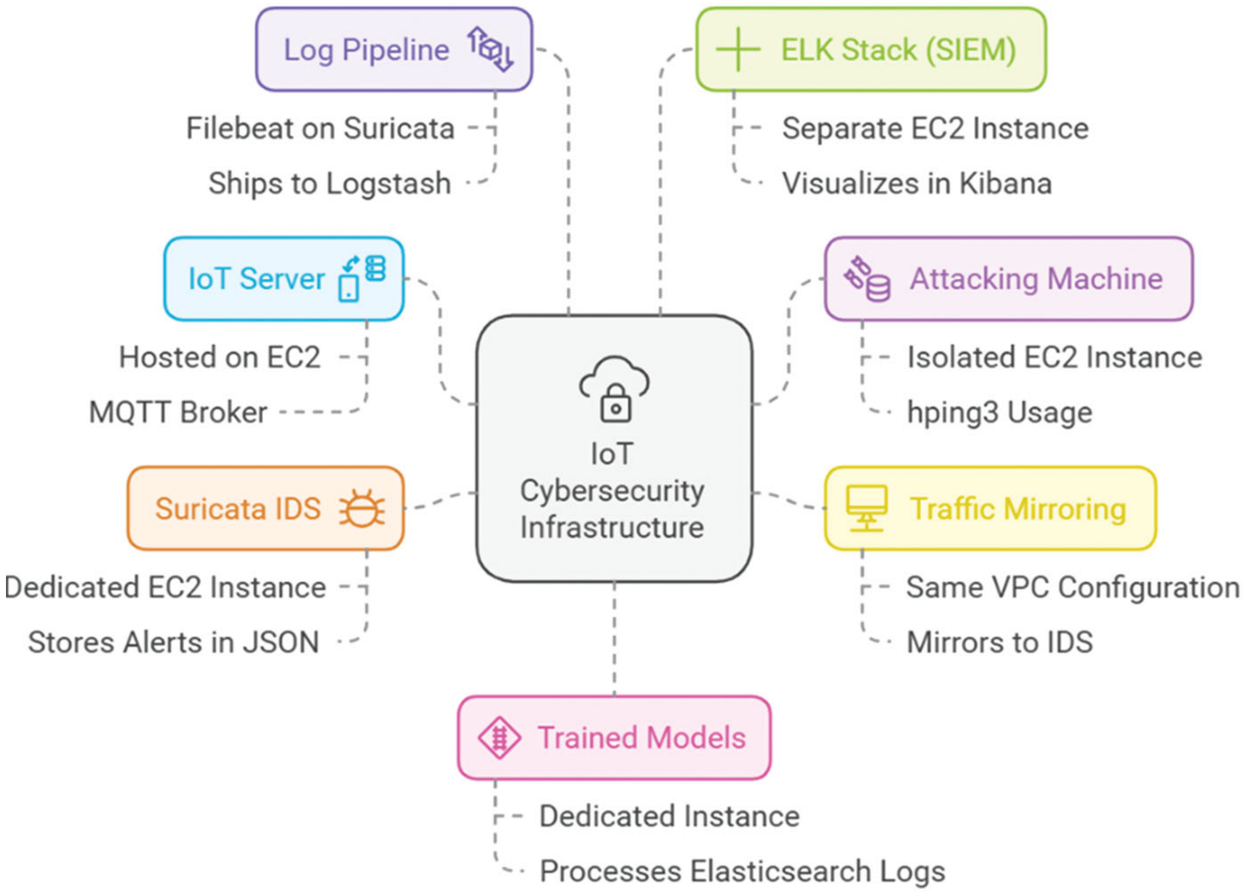


Figure 5.1 Cybersecurity threat detection layout.⏎

This research aims to contribute to the field of cybersecurity through several key points, such as demonstrating a practical integration of SIEM, IDS and ML models in order to create a hybrid detection framework which would be capable of securing the IoT systems through attacks like DDoS in real time. By providing visual alerts through Kibana dashboards, the research also offers real-time insights, helping the analysts in better decision-making. Despite the contributions, no research is without any limitation, including this. The experimental setup relies on simulated attack traffic, which sometimes may not capture the complexities of the attack fully and this arises the need for future research in this area to test the framework in more complex settings.

This chapter is structured as follows: Section 5.2 presents the related work detailing the analysis of existing researches in this area. The overview of the methodology is presented in Section 5.3, containing the information on research framework and experimental plan. Section 5.4 presents the design specifications, while Section 5.5 deals with the implementation, focusing on the technical aspects of the research. Evaluation consisting of the experimental findings and discussion is presented in Section 5.6. Finally, Section 5.7 concludes the chapter while proposing future enhancements.

5.2 Related Work

The world of cybersecurity has witnessed a lot of change in the past few years, with the integration of advanced tools such as SIEM and IDS for real-time threat detection [2,5,6]. Also, a lot of advancements have been made to traditional cybersecurity practices, with the use of ML models into it. These technologies play a vital role in the overall enhancements in the detection and monitoring techniques of different attacks. The following subsections will discuss about the literature review conducted for this research. Key search areas included the integration of SIEM and IDS for real-time monitoring, IoT systems and the attacks/security challenges associated with it, and implementation of ML models for DDoS detections. This literature review helped in forming the base foundation for the methodology and implementation sections of this research.

5.2.1 SIEM, IDS and ML Models in Cybersecurity

The usage of SIEM and IDS solutions has been thoroughly investigated in many of the recent researches defining the importance of these tools in the field of cloud security. A research by Tuyishime et al. (2023) [7] has come up with a SIEM-based approach for threat monitoring in cloud environments through components such as virtual machines, load balancers, and web application firewalls in order to have a central point for logging and analysis. It specifically tries to highlight the importance of SIEM in generating automated alerts to enhance the response times while also trying to evaluate the frameworks' scalability in handling large volumes of data. However, it mainly focuses on the system architecture and operational efficacy, with very limited work on identifying specific attack types. Granadillo, Zarzosa and Diaz (2021) [8] further extend this understanding by providing a comprehensive analysis of existing SIEM solutions, including their functionalities, benefits and limitations. It mainly focuses on the necessity of integrating SIEM in modern cybersecurity infrastructures while also pointing out the challenges the solution has to offer. Likewise, the research by Lee et al. (2017) [9] extends these ideas with a SIEM architecture especially designed for Security-as-a-Service (SECaaS) in the cloud. This research describes the key factors which include scalability, flexibility and multi-tenancy when adapting SIEM systems to cloud native applications. This study however does not have an evaluation setup to test the system in real-world scenarios.

In order to make further improvements in the SIEM technologies, Ayu et al. (2024) [10] have presented the concept of integrating ML with SIEM to improve the detection capabilities and thus showing the potential of ML models in solving the gaps in traditional rule-based systems. Here the researchers have implemented anomaly detection using the Random Forest (RF) classifier on CSE-CID-IDS2018 dataset collected from an IDS which is pre-processed using the principal component analysis. Despite its promising results the study lacks on evaluating the model's performance on real-time data and only focuses on one particular dataset. Similarly, Saeed et al. (2022) [3] present an ML-based IDS which is customised for usage in the cloud environments through the incorporation of a number of algorithms, such as Naive Bayes classifier, decision tree classifier, supporting classifier, logistic regression and RF classifier, to analyse network traffic and detect anomalies. The study stresses on how the implementation of ML techniques can lower down the false positive and enhance the detection capabilities of an IDS in cloud infrastructure. While Innab et al. (2024) [11] present different security attacks in cloud environment while evaluating the different IDS types and techniques that can tackle it, categorising the IDS into signature-based, anomaly-based, hybrid-based and other approaches and analyses, their effectiveness lies in handling and detecting the security events. Their research states that signature-based attacks can effectively detect known threats but fail in unknown threats, anomaly-based attacks are well known for detecting unknown threats but have a greater false positive alert, while hybrid approach balances the two and provides improved detection accuracy. While this research was comprehensive, it lacked any practical implementation to showcase the findings.

5.2.2 IoT Devices

In the study by Munshi et al. (2020) [12], the authors explain the growing susceptibility of IoT devices to DDoS attacks and the seriousness of the security risks arising from the expansion of IoT networks. The paper underlines how integrating the IoT devices improves the usability and the functionality of devices while these devices create new threats and risks addressed by attackers. It highlights fundamental risks like poor or invalid methods of identification and recognising device interaction protocols, which put IoT devices in the same level of risk as other attached devices during DDoS attacks. In turn, to address these threats, the authors suggest that security should be built up and enhanced in the form of better authentication mechanism, security protocols and network monitoring that can identify the attack and neutralise it in real time. This work is of particular relevance to the study of threats in IoT scenarios while emphasising the importance of dynamic methods to address the risks of DDoS attacks. In turn, these findings match the goals of this research and give a theoretical background for considering DDoS threat vectors and designing corresponding detection approaches in IoT-based systems.

5.2.3 DDoS Detection Mechanisms

Extensive researches conducted on DDoS detection have shown various innovative approaches that use SIEM and ML techniques to address the growing complexities of this attack. Çakmakçı et al. (2021) [13] propose a framework for detecting various types of DDoS attacks through SIEM. It encompasses an incident detection engine within the SIEM in order to identify the attack patterns. It also has an inference engine which automatically suggests the response and recovery steps after detection, thus reducing the reaction time for ongoing attacks. Future research on this framework could enhance the detection capabilities of the system when tested in diverse network environments [13]. Complementing this, research by Dhahir (2024) [14] introduces hybrid technique which combines the clustering-based local outlier factor for feature engineering with extreme gradient boosting (XGBoost) for the detection of DDoS attacks. This model was implemented and tested on the CIC-IDS 2017 and CIC-IDS 2018, achieving an accuracy of 99.99% and a precision score of 100%. However, its effectiveness has not been tested in dynamic settings. Similarly, Chen et al. (2018) [15] use the XGBoost algorithm for detecting the DDoS attack in a software-defined networking-based cloud environment. The proposed model had a high level of performance in detecting normal and malicious traffic, hence proving the effectiveness of using the XGBoost. It also depicted that XGBoost can effectively handle large volumes of data in dynamic cloud environments. However, exploring the impact of real-world noisy data with different traffic and attack types would actually test the model's performance which is lacking in this study. Collectively these researches highlight the efficiency of SIEM and advanced ML models, especially XGBoost, in improving the DDoS detections.

5.2.4 Summary

This section explores various existing researches conducted on SIEM, IDS, and advanced ML techniques for detecting different attacks including DDoS in various environments, especially cloud environments. Table 5.1 presents a brief summary of other findings while proposing the solutions that this research has to offer.

Table 5.1 Findings of Existing Techniques⏎

Aspects	*Findings*	*Gaps Identified*	*Research Contribution*
SIEM and IDS	SIEM is considered as an essential component for event management while IDS improves the detection rate.	Insufficient integration of ML models with SIEM and IDS to create new hybrid systems.	Combined the ML models with SIEM and IDS in order to enhance the overall detection rates.
DDoS detection	Results obtained from ML models (RF and XGBoost) have a higher accuracy in identifying DDoS attacks.	Models not tested with real-world dynamic data, thus questioning the scalability.	Framework designed for real-time assessment of ML models for DDoS detection in a cloud environment.
Recurring gaps	Lack of real-time experiments and inadequate combination of rule-based prediction and ML-based prediction approaches.	Inadequate evaluation of results in hybrid systems under real-time conditions.	Proposes a system that integrates rule-based and ML-based predictions in a real-time cloud setup evaluating its effectiveness and scalability.

5.3 Research Methodology

This research is designed with an experimental and quantitative approach methodology, wherein the controlled simulation of attack, its monitoring and analysis using SIEM and IDS reflects the experimental aspect, while the numerical nature of the collected logs and their analysis using ML models and evaluation metrics resembles the quantitative side. The system designed uses both rule-based and ML-based detection techniques in order to identify and analyse DDoS attacks. The methodology can be structured and divided into the following standard research categories.

5.3.1 Research Design

An experimental approach consisting of three main components – simulated attack, monitoring network activity and detection mechanism – was designed to answer the stated research question effectively.

The first component includes the simulation of attack that is DDoS. The attack was launched through an attacking machine setup on the Amazon Web Services (AWS) EC2 instance and the simulations was made sure to be designed in a way that reflects a real-world threat scenario. The DDoS attack was specifically targeted on the IoT server hosted on the EC2 instance using the hping3 command from a docker service on the attacking machine which generated high volumes of traffic. This was the same server which hosts the IoT devices on the MQTT broker. The attack simulation steps were crafted and implemented in order to generate and capture enormous network activity logs on the IDS.

As mentioned previously, the second component was monitoring all the activities in the cloud environment on a real-time basis. For this Suricata, which is an open-source IDS, was deployed on the EC2 instance in the same Virtual Private Cloud (VPC) as that of the IoT server and through traffic mirroring on AWS it was made sure that the IDS is able to capture all the logs depicting the network traffic in the design. The IDS was encompassed with all the emerging threat rules which made sure that the IDS is up to date and would identify different anomalous behaviour which is possibly associated with the launched attacks. Moreover, importing the emerging threat rules also made sure that the system also detects other attacks through a detailed inspection of logs. Filebeat was also installed and configured in the same instance as Suricata in order to send the logs to ELK (Elasticsearch, Logstash, and Kibana) [16] stack. This made sure that the logs received by the SIEM are transferred in real time without any data loss for further processing and analysis.

The final and the most crucial component was detection mechanism. The evaluation of the detection mechanisms can be summarised in two parts: rule-based detections and ML-based detections. The rules were encompassed in the Kibana to identify the specific patterns in the logs to identify DDoS activities. Additionally, ML models (RF and XGBoost) were trained on the IDS logs to classify the attacks. This dual approach in the detection mechanism ensured a comprehensive evaluation of the logs in real time, making sure of a secure network infrastructure deployed completely on the cloud. By combining these components, the research design ensured a robust framework for simulating, analysing and monitoring security threats in a cloud-based environment. This design also made sure that the testing is adhered to standard practices such as the National Institute of Standards and Technology

(NIST) guidelines by following a proper data collection and analysis from relevant sources.

5.3.2 Data Collection

The simulated real-world attack generated datasets in the form of logs (network traffic activity) which was the primary source for data collection in the entire setup. The main objective here was to collect network activity logs that would resemble the DDoS attack which would further be used for rule-based and ML-based detections. To simulate the DDoS attack hping3 command was used from a docker container on the attacking machine hosted on the EC2 instance. This command was targeted on the IoT server ensuring that a high-volume traffic is logged on the same imitating a typical DDoS-like scenario with traffic spikes and abnormal connection requests/attempts. The hping3 command was specifically used due to its flexible approach in sending network packets helping effective log generation.

All these testings were done in a controlled and isolated environment by adhering to the standard ethical testing practices. All these logs were as stated earlier captured on an open-source IDS- Suricata [17]. It was configured in a way to detect all the network traffic in the infrastructure and record the network metadata including the source and destination IPs, ports, protocols, timestamps and other relevant information associated in a genuine network traffic. These logs were structured and stored in the JSON format on the Suricata, which made it convenient for further data processing by the upcoming tools and technologies.

Specific Characteristics of the Logs:

The logs include critical metadata which is explained in Table 5.2.

Table 5.2 Metadata⏎

Fields Captured	*Reason*
Source IP (src_ip)	Origin of network traffic
Destination IP (dest_ip)	Target system or service
Source Port (src_port)	Port linked to the originating application
Destination Port (dest_port)	Service which is being targeted such as TCP/IP(port 1883) reserved for MQTT
Protocol	The network protocol which is used (e.g., TCP and UDP).
Timestamp	Precise time of the recorded activity
Event Type (event_type)	Classifies the nature of network activity whether it is an alert, flow or normal

Fields Captured	*Reason*
Bytes to Server (bytes_toserver)	Total data volume sent to the target server
Packets to Server (packets_toserver)	Number of packets sent to the target server

Characteristics:

- High traffic volume spikes: Reflecting DDoS attack patterns.
- Repeated connection attempts: Possible characteristic of SYN flood attacks.

The logs had special features like high consistency as the Suricata uses ET rulesets which make sure that the events are correctly tagged which is same as realistic environment. Data quality was a key aspect to be taken care of during the collection phase as the entire detection process was relying on the data in IDS and hence filebeat was introduced so that it could transfer the data ahead without dropping the essential metadata fields, ensuring seamless log transmission.

5.3.3 Data Preprocessing

The data preprocessing step mainly focused on the preparation of the network activity logs collected by the IDS for ML-based threat detection. The data preparation involved cleaning and feature engineering from the raw JSON logs stored in the Elasticsearch to make sure that the data on which the models are to be trained is accurate. The process started with filtering of fields from the raw JSON logs. Features including the protocol source IP, destination IP, source port, destination port, bytes to server, bytes to client, packet to server and packet to client were considered very essential in the detection of DDoS attacks. These fields contain information such as source and destination of the network traffic, amount of traffic and number of packets: all critical for scanning for attack-bearing high-volume traffic. This data combined can be used for an effective threat detection. This was then organised into a tabular format by using pandas. Through this process it was made sure that the data is suitable for training and further analysis. Furthermore, numerical encoding was applied to convert the categorical features such as IP addresses into numerical representations. This was done using a custom function which made sure the data is consistent and compatible with the ML models. Features such as src_port and dest_port were retained as it is due to its integer structure. The missing values were addressed either by assigning them with default values or excluding out the incomplete records. This was important as it made sure that the dataset generated contained right data that would be consistent and reliable for analysis.

The data was then labelled to its corresponding simulated attack scenario. The data was mainly tagged into two categories: DDoS and benign. DDoS logs were identified and labelled based on patterns such as high traffic volume (e.g., bytes_toserver exceeding 29,000 and packets_toserver exceeding 650), which is characteristic of SYN flood or other volumetric attack patterns, while the remaining data was tagged as benign. Once the data was completely ready for analysis, it was split into training and testing, ensuring there was no imbalance. Standardisation was also done on the features in order to put the data into correct format for enhancing the model's performance. Thus, this cleaned and pre-processed data was then used to train the RF and XGBoost models ensuring effective evaluation. The cleaned and labelled dataset formed the basis for accurate evaluation of the models in the context of DDoS detection.

5.3.4 Experimental Setup

The setup was designed and executed completely on AWS, making sure that the environment is completely controlled and isolated as it involved simulation of attack, real-time log collection followed by its evaluation. The main components of the setup are explained next.

- IoT Server: The IoT server which is hosted on an EC2 instance has an MQTT broker running, which is an important component for managing IoT device communication. It is a broker providing lightweight, real-time message exchange between IoT devices and their applications. The MQTT broker was chosen because in IoT environments it's a widely accepted IoT device operation hub. The IoT devices were simulated to reflect real-life experience, for instance, smart sensors and smart devices. Node-RED is a widely used flow-based development tool for visual programming mainly used for IoT applications. The simulation setup included virtual devices which would replicate the functions of temperature sensors, humidity sensors, and motion detectors. These devices are set up for continuously sending data to the MQTT broker, creating an actual realistic IoT ecosystem for experimentation purpose. Since the MQTT broker is the heart of the IoT ecosystem, any disruption as a result of a DDoS attack can cause the IoT devices to malfunction, which in turn may affect real-time operations. For this reason, this broker is selected as the primary candidate for DDoS simulations. Figure 5.2 presents the Node-RED design which was set up on port 1880 of the IoT server, simulating the virtual IoT devices.

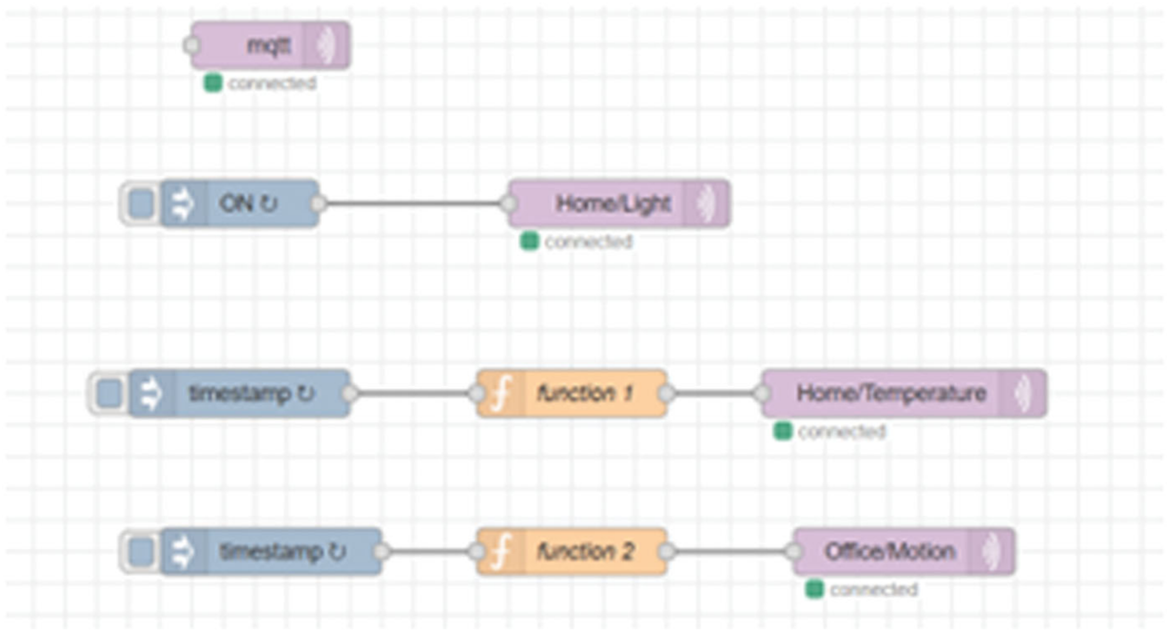


Figure 5.2 Virtual IoT devices on Node-RED.⏎

- Suricata IDS: It is deployed on a dedicated EC2 instance in order to monitor the real-time traffic associated with the network. This not only captures and analyses the malicious and normal traffic but also generates the logs and stores them in JSON format for further processing.
- Traffic Mirroring: While the IoT server and IDS instances were deployed within the same VPC, additional configurations were required to set up a clear connection between these in order to ensure that the IDS captures the network traffic from these sources and thus traffic mirroring was configured to establish and maintain this connection.
- Log Pipeline: For a smooth and seamless transfer of data, filebeat was deployed and configured on the Suricata instance. It is used to ship the logs from IDS to Logstash instance directly in real time. The Logstash then processes and structures the data before sending and storing it in the Elasticsearch. This flow makes sure an efficient way to handle the data transmission.
- ELK stack SIEM: This is again deployed on a separate instance which is responsible for functioning of Elasticsearch, Logstash and Kibana. The main functionality of the SIEM here is to maintain the logs centrally which are received from the IDS, visualising the alerts triggered through the rule’s setup in the Kibana and display it on the dashboards. It not just inputs the data from IDS but also from the ML model instance further displaying its predictions.
- Trained models (EC2 instance): A separate and dedicated instance was deployed in order to store the trained models wherein it can fetch the

logs from Elasticsearch, classify them according to the trained models and index the predictions back to the Elasticsearch.

- Attacking Machine: While the above setup displays the detection capabilities, this instance was deployed in order to generate the network traffic through the simulation attacks. This is deployed on an isolated instance wherein a docker container is installed and used to simulate the DDoS attack using the hping3 command.

5.3.5 Detection Mechanism

The detection mechanism used in this research uses a dual strategy wherein the detection relies on both rule-based detection using the SIEM capabilities and on the predictions made through the ML models. This combined approach complements each other and strongly puts together a robust framework for identifying security threats in real time.

Rule-based Detection – The ELK stack SIEM has a vast dictionary of pre-configured rules to detect a wide range of known threats and attacks. These rules were then imported in our system from SIEM’s default library. This helped in identifying the attack patterns such as high-volume traffic, unusual port usage, suspicious HTTP requests, etc. In order to enhance the system’s detection capabilities custom rule was also created in Kibana to identify and flag the attack patterns observed in the logs through the simulated scenarios focusing on DDoS wherein the rule [18] was customised to trigger the alerts after identifying high traffic anomalies, such as SYN flood attack targeting the IoT server. These rules helped in generating real-time alerts on Kibana which was then helpful in visualising the dashboards that enable efficient real-time monitoring of network activities.

ML-based Detection – The ML-models RF and XGBoost were deployed and trained based on the preprocessing steps mentioned in the earlier section. They were integrated in order to adapt the threat detection by analysing the network traffic logs and classifying the events as DDoS and benign. Here a python-based pipeline was designed and structured in way that it fetches the logs from Elasticsearch and passes them to the model for classification, and these predictions made by the models were then indexed back to Elasticsearch, ensuring an absolute integration with the SIEM system. This integration of ML-based detection thus strengthened the system’s detection capabilities as it detects the attack patterns which get evaded from the static rules enhancing the overall detection capabilities.

This combined approach can be viewed together at the same time on Kibana dashboard, making it effective to address the challenges posed by the attackers through attacks such as DDoS in real time.

5.4 Design Specification

5.4.1 Introduction to System Design

The main objective of this research was to structure a robust IoT design framed in a cloud-based environment having several virtual IoT devices which could be possibly targeted and thus create a real-time monitoring structure which would be able to capture, detect and classify these attacks. The architecture of the entire setup was precisely crafted to imitate a realistic IoT environment including multiple components that would seamlessly interact with each other in order to monitor and analyse the network traffic. The primary components integrated here are the IoT server which hosts the IoT devices using technologies such as Node-RED and MQTT broker, an IDS which would monitor the network traffic, a SIEM system which would centrally log and visualise the traffic and, finally, the ML models which would enhance the threat detection capabilities of the system through its adaptiveness.

The design was built by prioritising the scalability and adaptability of the solution, which would allow the system to handle real-time traffic flows and support multiple detection strategy. The setup was implemented completely on AWS due to its enhanced flexibility and great computational power resources which was necessary for simulating the real-world attack scenarios and analysing the network traffic activity in real time without any delays.

5.4.2 Architectural Overview

Figure 5.3 shows the architecture of the designed system consisting mainly of five AWS EC2 instances which are responsible for handling the function of IoT server, Suricata IDS, ELK stack SIEM, trained ML models (RF and XGBoost) and the attacking machine. All these instances had to be interconnected for a seamless interaction between them, which would enable real-time threat detection. Executing the solution on cloud also enabled the system to be completely isolated and controlled for the experimental purpose. The IoT server which hosts the virtual IoT devices designed on Node-RED and communicating with MQTT broker acts as the primary target for the DDoS attack, as it represents to be an integral part of any IoT setup. The attacking machine instance was used to simulate the DDoS attack via a docker container which would generate the SYN flood and UDP flood attack using the hping3 command. Traffic mirroring was done on AWS by setting up mirror session mainly between the IoT server and IDS. This approach ensured that all the network traffic would be routed towards the IDS, enabling complete monitoring of the resource. The Suricata IDS was then responsible for analysing this captured traffic and generating detailed logs for all the network activity. These logs were then passed to the SIEM server with the help of filebeat (a lightweight log shipper), where the logs were analysed, indexed and presented in real-time dashboards. The ML models were deployed and run on an EC2 instance which would link them to the SIEM instance for detection. This connection helped in SIEM sending the logs to the instance where the models could classify the events as DDoS and benign and send the predictions back to the SIEM for its visual representation along with the rule-based detections. Within Elasticsearch, two distinct indexes are maintained: one for rule-based detections generated by Suricata and another for logs that are analysed by ML models. These indexes help in splitting the results from both the sources, ensuring a comprehensive analysis. Lastly, Kibana is used as the visualisation layer to provide more engaging dashboards for the outcome of rule-based and ML-based detections. Being the central component for generating system alerts, the dashboards grant administrators visibility into system activity and expose possible attacks, their trends, and categorisation, which is useful for managing network activity.

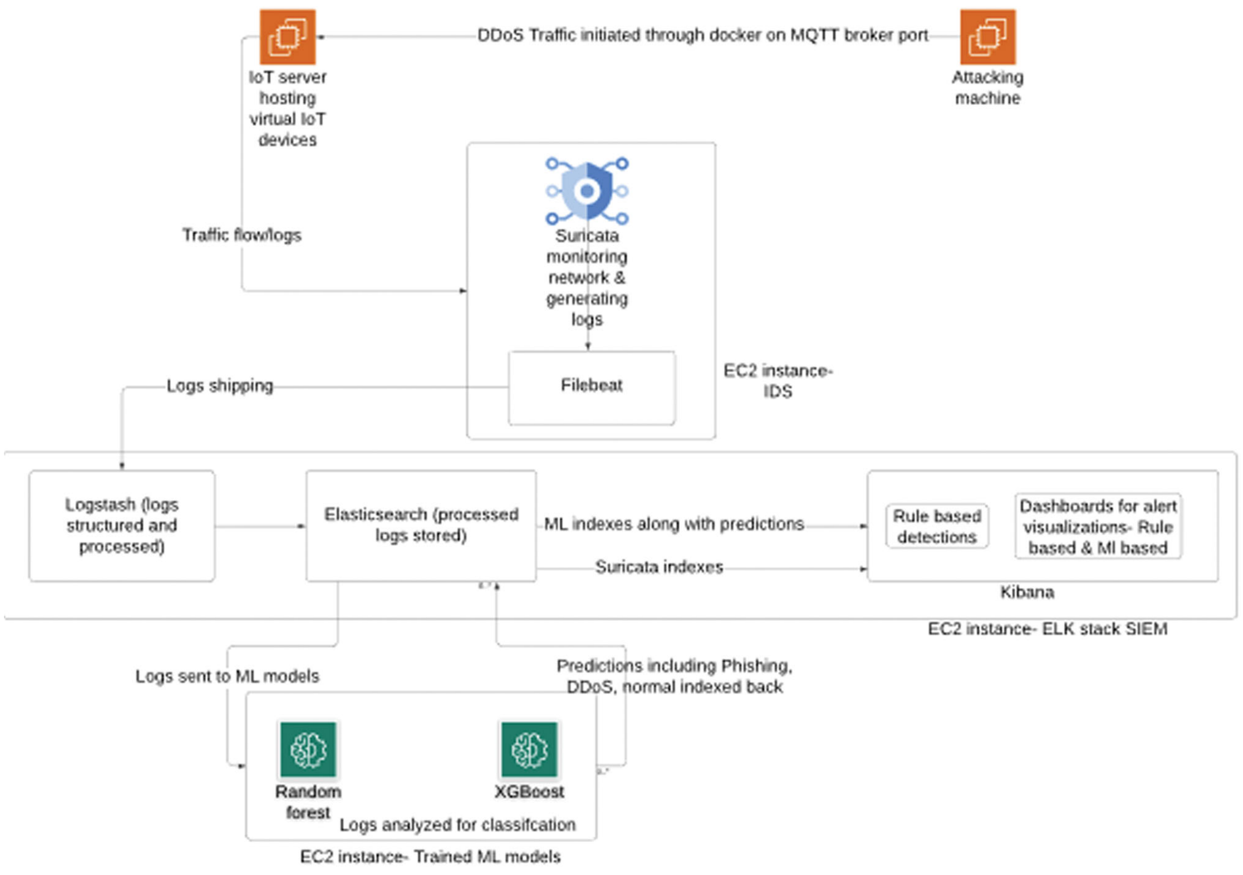

DDoS Traffic initiated through docker on MQTT broker port
IoT server hosting virtual IoT devices
Attacking machine
Traffic flow/logs
Suricata monitoring network & generating logs
Filebeat
EC2 instance- IDS
Logs shipping
Logstash (logs structured and processed)
Elasticsearch (processed logs stored)
ML indexes along with predictions
Suricata indexes
Rule based detections
Dashboards for alert visualizations- Rule based & Ml based
Kibana
EC2 instance- ELK stack SIEM
Logs sent to ML models
Predictions including Phishing, DDoS, normal indexed back
Random forest
XGBoost
Logs analyzed for classifcation
EC2 instance- Trained ML models

Figure 5.3 System architecture.⏎

5.4.3 Justification for Selected Tools and Technologies

All the tools and technologies selected were chosen to align with the current needs of the cybersecurity world, ensuring that the research's setup would imitate the practical use cases while maintaining the academic relevance as well.

- Suricata (IDS): This IDS being an open-source and highly customizable system emerges to have prominent ability to perform a real-time network traffic analysis. It generates detailed logs of the captured network activity which is a primary need of the research. It also supports emerging threat rule sets allowing real-time detection of not only DDoS and attack patterns but also hundreds of other attacks. Due to all these benefits, especially its ability to handle the high-volume traffic efficiently, its native multi-threading techniques made it a better choice than any other open-source IDS (viz. Snort IDS) [17] (Waleed, Jamali and Masood, 2022).
- ELK [19] stack SIEM: This again is an open-source tool with Elasticsearch which can efficiently store and index large volumes of data/logs for real-time analysis, Logstash which can process and structure the logs ensuring that the data is suitable for advanced querying in the Elasticsearch and, lastly, Kibana which provides a user-friendly graphical user interface for interpreting what is actually going on in the SIEM by visually representing data via dashboards. Due to all the necessary requirements being fulfilled by using stack and given its popularity in real world wherein it is used by multinational giants, this tool was highly relevant for the overall system.
- ML models (RF and XGBoost): Both being ensemble learning models have improved classification rate and can reduce overfitting issues notably. While RF is a robust ML model usually used to handle large datasets, XGBoost provides optimum performance with imbalanced datasets, making both an ideal selection for adaptive detection in real-time environment considering their speed and effectiveness.
- Node-RED: Being user-friendly with flow-based interface, Node-RED was chosen to simulate the IoT devices because it aids in quick prototyping and visualising the IoT data flows. It gives a simple platform to create various IoT devices which can replicate the functions of real devices such as temperature sensors, motion detectors and humidity monitors and also save them to the MQTT broker easily.
- MQTT Broker: The primary advantage of MQTT broker is its lightweight protocol that is specifically designed for artificial devices and low-bandwidth high-latency environment; therefore, it was chosen as a central hub for communication of IoT. Unlike HTTP-based communication, MQTT offers a better messaging with a lower overhead that makes it suitable for IoT use cases where devices are flooded with real-time data. In addition, its publish-subscribe model makes it possible to flow the

communication open among various IoT devices and the server, improving the scalability and reliability. The usage of MQTT broker helped the simulated system closely align with the realistic IoT environment.

- AWS EC2 instances: As the above-mentioned tools and technologies require heavy computational power and storage to process and function to the best of their requirements, it was very important to provide them the base they need for the setup. This was offered by EC2 instances with its immense scalability and flexibility for deploying multiple components in one go. It also helped in assigning the elastic IPs to our systems.
- hping3: This is a highly flexible command line tool which can generate custom packets with varying attack intensities to simulate flood attacks (Ariffin et al., 2021) [18].

5.4.4 Design Considerations and Constraints

The balance between realism, scalability and adaptability was the core while designing the entire system. Scalability was highly prioritised as the system had to deal with large volume of traffic which would be generated during the simulated DDoS attack; this traffic was also to be processed for giving the results in real time. Thus, the use of AWS EC2 instances became crucial as it provided the flexibility to deploy components with different computational needs, such as lightweight instances for attacking server and other machines and high-performance and high-storage instances for IDS and SIEM. As this setup was entirely run in an AWS environment, it ensured that the simulated attacks are not creating any issues for the external system. Docker container was deployed on the attacking machine to set apart the traffic sources of two distinguished attacks. This enabled control over traffic generation without any issues, such as cross-contamination of logs. The next big thing to be constituted was realism as the entire setup had to be replicated as a real-world attack scenario. The usage of MQTT broker resembles like an actual communication happening between the IoT devices with its central hub. Also, as the system encompasses the rule sets from emerging threat point on Suricata, the detection capabilities of the system in terming the DDoS attacks were enhanced and well equipped with industry standards.

While all this was to be done, several challenges arose during the implementation, which were to be addressed and overcome for making a resilient system. Initially, AWS lambda and AWS Sagemaker were implemented for the deployment of ML models. But due to configuration and connectivity issues and also because of problems raised by IAM constraints, this approach was not feasible and thus was replaced by the adoption of EC2 instances wherein these models are deployed. Moreover, with ML models another big issue faced in the early stage was that the models were trained on sample datasets initially but the raw logs which were generated from the IDS were nothing similar to the sample datasets available in various resources, and this created a massive feature mismatch causing the retraining of models from the datasets acquired from actual raw JSON logs from the experimental setup. Through this alignment between the dataset and real-time predictions, the results received were unparalleled. Connectivity issues between filebeat, Logstash and Elasticsearch were addressed through repeated trial and testing to ensure that the log transmission pipeline functions accurately. The resource limitations of EC2 instances in handling the heavy tools with huge volume of data were sorted out by promoting/upgrading the instance types and finding the accurate one which could balance the load for the system. These considerations and constraints overall framed the architecture of the entire system, resulting in a robust and reliable design.

5.5 Implementation

5.5.1 Infrastructure Setup

Table 5.3 shows the EC2 instances with their configured roles and resource allocations.

Table 5.3 EC2 Instances⏎

Component	*Instance Name*	*Elastic IP*	*Instance Type*	*Role*	*Configuration Details*
IoT server	IoT devices	54.81.38.135	t3.micro	Hosting the IoT devices	Node-RED and MQTT broker were used to simulate IoT devices.
IDS server	IDS server	34.192.113.33	t2.micro	Monitoring the network traffic	Configured Suricata with the emerging threat point rules. Deployed filebeat and connected with SIEM.
SIEM server	SIEM server	35.172.58.170	t3a.large	Hosting Elasticsearch, Logstash and Kibana	Logstash receiving data. Elasticsearch

Component	*Instance Name*	*Elastic IP*	*Instance Type*	*Role*	*Configuration Details*
					(port:9200), Kibana (port:5601).
Trained models server	ML models	52.3.50.159	t3.micro	Running trained models	Connected with SIEM instance to fetch logs and push predictions.
Attacking machine	Attacking machine	3.226.63.94	t2.micro	Simulating the DDoS attack	DDoS via hping3 through docker.

5.5.2 Detailed Configurations

- Suricata Configuration: Suricata as an IDS was deployed on a dedicated t2.micro instance running on Ubuntu with a storage space of 17 GB. All the primary configuration was modified in the customizable suricata.yml file. This instance was also set as the mirror target to have a clear mirror session from the IoT server hosting the IoT devices. The IPs of these servers were also added in the Home Net [20] section of Suricata configuration file so that it is able to capture the network traffic smoothly. The rule sets were then imported from Emerging Threat point in the IDS through suricata-update command to ensure that the detection capabilities of the IDS are well in place. This included critical rules covering several attack patterns, especially the one identifying DDoS attacks in case of SYN flood and UDP flood scenarios [21]. All these rules were imported and activated on the system.
- Filebeat configuration and integration: Filebeat was deployed and configured on the IDS [22] server due to its lightweight approach wherein it forwards the eve.json logs to the Logstash in the SIEM [19] server. This was done primarily by enabling the Suricata module on filebeat through (sudo filebeat modules enable Suricata). Through this module the Suricata logs were automatically parsed and structured, eliminating any need for custom input configurations in the filebeat.yml file. However, the output configuration had to be set in the filebeat.yml wherein it can target the Logstash server on its IP 35.172.58.170 which listens the data on port 5044. This allowed the filebeat to forward the structured logs.

- Logstash Configuration and Integration: Logstash plays an important role in the entire SIEM server as it listens to the raw logs sent from the filebeat and transmits it to the Elasticsearch. All this communication is defined in the filebeat-suricata.conf file located in Logstash. The configuration is mainly divided in three parts input, filter and output for handling the Suricata logs effectively. The input section depicts the listening of Json logs on port 5044, aligning with the output section in filebeat conf. Next, in the filter section, it identifies and processes the events of alert type. The output section of the file is defined to send the logs to Elasticsearch which is hosted on the same SIEM server on port 9200. These logs are indexed in a separate index pattern naming, suricata-logs-*. By specifying the formatting as index => "suricata-logs-%{+YYYY.MM.dd} the file also ensures that a separate index file is created on each day with different logs. This configuration is validated through immense testing of pipeline using testing commands and thorough inspection of whether the real-time logs are indexed in Elasticsearch.
- Elasticsearch and Kibana configuration: Elasticsearch is deployed as a single–node -cluster. It is configured to listen all the network interfaces (0.0.0.0) on port 9200 so that the connections beyond the localhost can also be accepted. This was mainly essential for forming a seamless connection with other SIEM components, that is, Logstash and Kibana. The xpack.security module was added to support the authentication and secure access through API keys. On the other hand, Kibana has also been configured to bind all the network interfaces as the server.host is set to 0.0.0.0 on port 5601. This was done for accessibility from any system, within the network and interaction with Elasticsearch. Although Elasticsearch and Kibana are configured to an initial localhost for development usage, these parameters were changed to fit the distributed cloud structure. Kibana accesses the Elasticsearch through its own credentials which are set in its conf file (kibana_system user and its passwords). Among all the default elastic rules, a total of 686 are updated in the Kibana along with the custom rules. Rule for detecting DDoS was defined in Kibana, which uses suricata-logs-* index and fields such as event_type, alert_signature and hostname. The DDoS detections are relying on SYN flood signatures tagged as Attempted Denial of Service attack patterns. A specific connector was put in place for the integration of this rule which made sure the alerts generated are stored in a particular index.
- ML integration:- Here the two .pkl files for the selected ensemble learning models which are pre-processed as per the instructions stated in Section 5.4 are hosted in a t3.micro server. These models are deployed to

analyse the logs in real time from suricata-logs-* index located at Elasticsearch. A connection to Elasticsearch is established here using the Representational State Transfer (REST) Application Programming Interface (API), using the SIEM server IP which is 35.172.58.170. Also it has been given the authentication credentials for a secure data connection. As the system is designed to work continuously with real-time data, the @timestamp field is used to fetch only the logs that are new or updated since the last processed batch. This dynamic strategy helped the models to deal with fresh data every time. The next step is the preprocessing pipeline which is applied to extract the relevant features from the fetched logs to match the feature requirements of the model for effective classification. Logs with missing values are filtered out during this stage to ensure that the model works on complete data. Both these models generate probability scores for each log entry for marking the classification of attacks. These scores are then averaged so that the strengths of these models are combined and a classification threshold is put in place, marking the entries with their respective scores. Logs with a score below 0.3 were tagged as DDoS, while those above this score are tagged as benign activities. These thresholds were chosen due to the nature of attack. As in the DDoS attack, it heavily relies on huge volume of traffic with repetitive patterns, such as SYN flood attempt, resulting in a generally lower score from the model. Thus, logs with a score less than 0.3 are tagged as DDoS. To minimise false positive rates, the logs which lack the distinguished patterns for the attacks are classified as benign. The processed logs which now contain the prediction classifications are indexed back to Elasticsearch in a new index pattern as ml-predictions-YYYY.MM.DD* using bulk API operations. The date-based naming system was introduced for a better organisation of logs and predictions. The system continuously processes the available logs in a continuous loop every 5 seconds, and if the system encounters any connectivity issues, it gracefully handles it through retry mechanism. Thus, this approach offers optimised data ingestion and scalability without affecting real-time prediction efficiency.

- Attack simulation setup: The attacking machine setup was implemented on an AWS EC2 instance (t3.micro) with a Docker container running the hping3 tool. This tool was used in targeting the IoT server’s port 1883 using the hping3 -S -p 1883 –flood 54.81.38.135 command. This command is executed from the docker container 4e1aabe808e3 to isolate the attack traffic. This command generates a huge number of SYN packets at a very high rate. The attack was directed at the MQTT broker located on the IoT server by attacking the port assigned to it, and using SYN flood to send as many TCP SYN packets as possible to the target.

This approach makes the simulation look like a practical scenario where IoT communication protocols could be exploited. Thus, Suricata receives all these logs as the traffic is mirrored from the employee machine and the whole detection process is initiated.

5.6 Evaluation

This section would evaluate the performance and functionality of the implemented IoT real-time security system through mainly two aspects: system performance (depicting the efficiency and stability of the detection mechanism) and detection accuracy (showing the effectiveness of the rule-based and ML-based approaches in identifying the DDoS attack). The results are mainly analysed against metrics such as detection times, system reliability, and classification performance aligning with the objectives of the research question.

5.6.1 System Performance

Figure 5.4 illustrates the detection times for individual events, showing fluctuations in detection time mostly between 0.2 and 1.0 ms. The detection mechanism remains consistent throughout the events without major spikes and delays. The overall detection time remains within a suitable sub-millisecond range, which depicts that the system is able to rapidly process the events. Also, even after the variability in detection times the upper limit of a maximum 1.0 ms reflects the real-time responsiveness. The system's ability to handle large volume of data consistently maintaining minimal detection timeline aligns well with the research's goal of real-time monitoring. This consistent performance depicted in the result increases the reliability of the system.

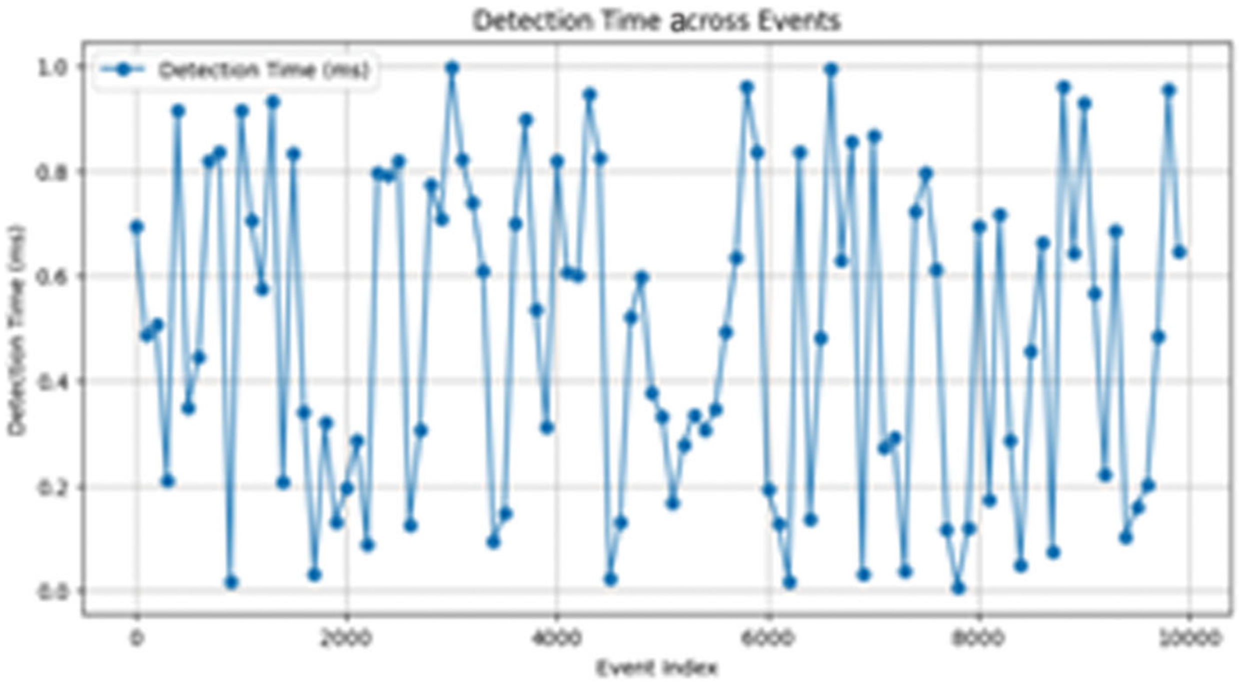


Figure 5.4 Detection time across events.⏎

Figure 5.5 illustrates the frequency distribution of detection times, which ranges between a minimum time of 0.0 ms and a maximum time of 0.999 ms, with the average detection time being 0.514 ms. The significant finding from this is that the efficiency of the monitoring system with an average detection time of 0.514 ms highlights the system’s ability to detect events instantly. The low detection scores for a dataset of 10,000 events also depict the scalability of the system, reflecting that it can handle larger workloads efficiently. This evaluation depicts the robustness of the system’s detection mechanism. Even when the events also contain high-density traffic flows such as during the DDoS simulation, the system can still handle this traffic without any performance degradation.

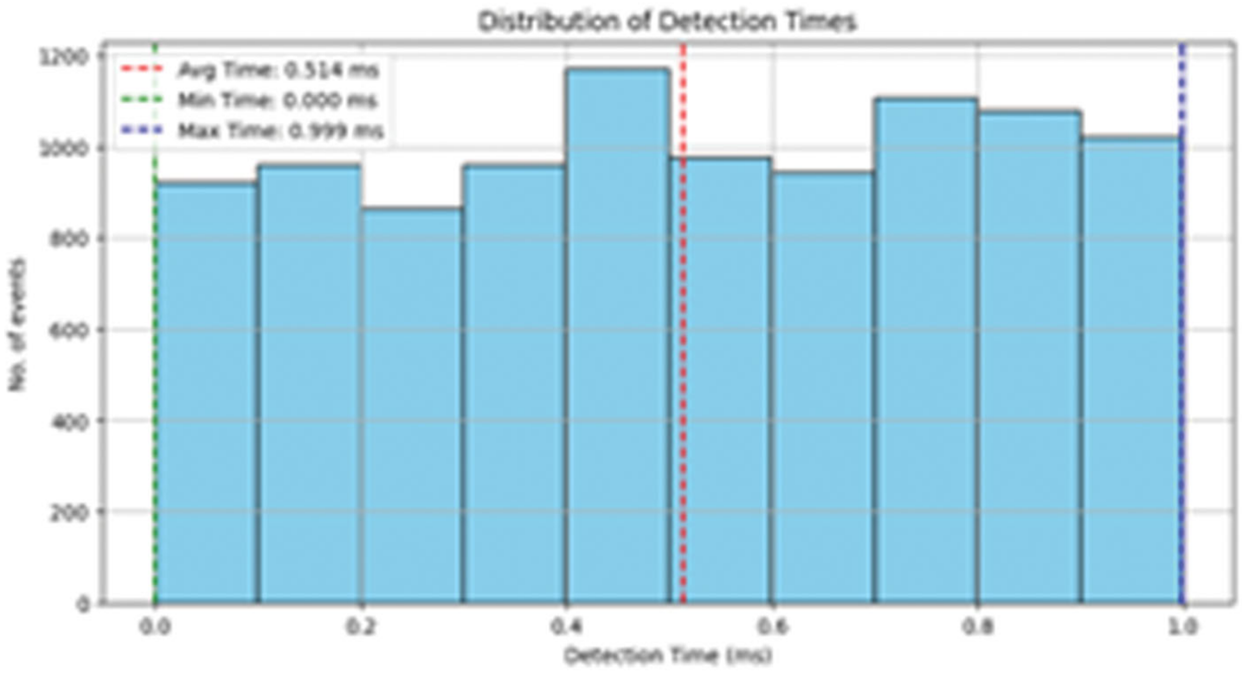


Figure 5.5 Distribution of detection times.⏎

5.6.2 Classification Reports for the ML Models

RF – The classification report for RF is presented in Table 5.4.

Table 5.4 RF Classification Report⏎

Metric	*Precision*	*Recall*	*F1-Score*	*Support*
Benign (0)	0.97	0.99	0.98	3760
Attack (1)	0.96	0.94	0.95	6240
Accuracy			0.96	10000
Macro Avg	0.97	0.97	0.97	10000
Weighted Avg	0.97	0.96	0.96	10000

Precision: The Random Forest model has high precision, especially for benign (0), meaning it rarely predicts benign when it is actually an attack.

Recall: It captures Benign (0) well (99%) but slightly misses some attacks (94% recall for Attack (1)). This model is sensitive to benign cases, ensuring that normal traffic is rarely flagged as an attack.

XGBoost – The classification report for XGBoost is presented in Table 5.5.

Table 5.5 XGBoost Classification Report⏎

Metric	*Precision*	*Recall*	*F1-Score*	*Support*
Benign (0)	0.98	0.98	0.98	3760
Attack (1)	0.95	0.96	0.95	6240

Metric	*Precision*	*Recall*	*F1-Score*	*Support*
Accuracy			0.97	10000
Macro Avg	0.96	0.97	0.96	10000
Weighted Avg	0.97	0.97	0.97	10000

Precision: XGBoost prioritises precision for both Benign (0) and Attack (1). It’s slightly better at identifying true attacks with minimal false positives.

Recall: It maintains balanced recall for both classes, ensuring both attacks and benign traffic are accurately flagged.

Key differences between the models are shown in Table 5.6.

Table 5.6 Key Differences between Models⏎

Feature	*Random Forest*	*XGBoost*
Strengths	Sensitive to benign scenarios (low false positives for normal traffic).	Good balance between the attack and benign detections.
Overall accuracy	96%	97%

5.6.3 Distribution of Detected Events (ML Predictions)

The pie chart in Figure 5.6 describes the distribution of detected events on the basis of ML-based predictions. These predictions are derived from separate datasets from a different timeline, making sure the accuracy is maintained even with dynamic input data reflecting real-time log analysis, which differs from the static dataset used for training the models. The graph shows the accurate findings of the ML models wherein the DDoS accounts for 37.5% of the total logs analysed, given the huge traffic DDoS attacks create, while 62.5% is classified as benign. This visual presentation shows the ability of the ML model by demonstrating how it can differentiate between the malicious traffic and normal traffic in the simulated IoT network. The use of both benign and attack traffic makes the dataset more balanced, thus preventing over-optimistic results. This balance validates the robustness of the model, confirming its capability to accurately detect DDoS attacks while handling a considerable proportion of benign traffic. The graph also highlights the adaptable capabilities of the models across different datasets (real-time logs), which was the core fundamental of the research question. The integration of ML-based predictions with SIEM demonstrates real-world application of the selected models, hence proving the research objective of creating a real-world solution for security monitoring.

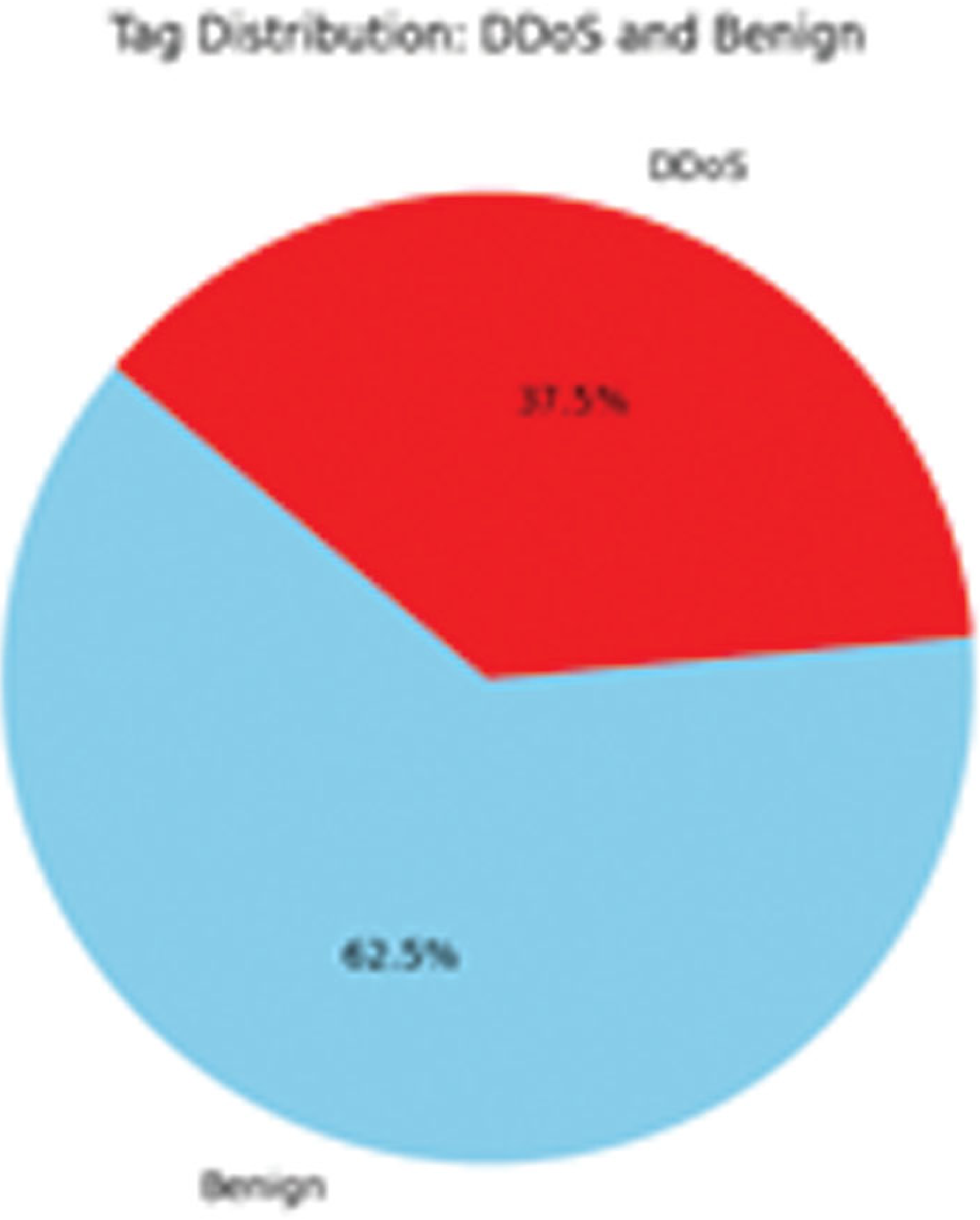


Figure 5.6 Tag distribution: DDoS and benign.⏎

5.6.4 Detection over Time (Rule-Based Kibana Visualisation)

The graph in Figure 5.7 displays the identification of DDoS attacks by time in rule-based mechanisms of Kibana. The grey line is drawn for alerts that have been classified as "Potential DDoS Attack – SYN Flood on MQTT Port 1883." This graph represents the identification of some particular time intervals which are associated with the DDoS attacks simulated. The lack of regular activity between these spikes confirms that at times there is no malicious traffic at all, highlighting the strength of the rule-based system in identifying the occurrence of attack-specific events.

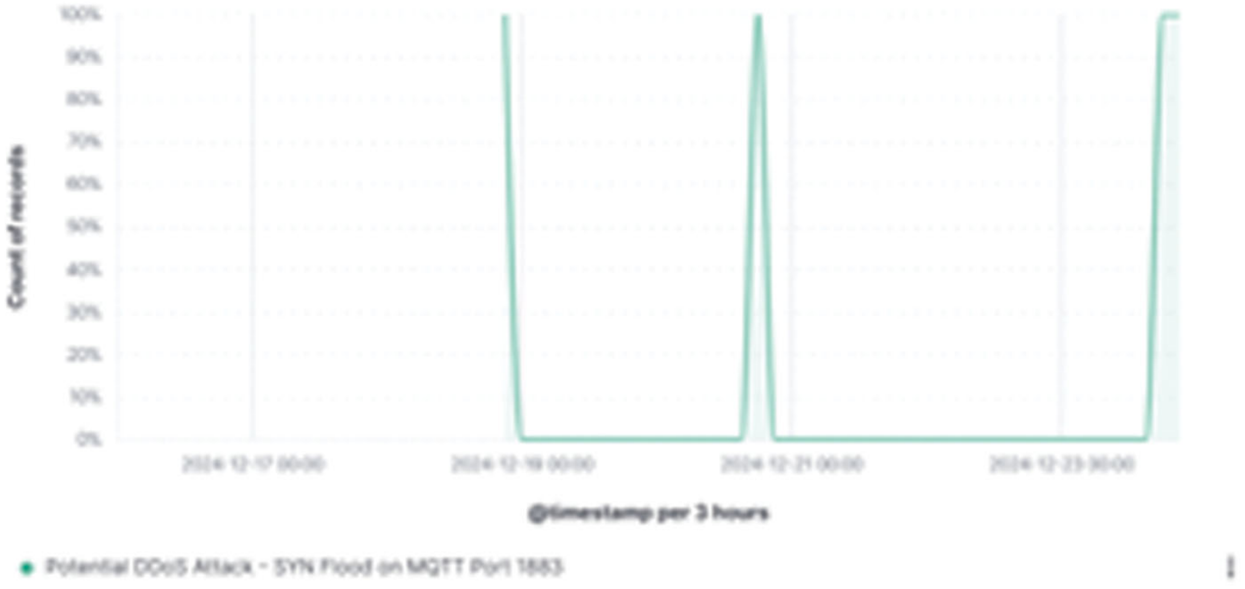


Figure 5.7 Rule-based detection.⏎

The graph in Figure 5.8 categorises network traffic into two main types: “flow” (grey) and “other” (black). The flow comprises the legitimate or non-attack traffic captured in the simulation, while “other” refers to other activities that are not triggered through an alert. This visualisation gives a general picture of the network activity, regular and abnormal traffic. The presence of “flow” traffic between peaks of “other” guarantees a more elaborate assessment of the system’s performance in detecting and tracking events of any kind except malicious ones.

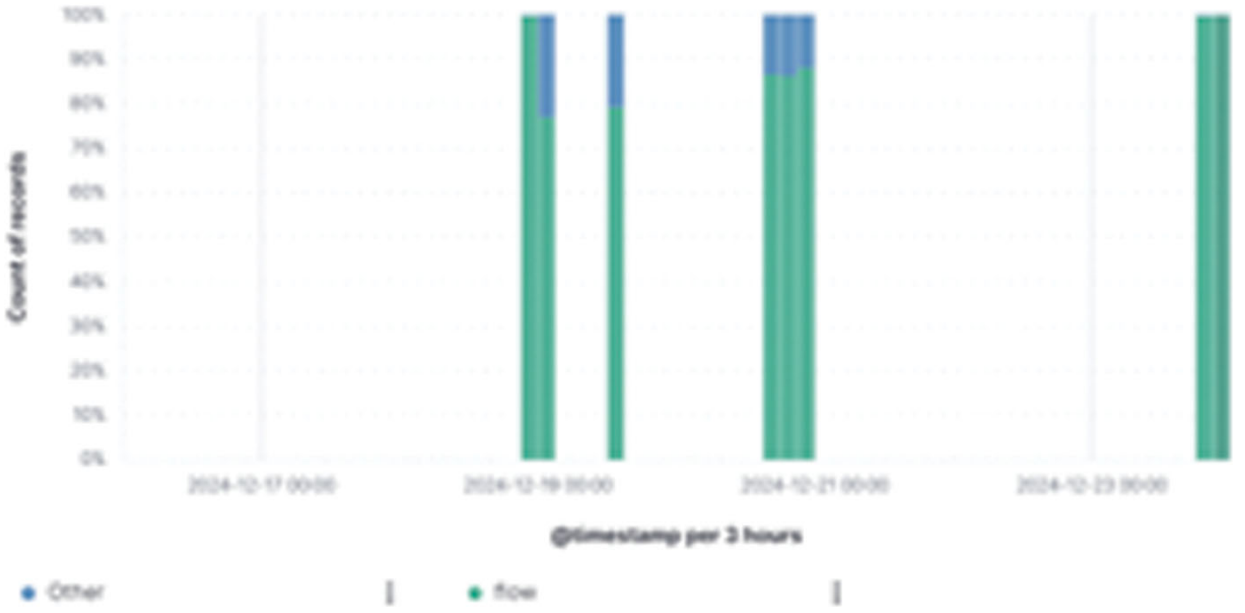


Figure 5.8 General traffic.⏎

5.6.5 Discussion

The above findings are critical in understanding the valuable insights which the hybrid detection system has to offer with the integration of SIEM, IDS and ML models in identifying the DDoS attack. This discussion will critically evaluate these results, highlighting the system's strengths and weaknesses in comparison with existing research and providing suggestions for future growth.

The system's performance, which is measured through detection time analysis, has an average detection time of 0.514 ms and a maximum of 0.999 ms, indicating its suitability in real-world applications. This shows the efficiency of the system in handling and processing the events as swiftly as required in a dynamic environment. However, there are certain spikes in the detection time across some events, which possibly indicate occasional computational delays that might be caused due to resource contention or overheads in data preprocessing. For maintaining consistent real-time performance there is a space for further optimisation in terms of resource allocation and multi-threading capabilities. Both the classification reports for the ensemble learning models demonstrate high accuracy in terms of detection, with nearly perfect detection classification for the DDoS category. But it has been noticed that XGBoost slightly outperforms RF in terms of recall and precision as XGBoost is known to work well with imbalanced datasets. The support values in the classification report reveal the number of samples in each category – benign and attack – highlighting the effectiveness of the system during high traffic during DDoS simulations. Moreover, the rule-based system effectively identifies the spikes in activities showcasing the system's real-time ability in capturing the flooding of SYN events, validating the system's robustness in dealing with high volumetric attacks. The high percentage of attack traffic proves the ability of the models and the system to detect and classify the volumetric DDoS attacks successfully.

With the context of prior research conducted in this field, this research demonstrates a practical implementation of a hybrid detection mechanism in a cloud environment. Unlike prior studies which mostly rely on conceptual frameworks, this research works under real-time conditions, showcasing the use of XGBoost and RF effectively in security applications such as SIEM and IDS and thus providing hands-on evidence of the system's applicability in addressing the gaps like scalability, adaptability and real-time performance.

5.7 Conclusion and Future Work

This research initially addressed critical challenges of developing a real-time hybrid security solution designed in cloud-based environment focusing primarily on DDoS attack. The research question was stated to discover how the SIEM, IDS and ML models can be integrated effectively to achieve a scalable and adaptive threat detection system for securing the IoT systems. The objective of the research was to design and evaluate a framework under simulated attacks replicating the real-world scenarios. Through the developments done in this research the implemented framework has successfully demonstrated its ability in detecting threats in real time by combining the strengths of rule-based and data-driven (ML-models) approaches. Key matrix in the evaluation section including detection time and classification accuracies validates the system's efficiency in addressing real-time threats.

This research work proves that the integration of SIEM, IDS and ML model technologies can be vital in developing a stronger and enhanced framework in order to combat the current threats posed in a dynamic IoT environment. The hybrid detection system scored an average detection time of 0.514 ms, depicting the robust real-time detection abilities. The system also underlines the potential strengths of combining rule-based and ML-based detections rather than being used as isolated solutions in a security architecture by visualising the detections made by both rules and ML predictions on Kibana. The system's real-time detection ability highlights its applicability in real-world environments. Through this the system has achieved its objectives of designing, implementing and evaluating a scalable hybrid detection system. The findings of this research are important to both academic researchers and real-world practitioners as they provide a link between theoretical and practical applications. Academically, it provides substantial evidence in integrating ML techniques with SIEM and IDS focusing on the practical approach, and for industries, the framework puts forward a robust system by enhancing real-time monitoring in cloud-based environment by an adaptive approach for threat detection. While the research exhibits the potential of the hybrid system, it also displays several limitations associated with the research. Attack scenarios were performed in a controlled environment which at times fails to capture all the possible variations of an attack that can occur in the actual environment.

5.7.1 Future Works

- Real-World Integration: Although our system is designed to work under real-world conditions, it is only made possible through simulations and thus it is critical in validating the system with real-world network traffic logs to capture the unpredictability and diversity in the data. This can be possible through partnering with organisations so that the system could get high volume of real-world logs.
- Adaptive Learning Models: Future developments of the system can be achieved by integrating self-adaptive models which will dynamically update based on different network activities or attack patterns. This will make the system to be more reliable in identifying the zero-day attacks.
- Commercialisation Potential: The proposed framework is robust enough to be deployed in a small and medium-sized enterprise by working on user-friendly interface and packaging the system for commercial use as it can turn out to be a reliable, scalable as well as cost-effective detection strategy for them.

References

1. Morkos, R. (2023) *Powering the Growth of Cloud Computing: Infrastructure Challenges and Solutions*. Available at: www.forbes.com/councils/forbestechcouncil/2023/07/24/powering-the-growth-of-cloud-computing-infrastructure-challenges-and-solutions/ [Accessed 25 November 2024].⏎
2. Ahmadi, S. (2024) ‘Network Intrusion Detection in Cloud Environments: A Comparative Analysis of Approaches’, *International Journal of Advanced Computer Science and Applications (IJACSA)*, 15(3), 2024. http://dx.doi.org/10.14569/IJACSA.2024.0150301⏎
3. Saeed, M. S., Saurabh, R., Bhasme, S. R. and Nazarov, A. N. (2022) ‘Machine Learning Based Intrusion Detection System in Cloud Environment’, in *2022 VIII International Conference on Information Technology and Nanotechnology (ITNT)*. Samara, Russian Federation, 23–27 May 2022, pp. 1–6. https://doi.org/10.1109/ITNT55410.2022.9848611⏎

4. NCCS (2024) *Traffic Generator/DDoS Tool*. Available at: https://nccs.gov.in/public/events/DDoS%20Presentation%2017092024.pdf [Accessed 15 October 2024].⏎

5. Oracle (2023) *Design Guidance for SIEM Integration*. Available at: https://docs.oracle.com/en-us/iaas/Content/cloud-adoption-framework/siem-integration.htm?u [Accessed 27 November 2024].⏎

6. Proofpoint (2024) *Community Alert: Ongoing Malicious Campaign Impacting Microsoft Azure Cloud Environments.* Available at: www.proofpoint.com/us/blog/cloud-security/community-alert-ongoing-malicious-campaign-impacting-azure-cloud-environments [Accessed 25 November 2024].⏎

7. Tuyishime, E., Balan, T. C., Cotfas, P. A., Cotfas, D. T., and Rekeraho, A. (2023) 'Enhancing Cloud Security—Proactive Threat Monitoring and Detection Using a SIEM-Based Approach', *Applied Sciences,* 13(22), 12359. https://doi.org/10.3390/app132212359⏎

8. González-Granadillo, G., González-Zarzosa, S. and Diaz, R. (2021) 'Security Information and Event Management (SIEM): Analysis, Trends, and Usage in Critical Infrastructures', *Sensors,* 21(14), 4759. https://doi.org/10.3390/s21144759⏎

9. Lee, J-H., Kim, Y. S., Kim, J. H. and Kim, I. K. (2017) 'Toward the SIEM architecture for cloud-based security services', in *2017 IEEE Conference on Communications and Network Security (CNS)*. Las Vegas, NV, USA, 09–11 October 2017, pp. 398–399. https://doi.org/10.1109/CNS.2017.8228696⏎

10. Ayu, M. A., Erlangga, D., Mantoro, T. and Handayani, D. (2024) 'Enhancing Security Information and Event Management (SIEM) by Incorporating Machine Learning for Cyber Attack Detection', in *2023 IEEE 9th International Conference on Computing, Engineering and Design (ICCED).* Kuala Lumpur, Malaysia, 07–08 November 2023, IEEE Xplore. https://doi.org/10.1109/ICCED60214.2023.10425288⏎

11. Innab, N., Atoum, I., Alghayadh, F., Abu-Zanona, M., Alrubayyi, N. and Basudan, F. (2024) 'Intrusion Detection System Mechanisms in Cloud Computing: Techniques and Opportunities, in *2024 2nd International Conference on Cyber Resilience (ICCR).* Dubai, United Arab Emirates, 26–28 February 2024, pp. 1–5. https://doi.org/10.1109/ICCR61006.2024.10532903⏎

12. Munshi, A., Alqarni, N. A. and Abdullah Almalki, N. (2020) 'DDOS Attack on IOT Devices', in *2020 3rd International Conference on Computer Applications & Information Security (ICCAIS).* Riyadh, Saudi Arabia, 19–21 March 2020, pp. 1–5. https://doi.org/10.1109/ICCAIS48893.2020.9096818⏎

13. Çakmakçı, S. D., Hutschenreuter, H., Maeder, C. and Kemmerich, T. (2021) 'A Framework for Intelligent DDoS Attack Detection and Response using SIEM and Ontology', in *2021 IEEE International Conference on*

Communications Workshops (ICC Workshops). Montreal, QC, Canada, 14–23 June 2021, pp. 1–6. https://doi.org/10.1109/ICCWorkshops50388.2021.9473869⏎

14.Dhahir, Z. S. (2024) “A Hybrid Approach for Efficient DDoS Detection in Network Traffic Using CBLOF-Based Feature Engineering and XGBoost”, *Journal of Future Artificial Intelligence and Technologies*, 1(2), pp. 174–190. https://doi.org/10.62411/faith.2024-33⏎

15.Chen, Z., Jiang, F., Cheng, Y., Gu, X., Liu, W. and Peng, J. (2018) ‘XGBoost Classifier for DDoS Attack Detection and Analysis in SDN-Based Cloud’, in *2018 IEEE International Conference on Big Data and Smart Computing (BigComp).* Shanghai, China, 15–17 January 2018, pp. 251–256. https://doi.org/10.1109/BigComp.2018.00044⏎

16.Rochim, A. F., Aziz, M. A. and Fauzi, A. (2020) ‘Design Log Management System of Computer Network Devices Infrastructures Based on ELK Stack’, in *2019 International Conference on Electrical Engineering and Computer Science (ICECOS).* Batam, Indonesia, 02–03 October 2019, pp. 338–342. https://doi.org/10.1109/ICECOS47637.2019.8984494⏎

17.STAMVS Networks (2024) *Suricata vs Snort*. Available at: www.stamus-networks.com/suricata-vs-snort#:~:text=Suricata’s%20ability%20to%20handle%20high,networks%20with%20heavy%20traffic%20loads [Accessed 10 October 2024].⏎

18.Ariffin, S. H. S., Chong, J. L. and Wahab, N. H. A. (2021) ‘Configuring Local Rule of Intrusion Detection System in Software Defined IoT Testbed’, in *2021 26th IEEE Asia-Pacific Conference on Communications (APCC)*. Kuala Lumpur, Malaysia, 1–13 October 2021, 298–303. http://dx.doi.org/10.1109/APCC49754.2021.9609824⏎

19.Yasar, K. (2022) *Elastic Stack (ELK Stack)*. Available at: www.techtarget.com/searchitoperations/definition/Elastic-Stack [Accessed 20 October 2024].⏎

20.Hariawan, F. R. and Sunaringtyas, S. U. (2022) ‘Design an Intrusion Detection System, Multiple Honeypot and Packet Analyzer Using Raspberry Pi 4 for Home Network’, in *2021 17th International Conference on Quality in Research (QIR): International Symposium on Electrical and Computer Engineering*. Depok, Indonesia, 13–15 October 2021, pp. 43–48. https://doi.org/10.1109/QIR54354.2021.9716189⏎

21.Waldman, A. (2024) ‘Microsoft confirms DDoS attack disrupted cloud services’, *TechTarget*, 31 July. Available at: www.techtarget.com/searchsecurity/news/366599523/Microsoft-confirms-DDoS-attack-disrupted-cloud-services [Accessed 25 November 2024].⏎

22.Waleed, A., Jamali, A. F. and Masood, A. (2022) ‘Which open-source IDS? Snort, Suricata or Zeek’, *Computer Networks,* 213, p. 109116. https://doi.org/10.1016/j.comnet.2022.109116⏎

Chapter 6

Federated Machine Learning Algorithm Aggregation Strategy for Collaborative Predictive Maintenance

Bhavna Saini, Vivek Kumar Verma, Bharat Singh, and Nidhi Kushwaha

DOI: 10.1201/9781003610168-6

6.1 Introduction

A key component of Industry 4.0, where automation, digital transformation, and the Internet of Things (IoT) are essential to upgrading industrial processes, is predictive maintenance (PdM). PdM eliminates the need for reactive maintenance and manual inspections by utilizing real-time sensor data, cloud computing, and advanced analytics to enable automated decision-making processes. The objectives of Industry 4.0, which include developing self-optimizing systems that can function intelligently without human assistance, are perfectly aligned with this. Additionally, the smooth connectivity made possible by IoT device integration enables predictive models to continuously assess the condition of equipment and recommend preventative actions. In addition to improving operational efficiency, this integrated ecosystem makes sure that firms maintain their competitiveness in a continually changing technological context [1].

Unplanned downtime is a serious problem for industries, frequently resulting in major financial losses, operational delays, and damage to one's reputation. PdM reduces these risks by addressing potential equipment failures before they happen, which can save a significant amount of money. For example, research indicates that unscheduled industrial downtime can cost businesses up to $260,000 per hour, while energy sector disruptions, including power outages or turbine problems, can result in millions of dollars in lost revenue. Equipment failures in the transportation industry can cause delays in services, raise maintenance costs, and interrupt vital supply lines. PdM not only lowers these costs but also maximizes resource allocation by extending the life of machinery and reducing unnecessary repairs.

The economic benefits of PdM make it a crucial strategy for sustainable operations as firms increasingly prioritize cost efficiency. PdM is used in many different industries, each with its own set of requirements and difficulties. IoT-enabled wind turbines can track variables like temperature, rotating speed, and vibration, allowing for the early identification of generator problems or blade fatigue in renewable energy. Predictive analytics can evaluate the operation of robotic arms and conveyor systems in automotive assembly lines to spot possible issues before they interfere with output. Similar to this, IoT-enabled transformers and substations in power grids may anticipate overheating or electrical surge-related breakdowns, guaranteeing a steady supply of electricity and cutting down on maintenance times.

In a variety of industrial industries, these examples show how PdM not only increases efficiency but also guarantees safety and dependability. With a focus on PdM in FOG and EDGE computing environments, this chapter aims to provide a thorough analysis of algorithm aggregation options inside Federated Learning (FL). Possible solutions found in the most recent literature are examined alongside important problems such as data heterogeneity, communication barriers, and resource restrictions. Figure 6.1 illustrates the FL architecture designed for collaborative PdM, in which a number of edge devices, including industrial machines with sensors, are essential components.

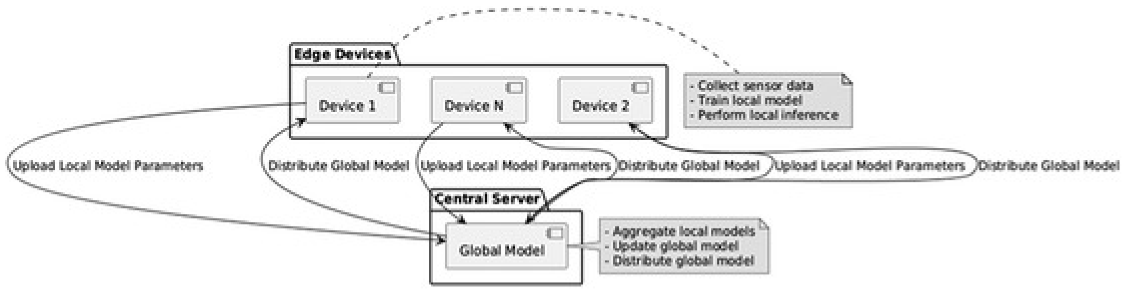


Figure 6.1 Federated learning for predictive maintenance.⏎

To build a thorough global model that incorporates knowledge from the whole fleet of equipment, the central server compiles these updates from all participating devices. To ensure that everyone benefits from the collective learning without jeopardizing data privacy, this global model is then redistributed to the edge devices. Until the prediction model reaches the required degree of accuracy and dependability, this iterative process of local training, central aggregation, and worldwide dissemination is repeated. Organizations can improve their PdM methods and guarantee timely interventions while upholding strict data privacy rules by utilizing this FL architecture. Data from multiple industrial equipment and sensors is sent to a single server for processing and analysis in centralized PdM designs, like the one shown in Figure 6.2.

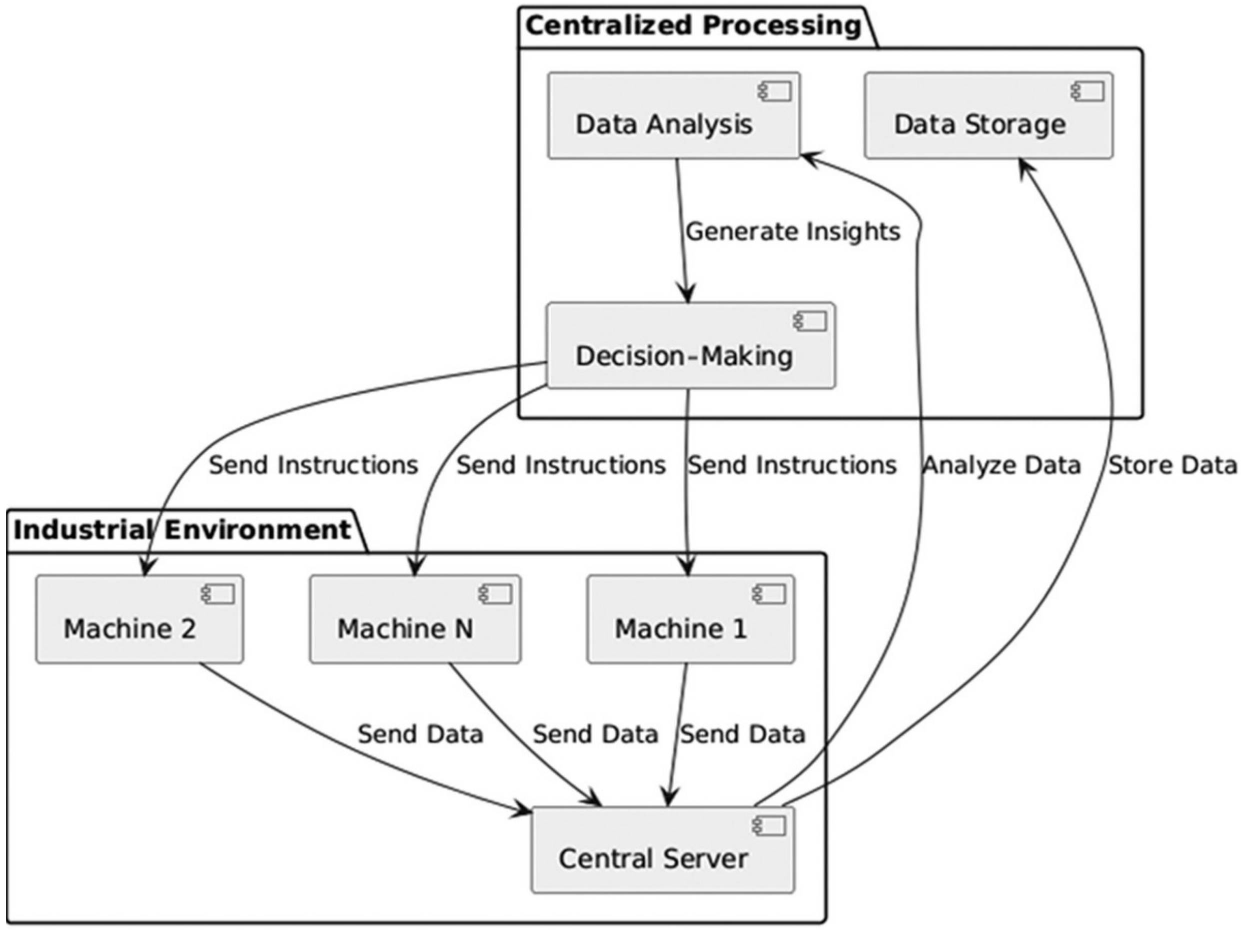


Figure 6.2 Centralize predictive maintenance.⏎

It can be prohibitively time-consuming to transfer raw data to a central server, process it, and then provide actionable insights back to the edge devices in high-frequency data environments like smart grids or factory assembly lines. Further aggravating delay problems are network dependability and bandwidth limitations as IoT implementations spread across widely separated locations. Because edge-based PdM solutions may process data closer to the source and provide real-time insights with minimal delay, these restrictions underscore the need for them. A central server processes raw data from multiple devices and sensors in traditional PdM techniques, which frequently rely on centralized machine learning models.

Although this makes comprehensive analysis possible, it raises serious privacy and security issues, especially in sectors like healthcare and critical infrastructure that handle sensitive data. Additionally, centralized systems raise the possibility of data breaches during storage and transfer, which is especially troublesome in settings with strict compliance requirements like General Data Protection Regulation (GDPR) [2]. Furthermore, centralized infrastructures may not be able to handle the massive amount of data produced by IoT devices in contemporary industrial systems, which could result in bottlenecks and decreased system efficiency [3]. These difficulties highlight the necessity of decentralized strategies like FL, which can maintain data localization while facilitating powerful predictive analytics.

6.1.1 Latency Issues

Due to latency created during data processing and transmission, centralized PdM systems are inherently limited in their ability to make decisions in real time. This delay can have disastrous effects in industrial settings where prompt action is essential to avert equipment malfunctions or safety issues [4]. For instance, the time required to transmit raw data to a central server, process it, and transmit actionable insights back to the edge devices can be prohibitive in high-frequency data environments like smart grids or factory assembly lines [5]. Furthermore, latency problems are made worse by network stability and bandwidth limitations when IoT implementations spread across widely separated locations [6]. These drawbacks emphasize the necessity of edge-based PdM systems, which can process data nearer to the source and provide insights in real time with little delay.

As shown in Figure 6.3, data from several industrial machines and sensors is sent to a central server for processing and analysis in centralized PdM architectures. Due to processing delays and data transmission durations, this centralized solution might introduce significant latency. This is especially troublesome in situations where real-time decision-making is necessary to avoid safety concerns or equipment breakdowns. Furthermore, latency problems are made worse by network stability and bandwidth limitations as IoT implementations spread across widely separated locations.

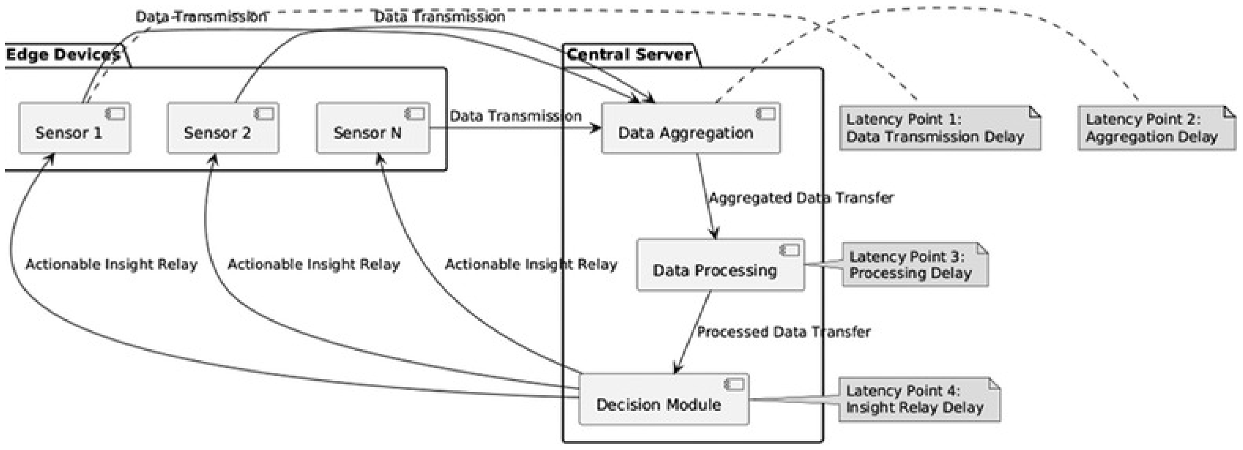


Figure 6.3 Latency in centralized predictive maintenance.⏎

6.1.2 Resource Constraints

Because industrial IoT devices create large amounts of data, centralized PdM solutions frequently need a large amount of processing and storage power. The processing power of edge or fog devices is limited, which presents a problem, particularly in resource-constrained contexts [7]. The centralized method significantly raises the cost of managing the computing load and maintaining massive data centers [8]. Continuous data transfers to central servers can also cause high energy usage and network bandwidth pressure, which is unsustainable for many enterprises [9]. These limitations highlight the necessity for lightweight, decentralized techniques like FL to maximize resource consumption while preserving PdM efficacy, as centralized systems are less practical for applications in distributed and resource-limited contexts.

Large volumes of operational data are continuously transmitted by a large number of industrial IoT devices to a central server for analysis in the systems shown in Figure 6.4. To handle and process the incoming data streams, this centralized method necessitates a significant amount of processing power and storage space at the central server. This centralized method may result in higher maintenance expenses for large-scale data centers and computational load management in settings where edge or fog devices have limited processing capability. Many organizations cannot handle the significant energy usage and network bandwidth strain caused by continuous data transfer to central servers. Because of these limitations, applications in distributed and resource-constrained environments find centralized systems less practical, highlighting the necessity of lightweight, decentralized techniques like FL to maximize resource use while maintaining PdM efficacy.

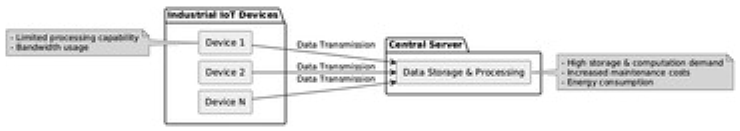


Figure 6.4 Resource constraints in centralized predictive maintenance.⏎

6.2 Federated Learning: A Paradigm for Collaborative Predictive Maintenance

In many industries, PdM has emerged as a key component for increasing operational effectiveness and decreasing downtime. Conventional PdM techniques frequently depend on centralized data gathering and analysis, raising issues with data security, privacy, and latency. By facilitating cooperative model training across numerous devices or organizations without requiring the sharing of raw data, FL emerges as a paradigm-shifting technique that tackles these issues. FL enables individual entities to train models on their local data in the context of PdM, sharing only the model parameters with a central server. In addition to protecting data privacy, this decentralized strategy makes use of everyone’s combined intelligence to create reliable prediction models. By integrating FL into PdM strategies, organizations can achieve more accurate and timely predictions, leading to proactive maintenance actions and significant cost savings.

6.2.1 Overview of Predictive Maintenance

In order to identify possible failures before they happen, PdM is a proactive approach that uses data analysis tools and techniques to identify abnormalities in operational processes and equipment. By scheduling maintenance tasks at the most convenient periods, this method enables businesses to minimize unscheduled downtime and lower operating expenses. PdM implementation is the ongoing observation of machinery using a variety of sensors that gather information on variables like temperature, vibration, pressure, and other pertinent metrics. This data is then subjected to sophisticated analytics and machine learning algorithms in order to forecast equipment breakdowns and optimize maintenance plans [10]. PdM has been much more effective in recent years as a result of the use of machine learning.

Conventional maintenance techniques, such as reactive maintenance, which fixes equipment after it breaks down, and preventive maintenance, which does maintenance at predetermined intervals, can result in needless maintenance tasks or unplanned equipment breakdowns. PdM, on the other hand, uses data-driven insights to forecast failures so that maintenance is only carried out when required. This enhances operational effectiveness and safety in addition to prolonging the equipment's lifespan [11, 12]. The use of PdM has been further accelerated by the emergence of the Industrial Internet of Things (IIoT). IIoT makes it possible for machines and devices to link seamlessly, which makes it easier to collect and analyze enormous volumes of data in real time. More precise forecasts of the condition and functionality of the equipment are made possible by this connectivity, which results in better maintenance plans.

6.2.2 Application of FL in PdM Offers Several Advantages

Data security and privacy are addressed by FL by storing the raw data on local devices and only communicating model parameters. This is crucial for sectors that handle sensitive data.
Decreased bandwidth usage and delay: The method lowers the quantity of data transferred between devices and the central server by just transmitting model parameters, which results in decreased bandwidth usage and delay.

Scalability: FL eliminates the requirement for a centralized data repository by enabling PdM systems to scale across multiple devices and locations. Recently, the incorporation of FL into PdM systems has been investigated. A one-dimensional (1D) convolutional neural network-bidirectional long short-term memory (1DCNN-BiLSTM) model, for example, was proposed by Ahn et al. [13] for the purpose of detecting time series anomalies and performing PdM in manufacturing processes.

The model efficiently extracts features from time series data and identifies abnormalities by combining a bidirectional LSTM with a 1D convolutional neural network. The study showed that by integrating this model with an FL framework, PdM can be done effectively while taking distributional variations in time series data into account. With a test accuracy of 97.2%, the suggested framework showed promise for practical uses. Pruckovskaja et al. [14] assessed the effectiveness of several FL aggregation techniques in the context of quality control and PdM in another investigation. According to the study, the data and how it is distributed among clients have a significant impact on the success of FL. FL can occasionally be a useful substitute for conventional central or local training techniques.

In order to provide useful insights into the actual application of FL in industrial settings, the study also presented a new FL dataset from a real-world quality inspection context. An FL technique that uses vibration sensor data from rotating machinery for condition monitoring and PdM was also proposed [15]. The method preserves data privacy and lessens reliance on the network by enabling distributed training on edge devices near the observed computers. Using real-world datasets for evaluation, it was found that the approach dramatically lowers resource and network utilization while enabling competitive performance when compared to prior results.

These studies demonstrate how FL may improve PdM systems by addressing issues with data privacy, lowering latency, and enhancing scalability. To fully reap the benefits of FL in PdM applications, however, issues including data heterogeneity, communication overhead, and resource limitations in edge and fog computing environments must be carefully considered.

6.3 Aggregation Strategies in Federated Learning

Aggregation techniques are essential for merging locally trained models from several clients into a single global model in FL. Federated Averaging (FedAvg), the fundamental technique, calculates a weighted average of the model parameters of each client, with weights usually proportionate to the size of each client’s dataset. However, in situations with heterogeneous data, this strategy may encounter difficulties [16]. Advanced aggregation approaches have been developed to address this issue. The Grouped Federated Averaging (GFedAvg) technique, for example, enhances model performance in non-IID environments by classifying clients according to data similarity prior to aggregation, improving upon FedAvg. A crucial area of research and development is the choice of aggregation approach, which has a substantial impact on the FL process's robustness, efficiency, and convergence [17].

A coherent global model can be created by integrating locally trained models from several clients using the aggregation procedures shown in Figure 6.5. FedAvg, the fundamental technique, uses private data to train a local model on each client, which then transmits the updated model parameters to a central server. The updated global model is created by the server combining these parameters and calculating a weighted average, usually based on the amount of each client's dataset. The global model is redistributed to clients for additional local training in an iterative process that continues until convergence is reached. FedAvg, however, may face difficulties in heterogeneous data environments, where the distribution of data among customers is identically distributed but not independent (non-IID).

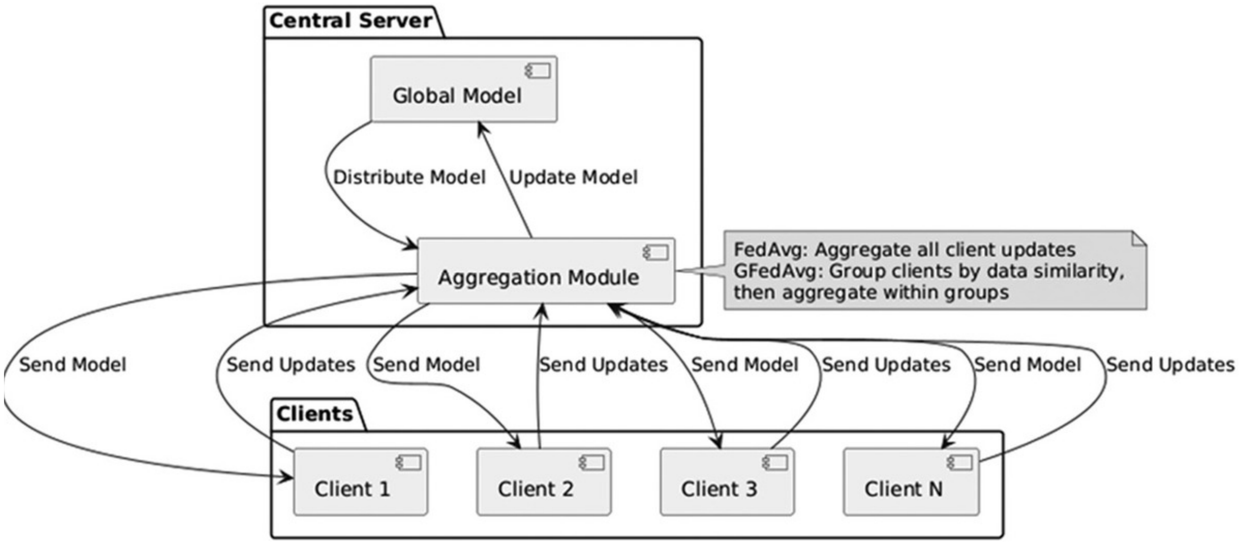


Figure 6.5 Federated learning (FL) aggregation strategy.⏎

In these situations, the diverse data distributions may cause the local models to diverge considerably, resulting in a global model that might not function at its best for every client. To tackle this problem, sophisticated aggregation methods have been created. By classifying clients according to data similarity prior to aggregation, the GFedAvg algorithm, for example, improves FedAvg [18]. Model updates within each group are collected, and the global model is updated by combining these group-level aggregates. With this method, similar data distributions are considered together during aggregation, which should enhance model performance in non-IID circumstances. The effectiveness, robustness, and convergence of the FL process are all strongly impacted by the aggregating approach selected.

6.3.1 The Foundational Aggregation Algorithm

FL is a decentralized machine learning technique that allows several clients to work together to build a common global model without exchanging local data. By allowing individual data sources to stay on-device, this paradigm reduces the risks associated with centralized data storage, which is especially advantageous in situations where data privacy and security are crucial. As the fundamental aggregation approach, the FedAvg algorithm is at the core of FL. A selection of participating customers receives an initial global model as part of FedAvg’s operation. Then, each client trains locally using their own private data, adjusting the model’s parameters as necessary. Clients send back to a central server their updated model parameter following the completion of the local training.

To create a new global model, the server combines these updates by calculating a weighted average, usually determined by the amount of each client’s dataset. Until the model converges to a performance level that is satisfactory, this iterative process keeps on. The FedAvg algorithm is a good fit for FL since it has a number of benefits. First, since raw data is never sent across a network, it improves privacy preservation by keeping data locally. Second, because clients and the server simply exchange model parameters rather than complete datasets, it lowers communication overhead. This effectiveness is especially crucial in settings with reduced bandwidth or when clients are devices with limited resources. FedAvg does have certain difficulties, though. One major problem is data heterogeneity, which occurs when data is not equally dispersed and independent across clients (non-IID). Under such circumstances, the local models that have been trained on different data may differ greatly, resulting in a global model that may not function as well for all customers [19]. Furthermore, different clients’ computational capacities and rates of engagement may have an impact on the training process's stability and convergence. Numerous improvements to the FedAvg algorithm have been suggested by experts in order to overcome these issues. For example, some strategies incorporate regularization techniques to lessen the impact of non-IID data, while others modify the aggregation weights to take client reliability or data quality into consideration. Despite these advancements, FedAvg remains the cornerstone of aggregation strategies in FL, providing a foundation upon which many modern FL algorithms are built.

6.3.2 Advanced Aggregation Techniques

Since FL allows several clients to jointly train a common global model without exchanging local data, it has attracted a lot of interest as a decentralized machine learning technique. By allowing individual data sources to stay on-device, this paradigm reduces the risks associated with centralized data storage, which is especially advantageous in situations where data privacy and security are crucial. As the fundamental aggregation approach, the FedAvg algorithm is at the core of FL.

A new global model is created by the server by combining these updates and calculating a weighted average, usually based on the amount of each client's dataset. This iterative procedure keeps going until the model converges to a level of performance that is suitable. With a number of benefits, the FedAvg algorithm is a good fit for FL. First, it improves privacy preservation by keeping data locally because raw data is never sent over the network. Second, it lowers communication overhead because clients and the server simply exchange model parameters rather than complete datasets. In settings with finite bandwidth or where clients are devices with limited resources, this efficiency is very crucial. FedAvg does have certain difficulties, though.

The distribution of data among customers is not independent and identical (non-IID), which is a major problem caused by data heterogeneity. When this happens, the local models that were trained on different data sets may differ greatly, which could result in a global model that isn’t the best for all of the customers. Diverse customer engagement rates and computational capacities can also impact the training process’s stability and convergence. Scholars have suggested a number of improvements to the FedAvg algorithm in order to overcome these difficulties. For example, some methods add regularization techniques to lessen the effects of non-IID data, while others modify the aggregate weights to account for data quality or client trustworthiness. Despite these advancements, FedAvg remains the cornerstone of aggregation strategies in FL, providing a foundation upon which many modern FL algorithms are built.

6.3.2.1 *FedProx*

FL enables decentralized model training across multiple devices, allowing each to train on its local data and share model updates with a central server. A significant challenge in FL lies in system heterogeneity, where devices have varying computational capabilities and data distributions. This variability can lead to instability and divergence during the training process. To address these challenges, the Federated Proximal (FedProx) algorithm was introduced. FedProx modifies the standard FL objective by incorporating a proximal term into the local loss functions. This proximal term acts as a regularization factor, constraining the local model updates to remain close to the current global model parameters. By doing so, FedProx mitigates the impact of system heterogeneity, ensuring that local updates do not deviate excessively from the global model, thereby enhancing stability during training. The proximal term in FedProx is scaled by a parameter μ, which controls the strength of the regularization. A larger μ places a stronger emphasis on keeping local updates close to the global model, which can be beneficial in highly heterogeneous environments [20]. Conversely, a smaller μ allows for more significant local updates, which may be advantageous when devices have similar data distributions and computational capabilities. This flexibility enables FedProx to adapt to various levels of system heterogeneity, balancing the trade-off between local adaptation and global consistency. Empirical studies have demonstrated that FedProx provides more robust convergence compared to traditional FL algorithms, especially in settings with significant heterogeneity. By allowing devices to perform a variable amount of local computation and incorporating the proximal term, FedProx effectively addresses the challenges posed by system variability, leading to improved performance and stability in federated learning environments.

The FedProx algorithm addresses challenges associated with system and statistical heterogeneity by introducing a proximal term to the local objective functions. This proximal term serves as a regularization factor, constraining local updates to remain close to the global model parameters, thereby enhancing stability during training. For instance, in a network of hospitals collaboratively developing a predictive model for patient readmission rates, each hospital trains a local model on its private patient data and shares the updated model parameters with a central server. Due to differences in patient demographics and treatment protocols, the data distributions across hospitals are heterogeneous. FedProx mitigates the impact of this heterogeneity by incorporating the proximal term, ensuring that local updates do not deviate excessively from the global model, leading to more robust convergence and improved model performance across the network.

In the context of federated learning, the FedProx algorithm addresses challenges associated with system and statistical heterogeneity by introducing a proximal term to the local objective functions. This proximal term serves as a regularization factor, constraining local updates to remain close to the global model parameters, thereby enhancing stability during training. For instance, in a network of hospitals collaboratively developing a predictive model for patient readmission rates, each hospital trains a local model on its private patient data and shares the updated model parameters with a central server. Due to differences in patient demographics and treatment protocols, the data distributions across hospitals are heterogeneous. FedProx mitigates the impact of this heterogeneity by incorporating the proximal term, ensuring that local updates do not deviate excessively from the global model, leading to more robust convergence and improved model performance across the network (Figure 6.6).

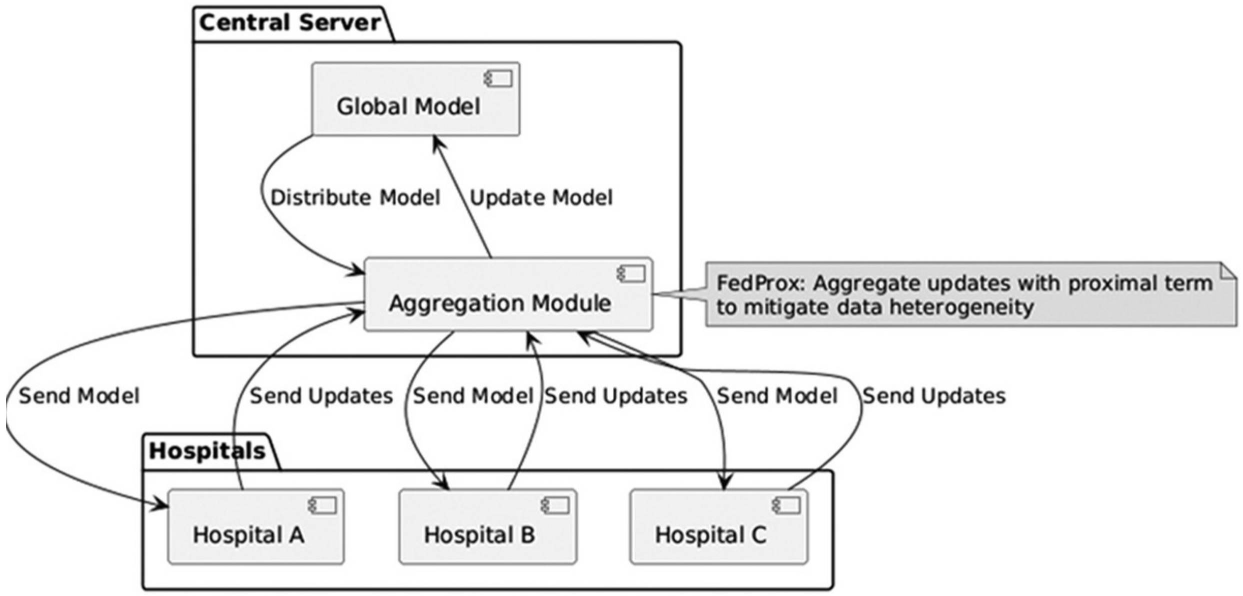


Figure 6.6 FedProx in collaborative healthcare.⏎

6.3.2.2 *FedMA*

In FL, effectively aggregating models from clients with non-independent and identically distributed (non-IID) data poses significant challenges. The Federated Matched Averaging (FedMA) algorithm addresses this issue by constructing the global model in a layer-wise manner. Specifically, FedMA matches and averages hidden elements—such as channels in convolutional layers, hidden states in LSTMs, and neurons in fully connected layers—that exhibit similar feature extraction characteristics across client models. By aligning and combining these analogous components, FedMA enhances the global model's performance in non-IID settings. Empirical studies have demonstrated that FedMA not only outperforms traditional FL algorithms like FedAvg on deep neural network [21] architectures but also reduces communication overhead during training [22] (Figure 6.7).

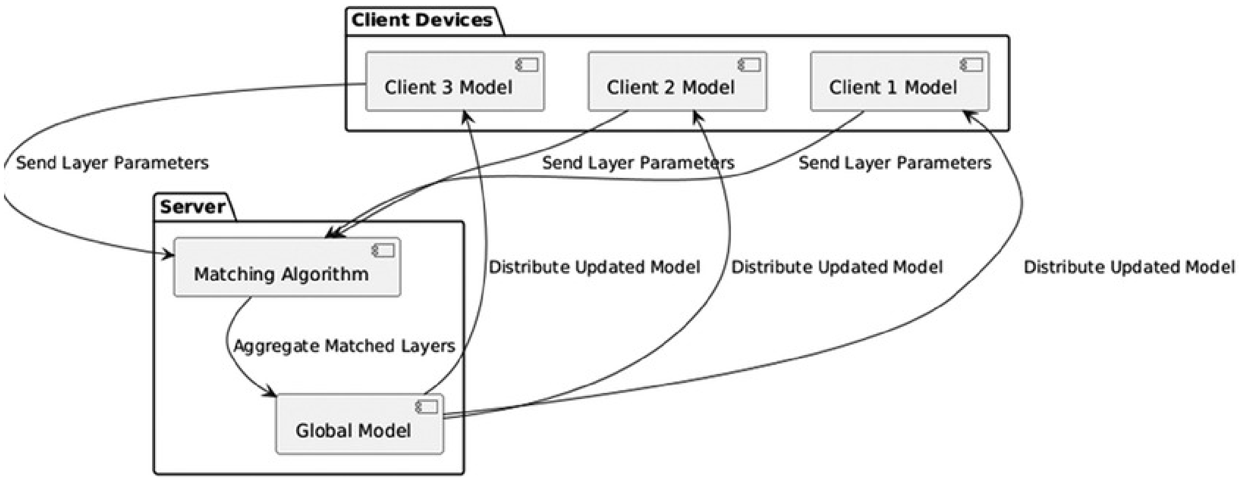


Figure 6.7 FedMA aggregating models.⏎

6.3.2.3 *FedNova*

In FL, client heterogeneity—stemming from variations in local data distributions and computational capabilities—can lead to inconsistencies in model updates, adversely affecting convergence. To address this, the Federated Normalized Averaging (FedNova) algorithm normalizes local updates by the number of local iterations each client performs, ensuring that each client's contribution to the global model is appropriately scaled. This normalization accounts for disparities in client computation, leading to more consistent and stable convergence across diverse client environments (Figure 6.8).

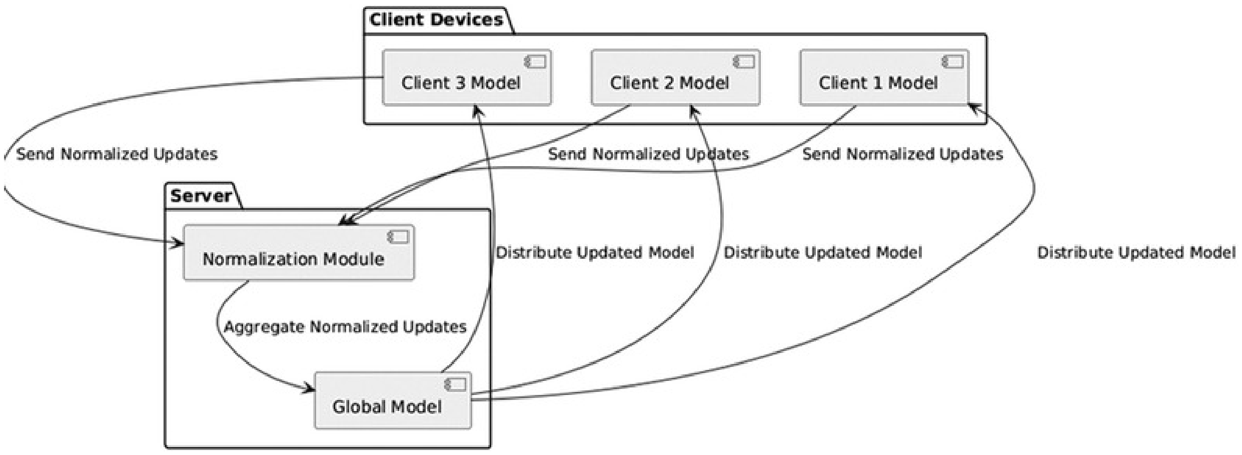


Figure 6.8 FedNova aggregating models.⏎

The client devices often exhibit significant variability in computational capabilities and data distributions, leading to differences in the number of local updates each client performs during a training round. This heterogeneity can cause objective inconsistency, where the global model converges to a stationary point of a mismatched objective function, potentially diverging from the true objective. The FedNova algorithm addresses this issue by normalizing local updates according to each client's training progress. Specifically, FedNova adjusts the aggregation of local models by accounting for the number of local updates performed by each client, thereby eliminating objective inconsistency and preserving fast error convergence. This approach ensures that the global model accurately reflects the contributions of all clients, regardless of their individual computational capacities or data heterogeneity.

6.3.3 Optimization through Metaheuristics

Optimizing aggregation algorithms is essential in FL to balance computational efficiency and model performance, particularly in situations with limited resources. To improve these aggregation techniques, metaheuristic algorithms like Genetic Algorithms (GAs) and Particle Swarm Optimization (PSO) have been used. PSO optimizes a problem by iteratively enhancing potential solutions in relation to a specified quality metric, drawing inspiration from the social behavior of bird flocks. PSO can be used in FL to find the best aggregation weights, which will increase the global model's accuracy and rate of convergence. The selection and combination of local models in FL have also been optimized through the use of GAs, which simulate the natural selection process.

GAs are able to efficiently look for optimal or nearly optimal aggregation procedures that strike a balance between resource usage and performance by evolving a population of candidate solutions. These metaheuristic techniques provide adaptable and effective ways to improve FL aggregation, especially when handling diverse data distributions and client-specific processing resources. In contexts with limited resources, aggregation strategy optimization is essential for striking a balance between computational efficiency and model performance. The social behavior of fish schools or flocks of birds serves as an inspiration for PSO. PSO can be applied to FL to find the best aggregation weights for merging local models into a global model. [23–27]

Every particle in the swarm is a collection of aggregation weights that could be a solution. In order to converge toward ideal weights that minimize a predetermined loss function, these particles modify their placements in response to both their own and nearby particles' experiences. When several client devices contribute to a global model, for instance, PSO can maximize each client's model update contribution, improving convergence rates and model correctness. By using operations like selection, crossover, and mutation to evolve solutions over generations, GAs replicate the process of natural selection. GA is able to choose and combine local models in FL in an optimal manner. Each member of the population might, for example, represent a subset of the client models that were chosen for aggregation. By using iterative evolution, the GA finds subsets that, when combined, produce a global model with good performance. This method is especially helpful when choosing the best subset of client models is difficult because of resource limitations or different client data quality.

A global objective function $F(w)=\sum k=1KpkFk(w)$, where pk is the relative contribution weight of client k and $Fk(w)$ is the local objective function of client k. The objective of metaheuristic algorithms is to minimize the global loss $F(w)$ by optimizing the aggregation weights pk. Particles in PSO adjust their positions (weights) according to velocity vectors that are impacted by both individual and collective experiences. Over iterations, candidate solutions (aggregation techniques) in GAs are developed to enhance the performance of the global model. By integrating metaheuristic optimization techniques into FL aggregation strategies, it is possible to achieve a more efficient balance between model accuracy and computational resource utilization, thereby enhancing the overall effectiveness of FL systems.

6.4 Challenges in Fog and Edge Environments

The constraints of standard cloud computing have been addressed by fog and edge computing, which have become key paradigms, especially for applications that need real-time processing and low latency. These methods improve responsiveness and lower bandwidth use by decentralizing data processing and putting it closer to the data source. However, there are a number of difficulties brought about by the dispersed character of fog and edge situations. The deployment of fog nodes at the edge of the network exposes sensitive data, making them potential targets for cyberattacks, raising serious concerns about data privacy. The large number of linked devices and gateways, each with its own set of vulnerabilities, makes security problems much worse. Another significant issue is interoperability; the wide range of edge devices and fog nodes frequently lack established protocols, making it difficult to integrate and work together seamlessly [28].

Furthermore, overseeing a dispersed infrastructure introduces complexity, requiring strong tools and techniques to track and improve performance. For fog and edge computing systems to be implemented and run effectively, certain issues must be resolved.

6.4.1 Data Heterogeneity

Effective model training and convergence in fog and edge computing environments are significantly hampered by data heterogeneity. These networks' decentralized structure makes it more difficult to create reliable machine learning models since they provide a variety of non-IID (different, non-independent, and identically distributed) data across devices. Clustered federated learning (CFL) has been suggested as a solution to this problem. By clustering devices with comparable data distributions, CFL enables the training of specialized models within each group. This method adapts learning procedures to the unique features of each cluster’s data, improving model performance. For example, the Iterative Federated Clustering Algorithm efficiently handles data heterogeneity in FL environments by switching between estimating user cluster IDs and optimizing model parameters for each cluster.

6.4.2 Communication Overhead

Iterative transmission of model updates between multiple edge devices and a central server creates a substantial communication cost for FL in fog and edge computing environments. Especially in environments with limited bandwidth, this frequent communication can put a burden on network resources and raise latency. Techniques like model compression and gradient scarification have been developed to lessen these difficulties. Gradient scarification reduces the quantity of data supplied without sacrificing model performance by only sending the most important gradients at each update. For example, considerable data reduction during model training can be accomplished by using gradient scarification in wireless traffic prediction. By lowering the size of the model parameters shared between devices and the server, model compression techniques like quantization and encoding further reduce communication costs. FL systems can improve scalability and performance in fog and edge environments by putting these strategies into practice and achieving more effective communication.

6.4.3 Resource Constraints

Devices in fog and edge computing environments frequently work with strict resource limitations, such as constrained memory, computational power, and energy capacity. These restrictions make it difficult to execute FL, which usually calls for a large investment of resources for both communication with central servers and training local models. In response to these difficulties, researchers have devised methods like selective participation and quantized updates to produce lightweight algorithms appropriate for environments with limited resources. By diminishing the accuracy of model parameters as they are being transmitted, quantized updates lower the volume of data that is exchanged between servers and devices. By allocating distinct bit-widths to different layers of a deep learning network, for instance, mixed-precision quantization successfully strikes a balance between resource usage and model accuracy.

This method reduces the processing and communication requirements, enabling devices with minimal capacity to take part in FL. By using a subset of devices in each training round, selective participation procedures further increase efficiency. The system can maximize resource consumption and preserve model performance by carefully choosing participants based on factors like the importance of their local data or the availability of resources at the moment. An adaptive federated learning system, for example, would enable edge devices to independently update local models, allowing for sporadic connectivity and fluctuating resource limitations. By putting these strategies into practice, FL systems can function well with fog and edge devices, encouraging a wider uptake of decentralized learning in settings with limited resources.

6.5 Applications of Federated Learning in Predictive Maintenance

PdM aims to anticipate equipment failures by analyzing data collected from machinery, thereby optimizing maintenance schedules and reducing unexpected downtimes. FL has emerged as a pivotal technology in this domain, enabling collaborative model training across decentralized devices while preserving data privacy. This section explores the applications of FL in PdM, focusing on equipment monitoring, fault diagnosis, and energy optimization.

6.5.1 Equipment Monitoring

In industrial settings, continuous equipment monitoring is essential for maintaining operational efficiency. Traditional centralized approaches to condition monitoring often require transferring vast amounts of sensor data to central servers, posing challenges related to data privacy and network reliability. FL addresses these issues by facilitating the decentralized training of machine learning models directly on edge devices located near the machinery. For instance, Becker et al. [29] proposed an autoencoder-based FL method utilizing vibration sensor data from rotating machines. This approach enables distributed training on-premises, allowing knowledge transfer across organizational boundaries without sharing raw data, thereby preserving privacy and reducing network load.

6.5.2 Fault Diagnosis

Accurate fault diagnosis is critical for preventing equipment failures and ensuring safety. FL enhances fault diagnosis by enabling the development of robust models trained on diverse datasets from multiple sources without compromising data confidentiality. By aggregating knowledge from various industrial environments, FL-based models can achieve improved generalization and fault detection capabilities. This collaborative approach allows for the identification of fault patterns that may not be apparent within isolated datasets, leading to more reliable and timely fault diagnosis.

6.5.3 Energy Optimization

Energy consumption is a significant concern in industrial operations, both from a cost and an environmental perspective. FL contributes to energy optimization by facilitating the development of models that analyze operational data to identify energy-saving opportunities. By collaboratively training models across different facilities, organizations can uncover best practices and operational adjustments that lead to reduced energy usage. This decentralized approach ensures that sensitive operational data remains on-site, addressing privacy concerns while promoting energy-efficient practices across the industry. FL offers a transformative approach to PdM by enabling collaborative, privacy-preserving model training for equipment monitoring, fault diagnosis, and energy optimization. Its application in industrial settings not only enhances the accuracy of predictive models but also fosters cross-organizational collaboration without compromising data security.

6.6 Key Research Gaps and Future Directions

FL has garnered significant attention due to its potential to enable collaborative model training without compromising data privacy. However, several research gaps exist that hinder its widespread adoption and efficacy. This section delves into three critical areas requiring further exploration: adaptive aggregation mechanisms, enhanced privacy measures, and scalability challenges.

6.6.1 Adaptive Aggregation Mechanisms

In FL, the central server aggregates model updates from distributed clients to form a global model. Traditional aggregation methods, such as FedAvg, often assume homogeneous data distributions across clients. However, in practical scenarios, data is typically non-IID, leading to suboptimal model performance. To address this, researchers have proposed adaptive aggregation techniques that account for data heterogeneity. For instance, Yeganeh et al. [30] introduced the inverse distance aggregation method, which assigns weights to client updates based on the distance between local and global model parameters, thereby mitigating the impact of outliers and enhancing convergence rates.

6.6.2 Enhanced Privacy Mechanisms

While FL inherently promotes data privacy by keeping raw data on local devices, vulnerabilities remain, particularly concerning inference attacks that can reconstruct sensitive information from model updates. To bolster privacy, various techniques have been explored. Differential privacy (DP) introduces noise to model updates, obscuring individual data contributions. Secure multiparty computation (SMC) enables collaborative computations without revealing individual inputs. Despite these advancements, challenges persist in balancing privacy with model utility and computational efficiency. Recent surveys provide comprehensive overviews of these privacy-preserving methods and highlight the need for novel solutions that effectively integrate DP and SMC within the FL framework.

6.6.3 Scalability Challenges

Scalability remains a significant concern in FL, especially when dealing with large numbers of clients and substantial model sizes. Synchronous aggregation methods can lead to communication bottlenecks and increased latency, as the central server must wait for updates from all clients. Asynchronous aggregation offers a potential solution by allowing the server to update the global model with client updates as they arrive, thus reducing waiting times. Nguyen et al. [31] proposed a buffered asynchronous aggregation approach, FedBuff, which combines the benefits of synchronous and asynchronous methods. FedBuff maintains a buffer of incoming updates, enabling efficient aggregation while remaining compatible with privacy-preserving techniques like secure aggregation. Empirical results demonstrate that FedBuff enhances training efficiency and scalability in FL systems. Addressing these research gaps is crucial for the advancement and practical deployment of FL across various domains. Future work should focus on developing robust adaptive aggregation strategies, enhancing privacy-preserving mechanisms without compromising model performance, and designing scalable FL architectures capable of handling diverse and extensive client networks.

6.7 Conclusion

FL has emerged as a transformative approach in PdM, enabling collaborative model training across decentralized devices while preserving data privacy. By facilitating the analysis of equipment data without necessitating centralized data storage, FL addresses critical concerns related to data confidentiality and network reliability. Studies have demonstrated the efficacy of FL in various industrial applications, including condition monitoring and fault diagnosis. Despite these advancements, several challenges persist that warrant further research. Adaptive aggregation mechanisms are needed to address data heterogeneity among clients, as traditional methods like FedAvg may not perform optimally with non-IID data distributions. Enhanced privacy measures are also crucial, as FL, while inherently privacy-preserving, is still susceptible to inference attacks that can reconstruct sensitive information from model updates. Balancing privacy with model utility and computational efficiency remains an open area of investigation. Additionally, scalability challenges must be addressed to accommodate large numbers of clients and substantial model sizes, with asynchronous aggregation methods offering potential solutions to reduce communication bottlenecks and latency. FL offers a promising framework for PdM by enabling collaborative, privacy-preserving analysis of equipment data. Ongoing research focused on adaptive aggregation, robust privacy mechanisms, and scalable architectures will be pivotal in realizing the full potential of FL in industrial applications.

References

1. McMahan, B., et al. (2017). Communication-efficient learning of deep networks from decentralized data.

Proceedings of AISTATS.⏎

2. Kairouz, P., et al. (2021). Advances and open problems in federated learning. *Foundations and Trends in Machine Learning* 14(1–2), 1–210.⏎
3. Zhao, Y., et al. (2018). Federated learning with non-IID data*. Proceedings of NeurIPS.*⏎
4. Li, T., et al. (2018). Federated optimization in heterogeneous networks*. Proceedings of MLSys.*⏎
5. Yang, Q., Liu, Y., Chen, T. and Tong, Y. (2019). Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10(2), 1–19.⏎
6. Bonawitz, K., et al. (2019). Towards federated learning at scale: System design. *Proceedings of SysML.*⏎
7. Du, J., Qin, N., Huang, D., Jia, X. and Zhang, Y. (2023). Lightweight FL: A Low-Cost Federated Learning Framework for Mechanical Fault Diagnosis with Training Optimization and Model Pruning. *IEEE Transactions on Instrumentation and Measurement* 73, 1–14.⏎
8. de Melo Rosa, G. L., Mohanram, P., Gilerson, A. and Schmitt, R. H. (2023). *Architecture for edge-based predictive maintenance of machines using federated learning and multi sensor platforms*. Preprints.org; 2023. DOI: 10.20944/preprints202305.1563.v1.⏎
9. Han, P., Wang, S. and Leung, K. K. (2020, November). Adaptive gradient sparsification for efficient federated learning: An online learning approach. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)* (pp. 300–310). IEEE.⏎
10. Felbab, V., Kiss, P. and Horváth, T. (2019, September). Optimization in federated learning. In *ITAT* (pp. 58–65).⏎

11. Sun, W., Lei, S., Wang, L., Liu, Z. and Zhang, Y. (2020). Adaptive federated learning and digital twin for industrial internet of things. *IEEE Transactions on Industrial Informatics* 17(8), 5605–5614.⏎
12. Wangni, J., Wang, J., Liu, J. and Zhang, T. (2018). Gradient sparsification for communication-efficient distributed optimization. *Advances in Neural Information Processing Systems* 31, 1–11.⏎
13. Ahn, J., Lee, Y., Kim, N., Park, C. and Jeong, J. (2023). Federated learning for predictive maintenance and anomaly detection using time series data distribution shifts in manufacturing processes. *Sensors* 23(17), 7331. https://doi.org/10.3390/s23177331⏎
14. Pruckovskaja, V., Weissenfeld, A., Heistracher, C., Graser, A., Kafka, J., Leputsch, P., Schall, D. and Kemnitz, J. (2023). Federated learning for predictive maintenance and quality inspection in industrial applications. *arXiv* preprint arXiv:2304.11101. https://arxiv.org/abs/2304.11101⏎
15. Kea, K., Han, Y. and Kim, T-K. (2023). Enhancing anomaly detection in distributed power systems using autoencoder-based federated learning. *PLoS One* 18(8), e0290337.⏎
16. Ahn, J., Lee, Y., Kim, N., Park, C. and Jeong, J. (2023). Federated learning for predictive maintenance and anomaly detection using time series data distribution shifts in manufacturing processes. *Sensors* 23(17), 7331.⏎
17. Liu, J., Lou, J., Xiong, L., Liu, J. and Meng, X. (2021). Projected federated averaging with heterogeneous differential privacy. *Proceedings of the VLDB Endowment* 15(4), 828–840.⏎

18. Le, K., et al. (2024). Efficiently assemble normalization layers and regularization for federated domain generalization. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.⏎
19. Ma, X., et al. (2022). A state-of-the-art survey on solving non-iid data in federated learning. *Future Generation Computer Systems* 135, 244–258.⏎
20. Yuan, X. and Li, P. (2022). On convergence of FedProx: Local dissimilarity invariant bounds, non-smoothness and beyond. *Advances in Neural Information Processing Systems* 35, 10752–10765.⏎
21. Gupta, P., Anand, A., Agarwal, P. and McArdle, G. (2024). Neural network inspired efficient scalable task scheduling for cloud infrastructure. *Internet of Things and Cyber-Physical Systems* 4, 268–279.⏎
22. Wang, H., et al. (2020). Federated learning with matched averaging. *arXiv preprint arXiv:2002.06440*.⏎
23. Mustapha, S. D. S. and Gupta, P. (2024). DBSCAN inspired task scheduling algorithm for cloud infrastructure. *Internet of Things and Cyber-Physical Systems* 4, 32–39.⏎
24. Gupta, P., Rawat, P. S., kumar Saini, D., Vidyarthi, A. and Alharbi, M. (2023). Neural network inspired differential evolution based task scheduling for cloud infrastructure. *Alexandria Engineering Journal* 73, 217–230.
25. Madhusudhan, H. S., Gupta, P., Saini, D. K. and Tan, Z. (2023). Dynamic virtual machine allocation in cloud computing using elephant herd optimization scheme. *Journal of Circuits, Systems and Computers* 32(11), 2350188.
26. Rawat, P. S., Gaur, S., Barthwal, V., Gupta, P., Ghosh, D., Gupta, D. and Rodrigues, J. J. C. (2025). Efficient virtual

machine placement in cloud computing environment using BSO-ANN based hybrid technique. *Alexandria Engineering Journal* 110, 145–152.

27. HS, M. and Gupta, P. (2024). Federated learning inspired Antlion based orchestration for Edge computing environment. *PLoS One* 19(6), Art. e0304067.⏎
28. Svorobej, S., et al. (2019). Simulating fog and edge computing scenarios: An overview and research challenges. *Future Internet* 11(3), 55.⏎
29. Becker, S., Styp-Rekowski, K., Stoll, O. V. L. and Kao, O. (2022). Federated learning for autoencoder-based condition monitoring in the Industrial Internet of Things. In *IEEE International Conference on Big Data (IEEE BigData 2022)*.⏎
30. Yeganeh, Y., et al. (2020). Inverse distance aggregation for federated learning with non-iid data. *Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning: Second MICCAI Workshop, DART 2020, and First MICCAI Workshop, DCL 2020, Held in Conjunction with MICCAI 2020, Lima, Peru, October 4–8, 2020, Proceedings 2*. Springer International Publishing.⏎
31. Nguyen, J., et al. (2022). Federated learning with buffered asynchronous aggregation. *International Conference on Artificial Intelligence and Statistics*. PMLR.⏎

Chapter 7

Advance Machine Learning Algorithm Aggregation Strategy for Decentralized Collaborative Models

Nayana Singh and Prateek Kumar Soni

DOI: 10.1201/9781003610168-7

7.1 Introduction

Predictive maintenance (PdM) is a critical application of machine learning (ML) in industrial settings, aimed at minimizing downtime, reducing maintenance costs, and improving operational efficiency. By leveraging ML techniques, PdM systems can analyze sensor data and operational parameters to predict potential equipment failures before they occur, enabling timely interventions.

Traditional PdM approaches rely on centralized ML models, where data from multiple machines, sensors, or locations is collected and processed in a central server.

However, this centralized model presents several challenges:

- **Data Privacy Concerns:** Industrial and enterprise data often contain sensitive information that organizations are unwilling or unable to share due to regulatory and competitive constraints.
- **High Data Transmission Costs:** Continuously sending large volumes of sensor and log data to a central location increases bandwidth usage and storage costs.
- **Scalability Limitations:** As the number of connected devices grows, the computational and network burden on central servers becomes unsustainable.

To address these issues, Federated Machine Learning (FML) [1] has emerged as a promising alternative. FML enables multiple edge devices, such as industrial Internet of Things (IIoT) sensors, manufacturing robots, and predictive maintenance systems, to collaboratively train a shared ML model without exchanging raw data. Instead, each device trains a local model using its own data and only shares model updates (e.g., gradients or weights) with a central server or aggregator, ensuring data privacy while maintaining model accuracy.

One of the key challenges in FML-based predictive maintenance is *aggregating the locally trained models* from different edge devices effectively. The choice of aggregation strategy significantly impacts the final model's performance, robustness, and adaptability to heterogeneous data distributions across different devices.

This document explores various *algorithm aggregation strategies* used in FML for collaborative predictive maintenance, analyzing their strengths, limitations, and applicability in real-world industrial environments. By selecting an appropriate aggregation approach, organizations can enhance the accuracy, reliability, and efficiency of PdM solutions while maintaining data security and minimizing communication overhead.

7.2 Challenges in Federated Predictive Maintenance

While FML [2] offers a decentralized approach to predictive maintenance, its implementation comes with several challenges that must be addressed to ensure effective model training and deployment. These challenges stem from industrial constraints such as data security, network limitations, and the variability in sensor-generated data across different machines.

7.2.1 Data Privacy and Security

One of the primary motivations for using FML in predictive maintenance is the need to preserve *data privacy and security*. Industrial environments generate vast amounts of sensitive operational data, including machine performance metrics, error logs, and sensor readings. Sharing this data with a central server for traditional ML raises several concerns:

- **Regulatory Compliance:** Industries such as healthcare, manufacturing, and energy are subject to strict data protection regulations (e.g., General Data Protection Regulation [GDPR] and Health Insurance Portability and Accountability Act [HIPAA]) that prohibit the transfer of sensitive operational data outside the organization’s premises.

- **Confidentiality Risks:** Companies are often reluctant to share proprietary machine data due to competitive reasons, as it may contain trade secrets or insights into operational strategies.
- **Risk of Cyberattacks:** Centralized data storage makes systems vulnerable to cyber threats, including unauthorized access, data breaches, and industrial espionage.

FML mitigates these risks by allowing training to occur locally on edge devices, but additional security measures, such as *secure aggregation, encryption, and differential privacy*, are necessary to prevent adversarial attacks and model inversion threats.

7.2.2 Heterogeneous Data Sources

In industrial settings, sensor data is collected from a variety of machines, each differing in:

- **Sensor Types and Configurations:** Different equipment manufacturers use different sensors, leading to variations in data format, resolution, and measurement units.
- **Data Sampling Rates:** Some machines generate data at high frequency (e.g., every millisecond), while others log readings at irregular intervals.
- **Data Quality and Noise:** Variability in environmental conditions, sensor aging, and calibration differences introduce inconsistencies in data quality.

Such heterogeneity makes it challenging to train a *generalizable federated model.* If local data distributions differ significantly (a phenomenon known as *non-IID data*), the global model may fail to converge or may favor certain machine types over others. Addressing this requires *advanced aggregation techniques, adaptive learning rates, and personalized federated learning models*.

7.2.3 Communication Overhead

FML reduces the need to transmit raw data, but frequent model updates between edge devices and the central aggregator introduce new communication challenges:

- **Bandwidth Constraints:** Industrial environments often have limited network bandwidth, making it costly to transmit large model updates frequently.
- **Network Latency:** Synchronizing multiple devices over a distributed network can lead to delays, especially when devices are in remote locations with unstable connectivity.
- **Asynchronous Updates:** Some edge devices may have higher computational capabilities than others, leading to **straggler effects**, where slower devices delay the overall training process.

To optimize communication efficiency, techniques such as **compressed updates, sparse model sharing, and asynchronous federated learning** are used to reduce network congestion while maintaining model performance.

7.2.4 Model Convergence

Unlike traditional centralized training, where data is uniform and well-distributed, federated training must ensure *global model convergence* [3] despite disparities in local datasets. Key challenges include:

- **Divergent Local Models:** Since each device trains on a unique subset of data, local models may develop biases that negatively impact global model performance.
- **Unbalanced Data Contributions:** Some machines may generate significantly more data than others, skewing the training process and leading to an imbalanced model.
- **Drift in Data Distributions:** Equipment behavior and operating conditions evolve over time, causing data distributions to shift and reducing model accuracy.

To achieve stable convergence*, adaptive federated optimization algorithms* such as *Federated Averaging (FedAvg), momentum-based aggregation, and personalized federated learning* can be employed to balance local updates while preventing drastic deviations in the global model.

7.3 Federated Learning Framework for Predictive Maintenance

FML provides a decentralized approach to PdM by enabling multiple edge devices, such as industrial sensors, IoT-enabled machinery, and smart controllers, to collaboratively train a shared ML model. Unlike traditional ML models, which require raw data to be centralized, FML ensures that data remains on the edge devices, improving privacy, reducing network congestion, and enhancing scalability.

This section details the architecture, workflow, and key components of an FML-based predictive maintenance framework.

7.3.1 Overview

The primary goal of the FML framework in PdM [5] is to build a highly accurate failure prediction model while preserving data privacy. Instead of transmitting raw sensor data to a central server, individual edge devices perform local training and only send model updates, such as gradients or weights, to a central coordinator. The central coordinator aggregates updates from multiple edge devices to refine the global model, which is then redistributed for further training. This iterative process continues until the global model reaches an optimal level of accuracy and convergence.

Key benefits of FML in PdM include:

1. **Data Privacy and Security:** No raw data leaves the local devices, ensuring compliance with industrial privacy regulations.
2. **Reduced Communication Overhead:** Only model updates, rather than entire datasets, are transmitted, lowering bandwidth usage.
3. **Scalability:** The system can support a large number of edge devices without overloading a centralized server.
4. **Adaptability to Edge Conditions:** Local training accounts for variations in machine behavior and environmental conditions, improving model robustness.

7.3.2 Workflow

The FML framework for PdM follows an iterative training process, involving multiple rounds of communication between edge devices and the central coordinator. Below is a step-by-step breakdown of this workflow:

1. **Initialization**
   - The central coordinator initializes a global model with random or pre-trained weights.
   - The model architecture and hyperparameters, such as learning rate and number of epochs, are defined.
   - The global model is sent to participating edge devices, such as industrial sensors and PdM units.
2. **Local Training at Edge Devices**
   - Each edge device receives the global model and trains it locally using its own sensor data.
   - The training process includes:
     - Feature extraction: Processing raw sensor data, such as vibration, temperature, and pressure, into meaningful features.
     - Anomaly detection: Identifying early signs of machine degradation.
     - Failure prediction: Training the model to predict failures based on historical trends.
   - After training, each device computes model updates, such as weights and gradients, without sharing raw data.
3. **Aggregation of Local Updates**

- The local model updates from all participating edge devices are sent to the central coordinator.
- The coordinator aggregates these updates using a chosen aggregation strategy, such as:
  - FedAvg: Computes a weighted average of model parameters.
  - FedProx: A modification of FedAvg that stabilizes training when devices have non-IID data.
  - Momentum-Based Aggregation: Assigns higher importance to recent updates for faster convergence.
- The aggregated model is refined to improve accuracy and robustness.

4. **Global Model Update and Redistribution**
   - The newly aggregated global model is distributed back to the edge devices.
   - Each device replaces its local model with the updated global model.

5. **Iterative Training and Convergence**
   - Steps 2–4 are repeated for multiple rounds until the model achieves satisfactory performance.
   - Model convergence is assessed based on metrics such as validation accuracy, loss reduction, and anomaly detection performance.
   - The final global model is deployed for real-time PdM.

7.3.3 Additional Considerations for Implementation

- **Optimizing Model Updates to Reduce Communication Costs**
  - Since frequent model updates can overload networks, techniques such as the following are used:
    - Gradient compression: Reducing the size of transmitted updates.
    - Sparse updates: Sending only significant parameter changes.
    - Asynchronous updates: Allowing edge devices to send updates at different intervals.
- **Handling Data and Device Heterogeneity**
  - To ensure model robustness despite differences in data distributions and device capabilities, the following techniques are implemented:
    - Personalized Federated Learning: Fine-tuning global models on individual devices to account for their unique data patterns.
    - Adaptive learning rates: Adjusting the contribution of devices with high- or low-quality data.
- **Security Enhancements**
  - Since attackers could tamper with model updates, the following techniques are applied:
    - Secure Aggregation: Encrypting updates to prevent adversarial model poisoning.
    - Differential Privacy: Adding noise to updates to enhance data confidentiality.

7.4 Algorithm Aggregation Strategies in Federated Learning

In FML for predictive maintenance, the aggregation strategy plays a crucial role in combining local model updates from multiple edge devices into a robust global model. The choice of aggregation algorithm directly impacts model accuracy, convergence speed, communication efficiency, and the ability to handle data heterogeneity.

This section elaborates on key aggregation strategies, their working principles, advantages, and associated challenges.

7.4.1 Federated Averaging

Concept: FedAvg is the most widely used aggregation strategy in FML. Instead of transmitting updates after every local training step, edge devices perform multiple local updates and then send only the final model parameters to the central server. The server computes a weighted average of these local models based on the number of data samples at each edge device:

where the parameters are:

- Global model parameters
- Local model parameters from device
- Number of training samples on device
- Total number of training samples across all devices
- Number of participating devices

Advantages:

- Simple and computationally efficient—requires only weighted averaging of local models.

- Reduces communication frequency by allowing multiple local updates before aggregation.
- Performs well in homogeneous environments where data distribution is IID (identically and independently distributed).

Challenges:

- Non-IID data degrades performance—if edge devices have vastly different data distributions, the global model may converge poorly.
- Devices with small datasets contribute less—FedAvg prioritizes devices with larger datasets, potentially neglecting smaller but important datasets.
- Straggler problem—slower edge devices may delay global updates, requiring asynchronous solutions.

7.4.2 Federated Stochastic Variance Reduced Gradient (FedSVRG)

Concept: FedSVRG is designed to improve the stability of gradient updates in highly non-IID settings. It introduces a variance-reducing estimator that helps balance global model updates and local variations. Instead of relying solely on the latest local updates, FedSVRG maintains a historical gradient estimate to stabilize the training process.

where the parameters are:

- pdated gradient
- Previous gradient estimate
- Local model gradient update
- Gradient of the global model
- Average of previous gradient estimates

- Learning rate

Advantages:

- Improves convergence speed—reduces gradient variance, leading to more stable training.
- More efficient than FedAvg in non-IID settings—handles varying local data distributions better.
- Reduces communication overhead—requires fewer updates to achieve a well-trained model.

Challenges:

- Increased computational burden—each device needs additional memory and processing for variance reduction.
- Requires fine-tuning of hyperparameters—choosing appropriate step sizes and learning rates is complex.

7.4.3 Federated Proximal (FedProx)

Concept: FedProx extends FedAvg by adding a proximal term that regularizes local updates, preventing drastic changes in model parameters. This helps stabilize training when there is heterogeneity in edge device capabilities or data distributions.

The FedProx optimization function includes a proximal term:

where the parameters are:

- Regularized objective function
- Local loss function on device
- Current model parameters
- Previous global model parameters

- Proximal term weight (controls how much local updates deviate from the global model)

Advantages:

- Handles heterogeneous edge devices—adapts to different computational and storage constraints.
- Prevents extreme deviations in local models—reduces the risk of inconsistent updates.
- Stabilizes model training in non-IID environments—regularization ensures that local updates do not diverge too much.

Challenges:

- Computationally expensive—each local update requires additional regularization computations.
- Slower convergence—proximal constraints can make training more conservative.

7.4.4 Federated Ensemble Learning

Concept: Unlike FedAvg, which merges model parameters, Federated Ensemble Learning allows each edge device to train a model independently. Instead of aggregating weights, the final decision is made using ensemble techniques such as:

- **Majority Voting**—Each local model makes a prediction, and the most common prediction is chosen.
- **Weighted Averaging**—Local models with higher accuracy receive greater weight in final predictions.
- **Stacking**—Outputs from local models are combined using a meta-learner.

Advantages:

- Highly robust to heterogeneous data distributions—each model specializes in its own data domain.
- Improves fault tolerance—if one model fails, others still contribute to predictions.
- Can combine multiple model architectures—allows different devices to train using different algorithms.

Challenges:

- Requires additional storage and computational power—multiple models must be stored and processed.
- Difficult to manage ensemble diversity—similar local models may not contribute much additional value.
- Higher latency in inference—combining multiple model predictions increases prediction time.

7.4.5 Comparison of Aggregation Strategies

Table 7.1 Different Aggregation Techniques⏎

Strategy	*Best for*	*Key Benefit*	*Major Challenge*
FedAvg	Homogeneous environments	Simple and widely used	Fails with non-IID data
FedSVRG	Non-IID data, unstable gradients	Faster convergence	Extra computation at edge devices
FedProx	Heterogeneous devices and data	Stabilizes updates	Increased computational complexity

Strategy	*Best for*	*Key Benefit*	*Major Challenge*
Federated Ensemble	Highly diverse edge models	Robustness to failure	High storage and computational cost

This structured approach ensures that aggregation strategies are applied effectively based on the specific challenges of federated PdM (Table 7.1).

7.5 System Architecture of Federated Learning for Predictive Maintenance

The system architecture of Federated Learning for PdM consists of multiple components that work together to enable decentralized model training while preserving data privacy. This architecture ensures real-time monitoring, efficient local model updates, and global model aggregation without requiring raw data transmission.

7.5.1 Data Collection and Preprocessing

7.5.1.1 *Data Collection*

PdM relies on sensor-generated data from industrial machines, providing valuable insights into equipment health. These sensors continuously monitor parameters such as:

- **Temperature**—Detecting overheating components.
- **Vibration Levels**—Identifying mechanical imbalances.
- **Pressure**—Monitoring hydraulic or pneumatic systems.

- **Acoustic Signals**—Recognizing abnormal sounds in machinery.
- **Voltage and Current**—Detecting electrical faults.

Each edge device, such as an IoT gateway or industrial controller, collects data from these sensors in real time.

7.5.1.2 *Data Preprocessing*

Raw sensor data often contains noise, missing values, or inconsistencies due to sensor failures, environmental variations, or communication errors. Preprocessing ensures that the data is:

- **Cleaned**—Removing outliers and handling missing values.
- **Standardized**—Normalizing features to a common scale for consistent training.
- **Feature-engineered**—Extracting useful insights such as rolling averages, trend analysis, or frequency domain features using Fourier Transform.
- **Labeled** (if supervised learning is used) —Assigning failure or anomaly labels based on past maintenance records.

Once preprocessed, the structured data is stored locally on edge devices for on-device training, eliminating the need for cloud storage of raw sensor data.

7.5.2 Local Training Mechanism

Each edge device, such as an IIoT node or factory server, independently trains a local predictive maintenance model using its own data. The training mechanism varies based on the following:

7.5.2.1 *Model Type*

- **Deep learning models (Convolutional Neural Networks [CNNs], Long Short-Term Memory networks [LSTMs])** [6,7]—Used for complex time-series analysis and anomaly detection.
- **Traditional ML models (Random Forest and Support Vector Machine** [**SVM**]**)** – Used for structured, tabular data with historical failure records.

7.5.2.2 *Hardware Capabilities*

- **Powerful edge devices**—Train deep learning models.
- **Low-power IoT nodes**—se lightweight models like decision trees or logistic regression.

7.5.2.3 *Training Frequency*

- **Batch training**—Models train at scheduled intervals (e.g., every 24 hours).
- **Real-time adaptation**—Models update dynamically based on streaming data.

Once training is complete, the edge device does not share raw data but instead transmits only model updates (gradients or weights) to the aggregation server.

7.5.3 Aggregation Server

The aggregation server plays a critical role in combining local model updates from multiple edge devices and redistributing the improved global model. It can be implemented using the following.

7.5.3.1 *Centralized Aggregation Server*

A single cloud-based or on-premise coordinator server collects local updates, aggregates them (using FedAvg, FedProx, or FedSVRG), and sends the new global model back to edge devices.

- **Advantage**: Faster aggregation and simpler implementation.
- **Challenge**: Single point of failure, potential bottlenecks in large-scale deployments.

7.5.3.2 *Blockchain-Based Decentralized Aggregation*

A blockchain ledger records model updates from each edge device in a secure, verifiable manner. Smart contracts automate the aggregation process without requiring a central server.

- **Advantage**: Trustless, tamper-proof, and scalable across multiple industrial facilities.
- **Challenge**: Higher computational overhead for consensus mechanisms.

7.5.4 Federated Learning Training Cycle in Predictive Maintenance

1. **Initialization**—The aggregation server initializes a global predictive maintenance model.
2. **Local Training**—Each edge device trains a model using its own sensor data.
3. **Secure Transmission**—Local model updates (not raw data) are encrypted and sent to the aggregation server.
4. **Aggregation**—The server computes a new global model using FedAvg, FedProx, FedSVRG, or an ensemble strategy.

5. **Redistribution**—The updated global model is sent back to all edge devices for the next training cycle.
6. **Continuous Improvement**—The process repeats until model performance stabilizes.

7.6 Performance Metrics

Evaluating the effectiveness of FML in PdM requires a set of well-defined performance metrics. These metrics help assess the model’s predictive accuracy, communication overhead, and resource efficiency, ensuring its practical implementation in industrial environments.

7.6.1 Model Accuracy

Model accuracy serves as a fundamental benchmark for evaluating the reliability of an FML system in predicting equipment failures. The key aspects include:

- **Precision and Recall**: In PdM, false positives (incorrect failure predictions) and false negatives (missed failures) can have significant consequences. Precision measures how many predicted failures are actual failures, while recall assesses the model’s ability to detect all real failures.
- **F1-Score**: A harmonic mean of precision and recall, providing a balanced evaluation of the model’s ability to make accurate failure predictions.
- **Comparison with Centralized Learning**: FML models are often compared against traditional centralized learning to determine trade-offs between decentralized learning efficiency and predictive performance. A successful FML model should perform

comparably to, if not better than, its centralized counterpart while preserving data privacy.

A well-optimized accuracy metric ensures timely maintenance actions, reducing unplanned downtime and improving operational efficiency.

7.6.2 Communication Efficiency

Communication efficiency is crucial in federated learning, where edge devices collaborate by exchanging model updates instead of raw data. The number of communication rounds required for the model to converge directly impacts network bandwidth and latency. Key factors influencing communication efficiency include:

- **Communication Rounds vs. Model Convergence**: The fewer the number of communication rounds needed for model convergence, the lower the bandwidth and computational overhead. Optimizing update frequency ensures a balance between rapid convergence and minimal data exchange.
- **Compression Techniques**: Methods such as quantization and sparsification can reduce the size of model updates, minimizing communication overhead.
- **Asynchronous Training**: Allowing devices to update the global model at different intervals prevents bottlenecks caused by slower nodes (stragglers) and enhances training speed.

Efficient communication ensures real-time responsiveness, making FML viable for large-scale industrial applications where latency and network constraints are critical.

7.6.3 Energy Consumption

Energy consumption is a critical consideration, particularly in edge computing environments with power-constrained devices. This metric assesses the energy required for:

- **Local Model Training**: The computational effort involved in training models on edge devices, which varies based on model complexity and data volume.
- **Model Update Transmission**: The energy cost of sending updates to the central aggregator, which can be reduced using optimized update strategies.
- **Global Aggregation**: The computational expense incurred by the central server in aggregating multiple local updates and distributing the refined model.

Strategies such as *lightweight model architectures, adaptive training schedules, and energy-aware aggregation techniques* help optimize power usage, making federated learning more sustainable for real-world PdM applications.

7.7 Case Studies

FML has been successfully applied across multiple industries to enhance PdM while addressing concerns about data privacy, communication efficiency, and resource constraints. This section explores real-world implementations of FML in PdM across manufacturing, energy, and healthcare sectors.

7.7.1 Manufacturing Industry: Predictive Maintenance of Computer Numerical Control (CNC) Machines Using FML-Based Anomaly Detection

CNC machines are widely used in manufacturing for precision cutting, drilling, and shaping of materials. Any unexpected failure in these machines can lead to costly downtimes, production losses, and increased maintenance expenses.

Challenges in Traditional Predictive Maintenance:

- High variability in machine usage due to different operational conditions.
- Data privacy concerns related to sharing machine usage data across plants or suppliers.
- High communication costs from transmitting large volumes of sensor data to a central server.

FML Solution:

- IoT sensors installed on CNC machines monitor vibration, motor temperature, and tool wear.
- Local edge devices train anomaly detection models using historical sensor data to detect early signs of mechanical degradation.
- Instead of transmitting raw sensor data, only model updates are shared with a central aggregator, preserving data privacy and reducing bandwidth usage.
- The central aggregator refines the global model using FedAvg and redistributes it to all participating machines, improving predictive accuracy.

Impact:

- Reduced downtime through early fault detection and proactive maintenance.
- Improved data security by keeping sensitive operational data local.

- Lower communication costs by minimizing the need for frequent cloud transmissions [8].

7.7.2 Smart Grid Systems: Fault Prediction in Power Transformers with Collaborative Edge-Based Learning

Power transformers play a crucial role in electricity transmission and distribution networks. Failure of these components can lead to large-scale power outages, economic losses, and safety hazards [9–11].

Challenges in Traditional Fault Prediction:

- Decentralized grid infrastructure complicates centralized data collection.
- Heterogeneous transformer specifications and operating conditions lead to non-uniform data distributions.
- Latency and bandwidth constraints make real-time fault detection difficult.

FML Solution:

- Smart meters and edge sensors in transformers monitor voltage fluctuations, temperature, and load variations.
- Each transformer’s edge device locally trains a fault classification model using historical failure patterns and real-time sensor data.
- Only model gradients or weight updates are sent to a regional energy control center for aggregation, reducing network congestion.
- The aggregated global model is redistributed to all connected transformers to improve predictive

capabilities.

Impact:

- Faster fault detection through real-time edge-based learning.
- Enhanced grid stability by improving transformer lifespan and performance.
- Reduced operational costs through optimized maintenance scheduling and lower bandwidth usage.

7.7.3 Healthcare Equipment Maintenance: Remote Monitoring of Medical Devices to Preemptively Identify Operational Failures

Medical devices such as Magnetic Resonance Imaging (MRI) scanners, ventilators, and infusion pumps are critical in healthcare facilities. Equipment failures can disrupt patient care and lead to serious medical risks.

Challenges in Traditional Healthcare Equipment Maintenance:

- Strict data privacy regulations (e.g., HIPAA and GDPR) restrict data sharing across healthcare facilities.
- High downtime costs associated with delays in detecting faults in medical devices.
- Manufacturer-specific maintenance models make it difficult to develop a generalized predictive system.

FML Solution:

- Edge devices attached to medical equipment continuously monitor operational parameters such as temperature, pressure, and mechanical movements.

- Local servers at hospitals train predictive maintenance models to detect early signs of failure.
- Instead of transmitting raw patient-related data, only encrypted model updates are shared with a central aggregator hosted by medical device manufacturers.
- Aggregated models enhance failure prediction accuracy while ensuring compliance with privacy regulations.

Impact:

- Enhanced equipment reliability through timely predictive maintenance.
- Compliance with privacy laws by avoiding direct sharing of sensitive patient data.
- Cost savings by optimizing preventive maintenance schedules and reducing emergency repairs.

Key Takeaways from Case Studies:

- **Enhanced Data Privacy:** Sensitive operational data remains local, reducing exposure risks.
- **Scalable Predictive Maintenance:** Edge-based training supports large-scale deployment across industries.
- **Reduced Communication Overhead:** Bandwidth-efficient learning minimizes network congestion and operational costs.
- **Adaptability to Heterogeneous Environments:** FML models adjust to diverse device behaviors, ensuring robust predictive performance.

By leveraging FML, industries can achieve cost-effective, secure, and intelligent predictive maintenance, driving greater operational efficiency and reliability.

7.8 Future Research Directions

As FML continues to evolve, new technologies and methodologies are emerging to enhance its effectiveness in PdM. This section explores three key future research directions: blockchain-enabled federated learning, transfer learning for FML, and integration with 6G networks.

7.8.1 Blockchain-Enabled Federated Learning

Federated learning relies on a central aggregator to combine model updates from multiple edge devices, posing potential security and trust challenges. Blockchain technology can establish decentralized trust mechanisms for secure and transparent model aggregation.

Challenges in Traditional FML Model Aggregation:

- Trust and security risks due to reliance on a single aggregator
- Vulnerability to model poisoning attacks and data integrity issues
- Lack of transparency and verifiability in model updates

Blockchain-Enabled Solution:

- Decentralized model aggregation eliminates dependence on a central server
- Immutable records of model updates ensure data integrity and traceability
- Incentive mechanisms using cryptographic tokens encourage honest participation

Potential Impact:

- Enhanced security and resilience against adversarial attacks
- Trustless collaboration among multiple stakeholders
- Improved transparency and auditability in model training

7.8.2 Transfer Learning for FML

Industrial environments often contain diverse machinery with different operational characteristics, making it difficult to train a generalizable PdM model. Transfer learning can enhance model adaptability across different equipment types while reducing the need for extensive labeled data.

Challenges in Current FML Implementations:

- Heterogeneous data distributions across different machines
- Limited labeled failure data for certain equipment
- High computational costs of training separate models for each machine type

Transfer Learning Approach:

- Pretrained global models serve as a starting point for new machine types
- Fine-tuning specific model layers to adapt to new environments
- Few-shot learning to reduce data requirements for new machines

Potential Impact:

- Improved generalization across different industrial settings
- Faster deployment of predictive maintenance models
- Reduction in computational and data collection costs

7.8.3 Integration with 6G Networks

The advent of 6G networks will enable ultra-low latency communication and enhanced edge computing capabilities, making real-time federated model training feasible for PdM applications.

Challenges with Current Network Infrastructure (4G/5G):

- Latency issues in large-scale federated learning deployments
- High communication overhead leading to network congestion
- Limited support for real-time processing in edge environments

6G-Enabled Enhancements:

- Sub-millisecond latency allows for near-instantaneous model updates
- Advanced edge computing capabilities support decentralized processing
- AI-driven network optimizations reduce energy consumption in edge devices

Potential Impact:

- Real-time failure prediction for industrial equipment

- Scalable federated learning deployments across large IoT networks
- Cost reduction through optimized communication protocols

As these research directions progress, they will further strengthen the potential of federated learning in PdM, driving innovation and enhancing industrial efficiency.

7.9 Conclusion

FML presents a transformative approach to PdM by decentralizing model training while preserving data privacy. Industries can leverage FML to implement scalable, secure, and efficient maintenance solutions without exposing sensitive data. By optimizing communication efficiency, enhancing model robustness, and integrating emerging technologies like blockchain and 6G, future research can further refine FML applications, driving innovation across various sectors. The continuous evolution of federated learning will pave the way for more intelligent, self-sustaining maintenance systems, ensuring reliability, cost-effectiveness, and operational efficiency.

References

1. Kairouz, P., et al. (2021). Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning* 14(1–2), 1–210.⏎
2. Li, T., et al. (2020). Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine,* (3), 50–60.⏎

3. Smith, V., Chiang, C. K., Sanjabi, M., & Talwalkar, A. S. (2017). Federated multi-task learning. *Advances in Neural Information Processing Systems*, 30.⏎

4. Yang, Q., et al. (2019). Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology*, 10(2), 1–19.

5. McMahan, B., et al. (2017). *Communication-Efficient Learning of Deep Networks from Decentralized Data*. AISTATS.⏎

6. Gupta, P., Anand, A., Agarwal, P., & McArdle, G. (2024). Neural network inspired efficient scalable task scheduling for cloud infrastructure. *Internet of Things and Cyber-Physical Systems,* 4, 268–279.⏎

7. HS, M., & Gupta, P. (2024). Federated learning inspired Antlion based orchestration for Edge computing environment. *PLoS One*, 19(6), Art. e0304067.⏎

8. Mustapha, S. D. S., & Gupta, P. (2024). DBSCAN inspired task scheduling algorithm for cloud infrastructure. *Internet of Things and Cyber-Physical Systems,* 4, 32–39.⏎

9. Gupta, P., Rawat, P. S., kumar Saini, D., Vidyarthi, A., & Alharbi, M. (2023). Neural network inspired differential evolution based task scheduling for cloud infrastructure. *Alexandria Engineering Journal,* 73, 217–230.⏎

10. Madhusudhan, H. S., Gupta, P., Saini, D. K., & Tan, Z. (2023). Dynamic virtual machine allocation in cloud computing using elephant herd optimization scheme. *Journal of Circuits, Systems and Computers*, 32(11), 2350188.

11. Rawat, P. S., Gaur, S., Barthwal, V., Gupta, P., Ghosh, D., Gupta, D., & Rodrigues, J. J. C. (2025). Efficient virtual machine placement in cloud computing environment

using BSO-ANN based hybrid technique. *Alexandria Engineering Journal,* 110, 145–152.↩

Chapter 8

Artificial Intelligence and Machine Learning-Based Predictive Maintenance in Fog and Edge Computing Environment

Sahil Raj, Pradeep Kumar Rawat, and Anirudh Negi

DOI: 10.1201/9781003610168-8

8.1 Introduction

Since software, hardware, and wireless communication technologies have advanced rapidly in recent decades, the number of Internet of Things (IoT) devices has skyrocketed, enabling data analysis and measurement from physical to cyber work [1]. The process of moving processing, data storage, and application software to cloud data centers has already been seen. An application can thus request access to shared processing and storage resources. The cloud computing approach offers a number of benefits, including high scalability, availability, accessibility, and inexpensive deployment costs [2]. IoT devices, which are widely utilized in manufacturing, smart cities, smart homes, autonomous driving, and the industrial Internet of Things, typically have tiny memory and low processing power. Large volumes of data are continuously generated by IoT devices, which must be gathered and examined. Sending so much data to servers in the cloud raises the cost of communication and network capacity, delays system response, and compromises data privacy [3]. Fog or edge computing has recently been offered as a solution to this problem, where data processing and computation take place in locations that are close to the actual location from whence the data originates [3, 4].

Industry 4.0 is defined by the convergence of edge–fog–cloud computing, which is driving the next wave of digital transformation, and artificial intelligence (AI), which includes augmented intelligence, machine learning (ML), and deep learning (DL) [5]. To fully realize the potential of intelligent IoT, it is necessary to alleviate severe data fragmentation through cross-company collaboration (such as multi-party computation, pooled analysis, data sharing, and data swapping among a network of collaborators/organizations) [6, 7].

IoT, AI, and digital change are all powered by data. However, when data is available and used by several groups, it is healthy [8]. But at the moment, IoT data is widely dispersed throughout silos that are owned or controlled by several parties, each of whom can only see a portion of the picture due to a variety of impediments. Examples include disparate technologies and data standards, sovereignty over data security, privacy, distrust, data governance policies among businesses, intellectual property rights, administrative roadblocks, legal and compliance issues, and stifling technologically driven breakthroughs and discovery [7, 8].

In fact, traditional ML/DL-based solutions (such as anomaly detection and failure prediction) are usually not directly applicable to decentralized applications in the IIoT with dispersed edge devices. Two factors are at play here:

1. Privacy concerns: Integrating IIoT data across several parties usually yields tremendous ML capabilities; nevertheless, businesses/factories and the associated edge devices are hesitant to share their safety-sensitive acquired data with one another, leading to data islands and silos.
2. Absence of high-quality data: obtaining high-quality, labeled data is typically costly and challenging, which significantly impairs the model's capacity for generalization and robustness [9].

In terms of accuracy and dependability, a few strategies have been used recently to address these problems (such as knowledge transfer methods, transfer learning approaches, and synthetic data generation); however, they fall short of solutions that aim to increase the size of the training dataset. In order to generate their own predictions, ML algorithms look for patterns in the data and acquire knowledge from them. To put it briefly, ML models and algorithms pick up knowledge through experience. Engineers create computer programs and provide them with instructions that allow them to transform incoming data into the desired output [10].

In contrast, ML builds the software to learn gradually and with little to no human interaction. Researchers from various domains have found ML appealing due to its extraordinary performance, great promise in regression and classification problems, and ability to apply supervised as well as unsupervised learning approaches [11]. Subsequent research demonstrated the range of ML applications that are evident in the sector, including:

- Product recommendations and e-commerce
- Speech, image, and pattern recognition
- Analytics of user behavior and context-aware smartphone apps
- Healthcare services
- Transportation and traffic forecasting
- IoT and smart cities
- Threat intelligence and cybersecurity
- Sentiment analysis and natural language processing
- Sustainable agriculture and industrial applications

The method of anticipating equipment breakdowns and performing maintenance to stop them before they happen is known as predictive maintenance (PdM). In the age of Industry 4.0 and smart production, this approach is crucial. PdM can maximize equipment longevity by lowering the probability of unplanned failures and minimizing the amount of needless repairs by utilizing sensor readings, parameters for the process, and other operational features. By maximizing equipment uptime and lowering maintenance costs, this approach boosts productivity. Since assembly line production is used in the majority of industrial operations, a machine failure has a cascading effect [9, 10]. As a result, avoiding assembly line failure sites is crucial. A significant amount of pertinent data must be gathered and analyzed in a fair length of time in order to produce accurate and ideal forecasts [10].

Intelligent edge computing solutions for distributed processing of data are helpful in addressing the related issues. Analyzing data on devices at the edge or local servers near the data generating source is known as edge computing. This minimizes the volume of data transferred to a centralized server and enables data processing at the source. Real-time data analysis by edge devices greatly lowers system load by sending just the information that is required to the central server [11]. By enabling AI training on dispersed IoT devices without requiring data sharing, federated learning (FL) has become a popular distributed, collaborative AI technique that can support a wide range of intelligent IoT applications [12].

8.2 Pertinent Literature

8.2.1 Federated Learning

Federated learning refers to a method where a central server and several local clients work together to learn the global framework in a scenario where data is decentralized. Local clients can be smartphones, IoT devices, and other devices [13]. FL makes learning possible without worrying about data leaks or privacy concerns. FL opens up new research directions for aim [14].

FL is a cutting-edge training technique for creating customized models without jeopardizing user privacy. With the advent of AI chipsets, client devices' computational capabilities have grown more potent. Similarly, AI model training shifts onto the central server to the terminal devices [15]. By effectively training the model using terminal instrument processing capabilities, FL is a confidential protection technique that keeps private information hidden from view during data transmission. The majority of DL/ML models rely on massive data for learning, which may raise security and privacy concerns. FL learns locally, keeps the data on each person's local client instead of a centralized server, and then uses the parameters of the modified model to update a model on the centralized server. Preserving privacy while avoiding data conflicts is the aim. A distributed ML configuration has several clients coordinating with one or more centralized servers [16]. By solely obtaining updates from local models, FL can lower costs when network traffic and expenses rise during the training phase. FL's fundamental tenet is that local updates from the central server alter the global model [17].

FL and distributed learning are closely related. Distributed compute and storage comprise a traditional distributed system. Distributed computation and the initial FL of models update for an Android client are rather comparable [18]. The most recent distributed ML research also pays significant attention to private-preserving distributed systems, even though FL placed a lot of emphasis on protecting one's privacy. FedAvg is the rudimentary FL framework. However, it might be able to handle certain light non-independent and identically distributed (non-IID) data. It continues to struggle with structural heterogeneity and significant communication overhead. Recent studies concentrate on algorithmic optimization to increase accuracy and efficiency as well as participant privacy to improve data protection.

Although FL differs greatly from common huge data privacy protection techniques like privacy differential and k-order inconspicuousness, its primary feature is its protection of user privacy. By transmitting encoded administered limits, FL mainly protects handler confidentiality by preventing intruders from accessing basic data [19]. This guarantees that FL won't violate GDPR and other regulations or jeopardize handler confidentiality at the data level. Because users have complete control over local data, data owners' anonymity is emphasized. This is the essential component of FL that guarantees confidentiality. There are two different kinds of privacy protection mechanisms in an FL environment. Commonly employed encryption techniques include safe aggregation and homomorphic encryption. Another popular approach is to include noise of variance confidentially in the method limitations. Google's proposed federated learning combines safe convergence and differential secrecy to protect privacy. To achieve confidentiality fortification, other research solely uses homomorphic encoding fortification settings [20].

In the FL paradigm, data from clients are processed collaboratively to train a model mFL in such a manner that any client cici does not expose its data didi to others. In addition, the accuracy of mFL, denoted as A(mFL), should be very close to the accuracy of a DL network mDL, denoted as A(mDL) [21]. Formally, let δ be a non-negative real number; then, the FL algorithm has δ-accuracy loss if:

FL assists in the reduction of the following:

- Potential of unauthorized data access, since data transmission over the network is not done.
- Cost and time of information transfer by reduction of the data that is transmitted over the network.

- Central computational cluster and central storage requirements.
- Network traffic.

Table 8.1 presents the basics of FL [22].

Table 8.1 Basics of FL⏎

Head	*Description*
Aim	Processing various data on heterogeneous data sources
Datasets	Heterogeneous data sets
Nodes	Often low-power devices, such as smartphones and IoT devices, with limited computing resources connected by less reliable networks, which results in their failures or dropping out of computation

8.2.2 Predictive Maintenance

A new preventative maintenance technique called PdM extends equipment life and ensures sustainable operational management, which enhances industrial process performance and efficiency. As a result, there is less downtime, fewer needless line pauses, and cheaper repair expenses [23]. Maintaining industrial equipment in the best possible shape and with the most efficiency is crucial. Techniques are required to guarantee the equipment's effectiveness by reducing machine downtime and averting problems before they happen. Given that PdM relies on data gathered from numerous sensors, it needs to handle a number of concerns, including privacy, operations, and development costs [24].

Four categories of maintenance can be distinguished. When parts fail, the simplest step is to simply replace them. The second option, which has the benefit of improving the overall time efficiency of scheduled maintenance, is the periodic replacement of each part according to the failure history. The third strategy is proactive maintenance, which entails getting rid of broken components to increase the facility's effectiveness. The last technique examines equipment data to try to find and fix anomalies before they cause failure [25].

Digitization and transformation of predictive processes provide for a greater knowledge of the underlying processes, as well as more accurate and justified conclusions that rely less on intuition. Furthermore, knowledge gained from linked data sources and advanced analytics enables the implementation of new maintenance methods, improved work and inventory planning, increased production efficiency, and higher safety levels [26]. The power business has seen significant modifications in recent years, spurred by new energy sources, climate change, and environmental concerns. This sector can thus benefit from current ways to keeping equipment in good operating order, avoiding environmental pollution, and preventing breakdowns that could have major ecological effects. Reference [27] discusses maintenance procedures and tactics for preventing breakdowns and lowering the risk of such issues in the power industry [27].

PdM is a very complex operation; for a real-time view of the condition of health and reliability of industrial machinery, data must be collected from the system's various sensors [27]. The maintenance strategy consists of four phases:

1. Data collection from various system sensors
2. Data preprocessing
3. Fault detection and prognosis
4. Decision-making.

AI and ML algorithms can be used to collect and evaluate data from the physical environment. Author describe numerous strategies for PdM:

- The physical model technique assesses component degradation using a system's physics or mathematical model. The accuracy of this approach is based on the model, which is validated using statistical approaches.
- Knowledge-based strategy, which depends on prior knowledge or experience in the system to lessen its complexity. Expert systems and fuzzy logic fall under this category.
- Data-driven method, which uses computer capacity and a big amount of data. This model is divided into three types: statistical, stochastic, and ML models.
- Digital twin approach, which combines data and models and creates a link between the physical world and the digital ones [28].

A basic method to defect prediction is to develop physics-based models that include a physical description of the machine deterioration process. Nowadays, even if data-based methods are primarily used, the selection of physics-based models may be more appropriate, especially in some sectors [29]. This mathematical technique correlates the phenomenon of wear with the usable life of components. Among the variables examined in the creation of the physical-mathematical model, several physical values characterize the thermal, mechanical, chemical, and electrical properties of the analyzed component.

Domain experts are also depended on to construct knowledge-based models, which try to imitate the experts' abilities and behavior. As a result, once knowledge has been formalized, it may be reproduced and applied automatically. Expert systems are programs that combine experts' knowledge in a certain field and inference processes to mimic thought while providing support and practical solutions. Rule-based systems and fuzzy logic are two of the most frequent techniques to implementing this type of model. Rule-based systems have the advantage of being simple to design and interpret, but they can perform badly, particularly when complex situations or a large number of rules are required.

Statistical and stochastic approaches allow us to cope with complicated systems whose evolution over time is difficult to anticipate [30]. As we will see in this part, the application of statistical approaches for the prediction, estimation, and optimization of the probability of survival and the average life span of a system can be beneficial in some specific circumstances connected to the operation of mechanical components.

AI-based PdM has six major components: data preprocessing, AI algorithms, decision-making modules, communication and integration, user interface, and reporting.

Sensors are the primary data collectors in a PdM system. These specialized gadgets are strategically positioned on equipment and machinery to continually monitor a variety of characteristics, including temperature, pressure, vibration, and others [31]. Sensor data provides real-time insights into equipment health and serves as the foundation for PdM analysis. Raw sensor data is frequently noisy and inconsistent.

Data preparation is the first stage in preparing data for analysis [32]. It entails data cleansing, normalization, and missing data management. High-quality data is required for accurate PdM modeling. AI methods, such as ML and DL approaches, serve as the brain of the PdM system.

The algorithms examine the data to determine the most critical characteristics associated with potential failures. They use previous data to forecast equipment failures, abnormalities, and Remaining Useful Life (RUL).

AI algorithms generate insights and forecasts, which are processed by decision-making modules. These modules are responsible for identifying when maintenance is required. They can prescribe preventative and corrective maintenance jobs, schedule maintenance, and provide notifications to maintenance personnel as needed [22].

Communication and integration ensure that the system's findings are successfully converted into action. This component entails interacting with a variety of stakeholders, including maintenance workers and management. Furthermore, connection with corporate systems like Enterprise Resource Planning (ERP) and asset management software helps to align PdM with overall organizational goals [32].

To make these insights available to maintenance personnel and decision-makers, user interfaces and reporting tools are required. These tools make it easier for users to grasp complicated data patterns and make informed decisions by incorporating data visualization, dashboard, and reporting capabilities. Data visualization tools and dashboards help maintenance teams and decision-makers understand data insights and forecasts. Visual aids help you understand complex data patterns and make informed decisions [22].

8.2.3 Federated Learning for Predictive Maintenance

Equipment failure can have a negative impact on productivity in production. As a result, it is critical to develop a systematic PdM model for process equipment in order to maintain production schedules and quality while also reducing energy waste and accidents. For this aim, we detect sensor data collected from the facility, compare it with the current normal state, and develop a predictive model for the facility's future state. FL for PdM is the integration of distributed ML with industrial applications to enhance equipment reliability and efficiency [33]. FL allows multiple devices or organizations to train ML models collaboratively without sharing sensitive data, which makes it ideal for PdM scenarios where data privacy and security are critical. By leveraging the decentralized data available in diverse sources, FL allows the creation of robust models that can identify patterns and predict equipment failures across various environments and operating conditions. This approach reduces downtime, optimizes schedules for maintenance, and minimizes costs while preserving proprietary data confidentiality [34]. The synergy of FL and PdM represents an industry-changing advancement in AI for industries such as manufacturing, transportation, and energy.

8.3 Related Work

Distributed machine learning methods are increasingly being implemented using edge and fog computing, particularly in contexts with limited resources. Four potential strategies for allocating the workload among the edge, fog, and cloud levels are covered by Ucar [35]. This article also highlights developments and problems for implementation in the areas of communication, security, privacy, machine learning, and hardware [35]. Generally speaking, there are two subsections in this area. The majority of relevant research on FL and optimization strategies, distribution methods, and hierarchical FL are included in ML at the edge, fog, and cloud levels. The majority of associated works that use FL in Predictive Maintenance (PM) application and distributed ML for collaboration PM scenarios are included in the second category, DML for collaborative PM.

8.3.1 Cloud, Edge, and Fog-Level Machine Learning

To facilitate shared PM, centralized cloud computing is the best option for data integration. However, manufacturing companies will not be content with corporations sharing production data because of data privacy and industrial competition. An organization may occasionally be hesitant to centralize the asset failure data gathered from many production sites due to legal concerns. Additionally, it is very expensive and unaffordable to centralize any raw data in a cloud [27]. As a result, edge and fog computing present a viable way for the manufacturing sectors to connect disparate data islands in order to improve models while safeguarding their business intelligence [32].

Because every algorithm has a unique communication pattern, it is difficult to design a system that allows ML to be distributed efficiently. DML is a developing system with a variety of solutions that vary in terms of performance, efficiency, algorithm, and architecture.

The topic of learning model parameters using data spread over several edge nodes without transferring raw data to a centralized place, such as cloud or fog, has been examined by Wang et al. [36] To train the models, which included SVM models, convolutional neural networks (CNNs), K-means, and linear regression, they developed a method based on distributed gradient descent. They demonstrated that non-IID data is incorporated into the rate of convergence to the suggested gradient-descent-based technique for distributed learning [29].

Numerous reviews have been produced in this field as a result of the FL research's explosive growth, which provides a thorough overview of FL and analyzes it from five perspectives—systems heterogeneity, communication architecture, ML model, privacy method, and data partitioning— and discusses the main communication issues with FL applications in edge devices and the IOT [28].

For FL in distributed optimization, there are a number of algorithms available. The majority of these algorithms, such as FedAvg, FedProx, CO-OP, and federated stochastic variance reduced gradient, have been assessed and contrasted. In order for the FedAvg algorithm to function, the training task is executed on the edge devices, which share an overall model—which is an average of all the parameters—with the central server.

8.3.2 Collaborative DML PM

According to research, ML algorithms for PM application can be implemented by utilizing fog computing, which has greater computational capacity than edge devices.

FL makes it possible for PM to collaborate at the edge level on cross-company data for various production locations or even data that is dispersed throughout several enterprises. The goal of this approach is to provide failure pattern information on a single asset without disclosing the raw information, which can be regarded as private business information [37].

Some edge devices might not have the processing power to train the global model in time, which is one of the problems with employing FL in the PM application. These clients have the potential to hinder other edge devices' effective collaborative learning of the pattern of failure from one another, delay model aggregation, and even disconnection during a training iteration. The Split Pred framework for collaborative PM, which offers a cross-device FL to conduct dependable model training at edge devices, was developed as a solution to this problem [38].

A real-time edge computing defect detection system was suggested. Based on an Long Short-Term Memory (LSTM) recurrent neural network running on the back end, they employed a two-layer architecture with a real-time fault detector using single-board computers. A system architecture for edge-based PM applications for IoT-based manufacturing was put forth by Yang et al. [39]. They demonstrated how distributed learning in edge computing offers certain benefits in terms of bandwidth optimization and delay response for edge control. To illustrate the operation of edge, fog, and cloud-based resources, a mechanism for collaboration among edge, fog, and cloud computing is examined [39].

An empirical investigation on predicting failures in the production line based on FL was given by Ning Ge et al.

[40]. For the horizontal FL scenario, they have created a federated SVM method; for the vertical FL scenario, they have created a federated random forest algorithm. Additionally, they have examined FL's efficacy in contrast to centralized learning. According to their findings, the centralized algorithm for failure maintenance and prediction can be replaced with the distributed FL algorithm [40].

Table 8.2 Federated ML concepts and applications

Federated ML concepts and applications	Stochastic method with variance reduction for solving the problem on federated learning [23]
	Challenges of non-IID data to model training on horizontal and vertical FL [29]
	Horizontal FL, vertical FL, and federated transfer learning [31]
	Analyzing FL regarding data partitioning, privacy, model, and communication [39]
	Overview of FL, technologies, protocols and applications [41]

8.4 Conclusion

A rapidly expanding field of study with many benefits and challenges, particularly in predictive maintenance (PM) applications, is distributed ML algorithms across edge devices and their collaboration with fog and cloud [42]. It has been demonstrated that using federated ML techniques enhances communication effectiveness and system reaction time for real-time applications, in addition to improving the privacy and security of data from edge devices. During a collaborative PM application, two distributed models, Federated Support Vector Machine (FedSVM) and Federated Long Short-Term Memory (FedLSTM), were developed in this research to allow local edge devices within an FL algorithm to jointly train a global model at the cloud level [43]. Two distinct communication topologies were used to examine the FedSVM model, and its accuracy and convergence time were evaluated. When it came to forecasting while maintenance should be performed, FedSVM was shown to be incredibly quick during training and appropriate for distributed web applications [36]. FedLSTM with the random connection among the neurons has been developed and investigated based on two distinct communication topologies, with both asynchronous and synchronous algorithms for precise RUL prediction on distributed systems. The model accuracy and convergence time have been evaluated using the Commercial Modular Aero-Propulsion System Simulation (CMAPSS) dataset. FedSVM and FedLSTM results are comparable to centralized algorithms, and they also improve communication effectiveness and system reaction time for real-time applications, in addition to improving edge device privacy and security, according to a comparison with state-of-the-art research on centralized prediction of RUL with CMAPSS [44]. The FedSVM model has also been applied to the Modified National Institute of Standards and Technology (MNIST) dataset's digit classification, demonstrating the generality of the aggregation technique and its compatibility with various learning algorithms [41].

Future research will focus on improving the model aggregation process to handle heterogeneous hardware at the edge level, non-IID data, and Simpson's paradox, which are common in PM applications because of the various anomalies that occur at edge devices [41, 44].

References

1. Awotunde, J. B., Jimoh, R. G., Ogundokun, R. O., Misra, S., & Abikoye, O. C. (2022). Big data analytics of IoT-based cloud system framework: Smart healthcare monitoring systems. In *Artificial intelligence for cloud and edge computing* (pp. 181–208). Cham: Springer International Publishing.⏎
2. Zhang, C., Hu, X., Xie, Y., Gong, M., & Yu, B. (2020). A privacy-preserving multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *Frontiers in Neurorobotics, 13*, 112.⏎
3. Xie, Y., Wang, H., Yu, B., & Zhang, C. (2020). Secure collaborative few-shot learning. *Knowledge-Based Systems, 203*, 106157.⏎
4. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., ... & Seth, K. (2017, October). Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1175–1191).⏎
5. Yang, Q., Liu, Y., Chen, T., & Tong, Y. (2019). Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *10*(2), 1–19.⏎
6. Nguyen, D. C., Ding, M., Pham, Q. V., Pathirana, P. N., Le, L. B., Seneviratne, A., ... & Poor, H. V. (2021). Federated learning meets blockchain in edge computing:

Opportunities and challenges. *IEEE Internet of Things Journal, 8*(16), 12806–12825.⏎

7. Lee, H., & Kim, J. (2021, August). Trends in blockchain and federated learning for data sharing in distributed platforms. In *2021 Twelfth International Conference on Ubiquitous and Future Networks (ICUFN)* (pp. 430–433). IEEE.⏎
8. Li, L., Fan, Y., & Lin, K. Y. (2020, October). A survey on federated learning. In *2020 IEEE 16th International Conference on Control & Automation (ICCA)* (pp. 791–796). IEEE.⏎
9. Yu, S., Chen, X., Zhou, Z., Gong, X., & Wu, D. (2020). When deep reinforcement learning meets federated learning: Intelligent multitimescale resource management for multiaccess edge computing in 5G ultradense network. *IEEE Internet of Things Journal*, *8*(4), 2238–2251.⏎
10. Leroy, D., Coucke, A., Lavril, T., Gisselbrecht, T., & Dureau, J. (2019, May). Federated learning for keyword spotting. In *ICASSP 2019–2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 6341–6345). IEEE.⏎
11. Feng, J., Rong, C., Sun, F., Guo, D., & Li, Y. (2020). PMF: A privacy-preserving human mobility prediction framework via federated learning. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, *4*(1), 1–21.⏎
12. Nguyen, D. C., Ding, M., Pathirana, P. N., Seneviratne, A., Li, J., & Poor, H. V. (2021). Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, *23*(3), 1622–1658.⏎

13. Ahn, J., Lee, Y., Kim, N., Park, C., & Jeong, J. (2023). Federated learning for predictive maintenance and anomaly detection using time series data distribution shifts in manufacturing processes. *Sensors*, *23*(17), 7331. https://doi.org/10.3390/s23177331⏎
14. Fausing Olesen, J., & Shaker, H. R. (2020). Predictive maintenance for pump systems and thermal power plants: State-of-the-art review, trends and challenges. *Sensors*, *20*(8), 2425.⏎
15. Chong, K. E., Ng, K. C., & Goh, G. G. G. (2015, December). Improving Overall Equipment Effectiveness (OEE) through integration of Maintenance Failure Mode and Effect Analysis (maintenance-FMEA) in a semiconductor manufacturer: A case study. In *2015 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)* (pp. 1427–1431). IEEE.⏎
16. Bousdekis, A., Lepenioti, K., Apostolou, D., & Mentzas, G. (2019). Decision making in predictive maintenance: Literature review and research agenda for industry 4.0. *IFAC-PapersOnLine*, *52*(13), 607–612.⏎
17. Compare, M., Baraldi, P., & Zio, E. (2019). Challenges to IoT-enabled predictive maintenance for industry 4.0. *IEEE Internet of Things Journal*, *7*(5), 4585–4597.⏎
18. Servetnyk, M., Fung, C. C., & Han, Z. (2020, December). Unsupervised federated learning for unbalanced data. In *GLOBECOM 2020-2020 IEEE Global Communications Conference* (pp. 1–6). IEEE.⏎
19. Haq, I. U., Anwar, S., & Khan, T. (2023, March). Machine vision based predictive maintenance for machine health monitoring: A comparative analysis. In *2023 International Conference on Robotics and Automation in Industry (ICRAI)* (pp. 1–8). IEEE.⏎

20. Calikus, E., Nowaczyk, S., Sant'Anna, A., & Dikmen, O. (2020). No free lunch but a cheaper supper: A general framework for streaming anomaly detection. *Expert Systems with Applications, 155*, 113453.⏎
21. Marchiningrum, A. U. (2022, December). Digital twin for predictive maintenance of palm oil processing machines. In *2022 6th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)* (pp. 1–6). IEEE.⏎
22. Kholod, I., Yanaki, E., Fomichev, D., Shalugin, E., Novikova, E., Filippov, E., & Nordlund, M. (2021). Open-source federated learning frameworks for IoT: A comparative review and analysis. *Sensors*, *21*(1), 167. https://doi.org/10.3390/s21010167⏎
23. Konečný, J., McMahan, B., & Ramage, D. (2015). Federated optimization: Distributed optimization beyond the datacenter. arXiv preprint arXiv:1511.03575.⏎
24. Lim, W. Y. B., Luong, N. C., Hoang, D. T., Jiao, Y., Liang, Y. C., Yang, Q., ... & Miao, C. (2020). Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials, 22*(3), 2031–2063.⏎
25. Saputra, Y. M., Hoang, D. T., Nguyen, D. N., Dutkiewicz, E., Mueck, M. D., & Srikanteswara, S. (2019, December). Energy demand prediction with federated learning for electric vehicle networks. In *2019 IEEE Global Communications Conference (GLOBECOM)* (pp. 1–6). IEEE.⏎
26. Silva, S., Gutman, B. A., Romero, E., Thompson, P. M., Altmann, A., & Lorenzi, M. (2019, April). Federated learning in distributed medical databases: Meta-analysis of large-scale subcortical brain data. In *2019 IEEE 16th*

International Symposium on Biomedical Imaging (ISBI 2019) (pp. 270–274). IEEE.⏎

27. Lu, Y., Huang, X., Dai, Y., Maharjan, S., & Zhang, Y. (2019). Blockchain and federated learning for privacy-preserved data sharing in industrial IoT. *IEEE Transactions on Industrial Informatics*, *16*(6), 4177–4186.⏎
28. Bonawitz, K. (2019). Towards federated learning at scale: System design. *arXiv* preprint arXiv:1902.01046.⏎
29. Zhu, H., Xu, J., Liu, S., & Jin, Y. (2021). Federated learning on non-IID data: A survey. *Neurocomputing*, *465*, 371–390.⏎
30. Zeng, T., Semiari, O., Mozaffari, M., Chen, M., Saad, W., & Bennis, M. (2020, June). Federated learning in the sky: Joint power allocation and scheduling with UAV swarms. In *ICC 2020–2020 IEEE International Conference on Communications (ICC*) (pp. 1–6). IEEE.⏎
31. Aledhari, M., Razzak, R., Parizi, R. M., & Saeed, F. (2020). Federated learning: A survey on enabling technologies, protocols, and applications. *IEEE Access*, *8*, 140699–140725.⏎
32. Molęda, M., Małysiak-Mrozek, B., Ding, W., Sunderam, V., & Mrozek, D. (2023). From corrective to predictive maintenance—A review of maintenance approaches for the power industry. *Sensors*, *23*(13), 5970. https://doi.org/10.3390/s23135970⏎
33. Tinga, T., & Loendersloot, R. (2019). Physical model-based prognostics and health monitoring to enable predictive maintenance. In *Predictive Maintenance in Dynamic Systems: Advanced Methods, Decision Support Tools and Real-World Applications* (pp. 313–353). Springer, Singapore.⏎

34. Arena, F., Collotta, M., Luca, L., Ruggieri, M., & Termine, F. G. (2022). Predictive maintenance in the automotive sector: A literature review. *Mathematical and Computational Applications*, 27(1), 2. https://doi.org/10.3390/mca27010002⏎

35. Ucar, A., Karakose, M., & Kırımça, N. (2024). Artificial intelligence for predictive maintenance applications: Key components, trustworthiness, and future trends. *Applied Sciences*, *14*(2), 898. https://doi.org/10.3390/app14020898⏎

36. Xu, G., Liu, M., Wang, J., Ma, Y., Wang, J., Li, F., & Shen, W. (2019, August). Data-driven fault diagnostics and prognostics for predictive maintenance: A brief overview. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)* (pp. 103–108). IEEE.⏎

37. Bemani, A., & Björsell, N. (2022). Aggregation strategy on federated machine learning algorithm for collaborative predictive maintenance. *Sensors*, *22*(16), 6252. https://doi.org/10.3390/s22166252⏎

38. Moshawrab, M., Adda, M., Bouzouane, A., Ibrahim, H., & Raad, A. (2023). Reviewing federated machine learning and its use in diseases prediction. *Sensors*, *23*(4), 2112. https://doi.org/10.3390/s23042112⏎

39. Yang, Q., Liu, Y., Chen, T., & Tong, Y. (2019). Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2), 1–19.⏎

40. Ge, N., Li, G., Zhang, L., & Liu, Y. (2022). Failure prediction in production line based on federated learning: an empirical study. *Journal of Intelligent Manufacturing*, *33*(8), 2277–2294.⏎

41. Zhang, C., Xie, Y., Bai, H., Yu, B., Li, W., & Gao, Y. (2021). A survey on federated learning. *Knowledge-Based Systems*, *216*, 106775.⏎

42. Arena, F., Collotta, M., Luca, L., Ruggieri, M., & Termine, F. G. (2021). Predictive maintenance in the automotive sector: A literature review. *Mathematical and Computational Applications*, *27*(1), 2.⏎

43. Achouch, M., Dimitrova, M., Ziane, K., Sattarpanah Karganroudi, S., Dhouib, R., Ibrahim, H., & Adda, M. (2022). On predictive maintenance in industry 4.0: Overview, models, and challenges. *Applied Sciences*, *12*(16), 8081.⏎

44. Metwally, M., Moustafa, H. M., & Hassaan, G. (2020). Diagnosis of rotating machines faults using artificial intelligence based on preprocessing for input data. In *Conference of Open Innovations Association, FRUCT* (No. 26, pp. 572–582). FRUCT Oy.⏎

Chapter 9

Deep Reinforcement Learning-Based Task Scheduling in Edge Computing

Shristi Sonam, Pratik Gupta, and Saanch Sapra

DOI: 10.1201/9781003610168-9

9.1 Introduction

9.1.1 Introduction to Edge Computing and Task Scheduling Challenges

Edge computing has transformed data processing by shifting computational tasks closer to data sources, reducing latency, minimizing bandwidth usage, and improving real-time responsiveness. However, resource constraints such as limited computational power, storage, and energy at edge nodes introduce significant challenges in task scheduling and resource management. Efficiently distributing tasks while considering dynamic workloads, varying network conditions, and heterogeneous resource availability is crucial to maintaining system performance.

Traditional heuristic-based scheduling approaches, such as round-robin, earliest deadline first, or min-min/max-min algorithms, rely on predefined rules and assumptions about workload patterns. These methods often struggle to adapt to unpredictable changes in resource availability and workload fluctuations, leading to suboptimal performance in real-world edge computing scenarios.

9.1.2 Deep Reinforcement Learning (DRL) as a Solution

DRL offers a promising alternative by leveraging continuous learning and real-time decision-making capabilities. DRL integrates deep learning with reinforcement learning (RL), enabling an agent to learn optimal scheduling policies through interaction with the environment. The agent refines its scheduling strategy over time by maximizing a reward function, which considers factors such as latency, energy consumption, task completion time, and overall system throughput.

9.1.3 Key Components of DRL-Based Task Scheduling

1. **Agent** – The decision-making model that learns from past scheduling experiences.
2. **Environment** – The edge computing infrastructure, including edge nodes, tasks, and network conditions.
3. **State Space** – System parameters such as available computational resources, task queue length, energy consumption, and network latency.
4. **Action Space** – Possible scheduling decisions, such as allocating a task to a specific edge node or offloading it to the cloud.

5. **Reward Function** – A feedback mechanism that guides the agent toward optimal scheduling by rewarding low latency, high efficiency, and energy conservation while penalizing suboptimal choices.

9.1.3.1 *DRL Algorithms for Task Scheduling*

Several DRL algorithms have been explored for optimizing edge task scheduling:

- **Deep Q-Networks (DQN)** – Uses Q-learning combined with deep neural networks (DNN) to estimate the best scheduling action.
- **Policy Gradient Methods (e.g., Proximal Policy Optimization [PPO] and Asynchronous Advantage Actor–Critic [A3C])** – Optimize scheduling decisions by directly adjusting policies rather than estimating value functions.
- **Deep Deterministic Policy Gradient (DDPG)** – Suitable for continuous action spaces, useful for fine-grained resource allocation in edge computing [1].

9.1.3.2 *Advantages of DRL-Based Scheduling*

- **Adaptive Decision-Making** – DRL dynamically adjusts to changing workloads and resource availability.
- **Self-Learning and Optimization** – Unlike heuristics, DRL continuously improves performance over time.
- **Reduced Latency and Energy Consumption** – Optimized scheduling decisions lead to more efficient resource utilization.
- **Scalability** – DRL can handle complex edge environments with heterogeneous devices and networks.

9.1.3.3 *Challenges and Future Directions*

While DRL holds great potential for edge computing task scheduling, challenges remain:

- **High Computational Overhead** – Training deep learning models on resource-constrained edge devices is difficult.
- **Exploration-Exploitation Trade-off** – Finding a balance between trying new scheduling strategies and refining known optimal ones.
- **Model Generalization** – Ensuring the DRL model can adapt to different edge computing scenarios without retraining.

Future research can explore *federated learning* (FL) to distribute training across multiple edge nodes, *hybrid scheduling models* combining DRL with heuristics for faster convergence, and *lightweight DRL frameworks* for deployment on edge devices.

9.1.3.4 *Conclusion*

DRL-based task scheduling presents a transformative approach for managing computational workloads in edge computing environments. By leveraging adaptive learning, DRL can outperform traditional heuristic methods, leading to enhanced efficiency, lower latency, and better resource utilization. As advancements in AI and edge computing continue, DRL will play an increasingly crucial role in optimizing real-time task scheduling in distributed systems.

9.2 Challenges in Task Scheduling for Edge Computing

Task scheduling in edge computing is a highly complex process due to the unique characteristics of edge environments, such as decentralized resource management, dynamic workloads, and stringent real-time requirements. Unlike traditional cloud computing, where resources are centralized and scalable, edge computing requires efficient and adaptive task scheduling to optimize performance while addressing several critical challenges. [2–5]

9.2.1 Resource Constraints

9.2.1.1 *Limited Computational Power*

Edge computing devices vary widely in terms of computational capacity, ranging from low-power IoT devices (e.g., sensors and embedded systems) to powerful edge servers. Unlike cloud data centers that have virtually unlimited processing power, edge nodes have finite CPU (Central processing unit), GPU (Graphical processing unit), and memory resources. Inefficient scheduling can lead to computational overload, reducing overall system performance and causing task execution delays.

- Example: In an industrial IoT (IIoT) setup, smart cameras processing real-time video streams for defect detection may struggle to handle multiple high-resolution video feeds due to limited processing power. A poor scheduling decision could cause delays in detecting manufacturing defects.

9.2.1.2 *Storage Limitations*

Edge devices often have constrained storage capacities, making it challenging to store large datasets locally. Unlike cloud environments that provide vast storage resources, edge nodes must manage their data carefully, ensuring that storage limitations do not hinder processing efficiency.

- Solution: Effective task scheduling should include data-aware strategies, where tasks are assigned based on the availability of storage and the ability to process or offload data efficiently.

9.2.1.3 *Power and Energy Constraints*

Many edge nodes rely on battery-powered or low-energy devices, making energy efficiency a key challenge. Continuous high-power computation can drain batteries quickly, reducing the lifespan of edge devices and affecting system reliability.

- Example: In a remote environmental monitoring system, sensors and drones collecting climate data must balance task execution efficiency with power consumption to avoid frequent battery replacements.
- Solution: Energy-aware scheduling algorithms can dynamically adjust task allocation based on the energy levels of each node, ensuring that critical tasks are executed while conserving power.

9.2.2 Dynamic Workloads

9.2.2.1 *Unpredictable Task Arrivals*

Edge computing environments experience fluctuating workloads due to varying user demands and environmental conditions. Task arrival rates can be highly unpredictable, requiring scheduling mechanisms that can dynamically adjust to these changes [6,7].

- Example: A smart traffic management system analyzing real-time traffic patterns will experience varying workloads throughout the day. Traffic surges during rush hours can overwhelm edge nodes if task scheduling is not adaptive.

9.2.2.2 *Adaptive Load Balancing*

Traditional load balancing techniques used in cloud computing are not directly applicable to edge environments due to resource constraints and network variability. Edge-based task scheduling must dynamically balance workloads across available nodes to prevent overloading while ensuring efficient processing.

- Example: In a smart healthcare monitoring system, sensors collecting patient data must distribute computational tasks across multiple edge servers to ensure continuous real-time monitoring without overloading any single node.
- Solution: Reinforcement learning-based scheduling models can continuously learn and adjust load distribution based on real-time workload variations [8].

9.2.2.3 *Task Prioritization and Deadline Constraints*

Many edge computing applications require prioritizing tasks based on urgency and importance. For example, critical tasks (such as emergency alerts in healthcare systems) should be processed immediately, while non-critical tasks (such as background data synchronization) can be scheduled with lower priority.

- Example: In an autonomous vehicle network, vehicle-to-vehicle communication tasks must be executed with minimal delay to avoid collisions, whereas entertainment system updates can be scheduled with lower priority.
- Solution: Deadline-aware scheduling techniques that factor in task priority, execution deadlines, and system load can improve performance in time-sensitive applications.

9.2.3 Latency Sensitivity

9.2.3.1 *Real-Time Processing Requirements*

Edge computing supports applications that require near-instantaneous data processing, such as autonomous driving, augmented reality (AR), and real-time medical diagnostics. Any scheduling delay can lead to system failures or degraded user experience.

- Example: In AR applications like smart glasses, any delay in processing graphical overlays will result in motion lag, reducing user experience and usability.

- Solution: Proximity-aware scheduling algorithms that allocate tasks to the nearest available edge node can significantly reduce latency.

9.2.3.2 *Network Latency Considerations*

Network delays caused by congestion, unreliable wireless links, or variable bandwidth availability can impact task execution time. Unlike cloud computing, where high-speed fiber-optic connections provide stable communication, edge devices often rely on Wi-Fi, 5G, or low-power wide-area networks (LPWANs) with fluctuating reliability.

- Example: In a smart surveillance system, if network congestion delays video analytics processing, the system may fail to detect security threats in real time.
- Solution: Latency-aware scheduling should dynamically adjust task placement based on real-time network conditions, ensuring that tasks are processed on nodes with the lowest communication overhead.

9.2.4 Heterogeneous Infrastructure

9.2.4.1 *Diverse Computational Capabilities*

Edge computing involves a highly heterogeneous environment where edge nodes have varying levels of computational power, architectures, and capabilities. Scheduling tasks efficiently requires an understanding of each node's processing capabilities and compatibility with different applications.

- Example: A smart city infrastructure may include Raspberry Pi-based edge nodes, powerful edge servers, and 5G base stations with AI accelerators. Scheduling must allocate AI-based tasks (such as image recognition) to nodes with GPU support while assigning simple computation tasks to less powerful nodes.

- Solution: Hardware-aware task scheduling techniques can optimize task placement based on the processing power of available edge nodes.

9.2.4.2 *Varied Network Connectivity*

Edge devices operate in a range of connectivity environments, from high-speed 5G networks in urban areas to low-power LPWANs in remote industrial sites. These connectivity variations add complexity to scheduling decisions.

- Example: A remote weather station in a rural area may rely on intermittent satellite communication, making it infeasible to offload tasks frequently. Scheduling should prioritize local task execution instead of cloud offloading.
- Solution: Adaptive network-aware scheduling that considers network conditions and bandwidth availability can enhance system reliability.

9.2.4.3 *Multi-Tier Architecture Complexity*

Edge computing often involves a multi-tier architecture comprising edge nodes, fog computing layers, and cloud integration. Effective task scheduling must decide whether to:

1. Execute tasks locally on the edge (low-latency but resource-constrained).
2. Offload tasks to fog nodes (moderate latency, better resources).
3. Send tasks to the cloud (high latency, unlimited computing power).

    - Example: In a smart factory, machine failure detection should be processed at the edge (for real-time alerts), whereas historical data analytics can be offloaded to the cloud for long-term insights.

- Solution: Hybrid scheduling algorithms can intelligently decide task placement based on workload type, latency requirements, and system constraints.

9.2.4.4 *Conclusion*

Task scheduling in edge computing is a multi-faceted challenge that involves optimizing resource utilization, adapting to fluctuating workloads, minimizing latency, and handling heterogeneous infrastructures. Traditional scheduling approaches struggle to meet the dynamic and decentralized nature of edge environments.

To overcome these challenges, AI-driven approaches such as DRL, federated learning (FL), and hybrid scheduling techniques offer promising solutions. These advanced methods can enhance task scheduling efficiency, improve real-time responsiveness, and ensure optimal resource utilization in edge computing environments. As edge computing continues to evolve, research in intelligent and adaptive scheduling mechanisms will be critical in unlocking its full potential.

9.3 Reinforcement Learning for Task Scheduling

Edge computing introduces significant challenges in task scheduling due to resource constraints, dynamic workloads, and latency-sensitive applications. Traditional heuristic-based scheduling approaches fail to adapt efficiently to changing conditions. DRL offers a promising solution by enabling an intelligent scheduling mechanism that continuously learns and improves its decisions through interaction with the environment [9].

9.3.1 Basics of Reinforcement Learning

9.3.1.1 *Definition and Core Concept*

RL is a branch of machine learning where an agent learns to make optimal decisions by interacting with an environment [10]. Instead of relying on predefined rules, the agent explores different actions, observes their outcomes, and refines its strategy based on a reward-based feedback system.
Key elements of RL include the following:

1. Agent – The decision-maker that interacts with the environment.
2. Environment – The system with which the agent interacts (e.g., edge computing infrastructure).
3. State (S) – The current condition of the environment, such as resource availability and task queue length.
4. Action (A) – A set of possible decisions the agent can make, such as assigning a task to an edge node.
5. Reward (R) – A feedback signal indicating the quality of an action (e.g., a positive reward for reducing latency).
6. Policy (π) – A mapping from states to actions that defines the agent’s decision-making strategy.
7. Exploration vs. Exploitation – The agent must balance exploration (trying new actions to discover better strategies) and exploitation (choosing known optimal actions to maximize rewards).

9.3.1.2 *How RL Works in Task Scheduling*

In an edge computing scenario, an RL agent can be trained to schedule tasks optimally by interacting with the environment. Over time, it learns which scheduling decisions lead to better outcomes, such as lower latency, reduced energy consumption, and efficient resource utilization.

9.3.2 Deep Reinforcement Learning

9.3.2.1 *Limitations of Traditional RL in Task Scheduling*

While RL is effective for decision-making in structured environments, traditional RL methods struggle with high-dimensional state spaces in edge computing. The sheer number of system parameters (e.g., CPU utilization, bandwidth, and task priority) makes it difficult for conventional RL algorithms to handle complex scheduling scenarios.

9.3.2.2 *How DRL Overcomes These Limitations*

DRL extends RL by incorporating DNNs, which allow the agent to efficiently process large state spaces and identify complex patterns in system dynamics. DRL enables the scheduler to learn optimal task allocation policies dynamically without relying on rigid, predefined rules.

9.3.2.3 *Key DRL Algorithms Used for Task Scheduling*

1. Deep Q-Network) – Uses a DNN to approximate the Q-value function, helping the agent choose the best scheduling actions.
2. Policy Gradient Methods (PPO, A3C) – Optimize the agent’s policy directly, making them well-suited for continuous and complex decision spaces.
3. Deep Deterministic Policy Gradient – Handles continuous action spaces, making it ideal for fine-tuned resource allocation in edge computing.

9.3.2.4 *Advantages of DRL for Edge Task Scheduling*

- Handles Complex Environments – Learns optimal scheduling policies even in dynamic and heterogeneous edge networks.
- Adaptability – Continuously improves its decision-making based on real-time system feedback.

- Scalability – Can be deployed across diverse edge nodes, adjusting strategies to match varying resource capabilities.

9.3.3 DRL Framework for Edge Task Scheduling

A DRL-based task scheduling system consists of three key components: state space, action space, and reward function. These components define how the agent perceives the environment, takes scheduling actions, and learns from feedback.

9.3.3.1 *State Space (S)*

The state space represents the current status of the edge computing environment. It includes a variety of system parameters that influence scheduling decisions, such as:

1. CPU Utilization – Measures the processing load on each edge node to avoid overloading.
2. Memory Availability – Tracks available RAM to ensure tasks do not exceed storage limits.
3. Network Bandwidth – Represents data transmission capacity, influencing task offloading decisions.
4. Task Queue Length – Indicates the number of pending tasks at each node, preventing bottlenecks.
5. Energy Levels – Monitors battery status in power-constrained edge devices to optimize scheduling.
6. Latency Requirements – Ensures time-sensitive tasks (e.g., AR applications) are processed within deadline constraints.

    - Example: In a smart healthcare monitoring system, if a patient's Electrocardiogram (ECG) data needs real-time analysis, the state space should include network latency and available computational resources to prioritize immediate processing.

9.3.3.2 *Action Space (A)*

The action space defines the set of possible decisions that the DRL agent can take to schedule tasks efficiently. These include:

1. Task Offloading – Deciding whether to process a task locally at an edge node or offload it to the cloud.
2. Resource Allocation – Assigning computational power, memory, and network bandwidth to specific tasks.
3. Task Prioritization – Ranking tasks based on urgency, deadline, and importance.
4. Load Balancing – Distributing workloads across multiple edge nodes to prevent bottlenecks.
5. Power-Aware Scheduling – Assigning tasks to energy-efficient nodes to maximize battery life.

    - Example: In an IIoT system, a scheduling agent may decide to offload complex AI-based defect detection tasks to a high-performance edge node while handling simple temperature monitoring tasks locally.

9.3.3.3 *Reward Function (R)*

The reward function provides feedback to the DRL agent, guiding it toward optimal scheduling decisions. The function should balance multiple objectives, including:

1. Latency Optimization – The agent is rewarded for minimizing task execution delays.
2. Energy Efficiency – Lower power consumption leads to higher rewards, preventing unnecessary energy wastage.
3. Fair Resource Utilization – Ensuring that all nodes contribute to processing tasks rather than overloading a single node.
4. Deadline Adherence – The agent receives penalties for failing to complete tasks within specified deadlines.

5. Network Efficiency – Rewards are given for minimizing data transfer costs and avoiding network congestion.
    - Example: If a task is completed within its deadline while minimizing energy consumption, the agent receives a high reward. If the task exceeds the deadline or consumes excessive power, the agent is penalized with a low reward.

9.3.3.4 *Training Process for DRL-Based Scheduling*

The DRL agent undergoes a continuous learning process:

1. Observe System State – The agent monitors the edge computing environment.
2. Select an Action – Based on its policy, the agent makes a scheduling decision (e.g., task offloading).
3. Execute and Receive Reward – The decision impacts system performance, and the agent receives a reward.
4. Update Policy – The agent refines its strategy based on the observed reward.
5. Repeat Process – Over time, the agent learns the optimal scheduling strategy.

9.3.3.5 *Real-World Example*

Consider an autonomous vehicle network, where cars rely on edge computing to process sensor data in real time. A DRL-based scheduler could:

- Prioritize collision detection tasks over infotainment updates.
- Allocate high-bandwidth network resources to urgent safety alerts.
- Offload non-critical computations (e.g., map updates) to the cloud during low-traffic periods.

9.3.3.6 *Conclusion*

A DRL-based task scheduling system provides a dynamic and adaptive approach to managing edge workloads. By defining an efficient state space, action space, and reward function, the DRL agent can optimize task allocation, minimize latency, and enhance resource utilization. With continuous learning, it outperforms traditional scheduling methods, making edge computing systems smarter, faster, and more efficient.

9.4 DRL Algorithms for Task Scheduling

DRL employs various algorithms to optimize task scheduling in edge computing environments. These algorithms can be categorized into three main types: *value-based (Deep Q-Network), policy-based (Policy Gradient Methods), and hybrid approaches (Actor-Critic Algorithms).* Each method has its unique strengths, which make them suitable for different scheduling challenges.

9.4.1 Deep Q-Network

9.4.1.1 *Overview*

DQN is a value-based reinforcement learning algorithm that extends the traditional *Q-learning* approach by incorporating DNNs to handle large state spaces. It is well-suited for *discrete action spaces*, where scheduling decisions involve selecting from a fixed set of possible actions (e.g., selecting one of several available edge nodes to process a task).

9.4.1.2 *How DQN Works in Task Scheduling*

DQN estimates the *Q-value function*, which represents the expected cumulative reward for taking a specific action in a given state. The algorithm updates the Q-values iteratively to refine scheduling decisions:

1. **State Representation** – The scheduler observes system parameters (e.g., CPU load, network bandwidth, and task priority).
2. **Action Selection** – The agent chooses a scheduling action (e.g., assigning a task to an edge node).
3. **Reward Computation** – The agent receives a reward based on system performance (e.g., lower latency and better resource utilization).
4. **Q-Value Update** – The neural network updates Q-values using experience replay and target networks to stabilize learning.

9.4.1.3 *Key Features of DQN for Edge Computing*

- **Handles large state spaces** using DNN.
- **Learns optimal scheduling strategies dynamically** without predefined rules.
- **Uses experience replay** to improve learning efficiency by storing past experiences and training on them randomly.

9.4.1.4 *Example: Smart Traffic Management System*

A DQN-based scheduler can dynamically allocate computing resources to process *real-time traffic camera feeds* in a smart city.

- If an intersection has *high traffic congestion*, the agent may prioritize allocating GPU resources to process video analytics quickly.
- If a *low-traffic area* is detected, fewer computational resources can be allocated to reduce energy consumption.
- Over time, the scheduler learns which intersections require *more or fewer computing resources* based on historical traffic patterns.

9.4.1.5 *Limitations of DQN*

- **Inefficient for continuous action spaces** (e.g., fine-grained resource allocation).
- **High memory and computational overhead** due to experience replay.
- **Slow convergence** in large-scale scheduling environments.

9.4.2 Policy Gradient Methods

9.4.2.1 *Overview*

Unlike value-based methods like DQN, *policy gradient (PG) methods* directly learn an optimal scheduling policy by maximizing the expected reward. Instead of maintaining a Q-value function, these methods optimize a *policy function* that maps states to actions.

9.4.2.2 *Popular Policy Gradient Algorithms*

1. **Proximal Policy Optimization**
    - Uses a clipped objective function *to prevent large policy updates*, improving training stability.
    - Suitable for edge scheduling scenarios requiring *continuous decision-making* (e.g., fine-grained bandwidth allocation).
2. **Trust Region Policy Optimization (TRPO)**
    - Enforces a *trust region constraint* to ensure *gradual and stable policy updates*.
    - Well-suited for *high-dimensional task scheduling problems* where abrupt changes in policy can degrade performance.

9.4.2.3 *How Policy Gradient Methods Work in Edge Scheduling*

1. The agent *observes the system state* (e.g., CPU usage and energy levels).
2. It selects an *action* (e.g., allocating CPU cores and offloading a task).
3. The *reward function evaluates* the effectiveness of the scheduling decision.
4. The *policy is updated* to maximize expected long-term rewards.

9.4.2.4 *Key Features of Policy Gradient Methods*

- **More effective in continuous action spaces** (e.g., adjusting CPU frequency dynamically).
- **Better exploration–exploitation balance** by sampling diverse scheduling strategies.
- **Stable training performance** compared to DQN in large, dynamic environments.

9.4.2.5 *Example: Cloud-Assisted Edge Task Scheduling*

A *PPO-based scheduler* could dynamically decide *how much processing power to allocate* to different edge nodes based on real-time workloads.

- If an edge node has *low computational load*, the scheduler can allocate *fewer CPU cores* to conserve energy.
- If the system detects *high-priority video analytics tasks*, it can allocate *more GPU resources* to speed up processing.

9.4.2.6 *Limitations of Policy Gradient Methods*

- **Slower convergence** than value-based methods like DQN.
- **High variance in learning**, requiring techniques like *baseline subtraction* to stabilize training.

9.4.3 Actor–Critic (AC) Algorithms

9.4.3.1 *Overview*

AC algorithms combine *value-based and policy-based* approaches to leverage the advantages of both. These methods consist of two components:

1. **Actor** – Learns the optimal scheduling policy (decides actions).
2. **Critic** – Evaluates the actions taken by the actor using a value function.

This hybrid approach improves stability and efficiency in dynamic scheduling environments.

9.4.3.2 *Popular Actor–Critic Algorithms for Task Scheduling*

1. **Asynchronous Advantage Actor-Critic (A3C)**
   - Runs multiple agents in parallel to *speed up training*.
   - Uses an *advantage function* to improve policy updates.
2. **Deep Deterministic Policy Gradient**
   - Handles *continuous action spaces* (e.g., adjusting task priority dynamically).
   - Well-suited for scheduling *real-time tasks* with fine-grained control.

9.4.3.3 *How Actor–Critic Works in Edge Scheduling*

1. The Actor selects a scheduling action (e.g., sending a task to an edge node).
2. The Critic evaluates the action by estimating its Q-value.
3. The Actor updates its policy based on the Critic’s feedback.
4. Over time, the scheduler improves efficiency by learning better resource allocation strategies.

9.4.3.4 *Key Features of Actor–Critic Methods*

- **Efficient in high-dimensional state-action spaces**.
- **More stable learning process** compared to standalone policy gradient methods.
- **Faster convergence** than DQN in real-time environments.

9.4.3.5 *Example: Edge-Based Video Streaming Optimization*

In a video streaming service, A3C could:

- **Dynamically allocate network bandwidth** based on real-time congestion levels.
- **Prioritize HD streaming** for high-demand users while reducing resolution for lower-priority requests.
- **Optimize resource allocation** across multiple edge nodes to ensure smooth playback with minimal buffering.

9.4.3.6 *Limitations of Actor–Critic Methods*

- **Complex implementation** due to multiple neural networks.
- **Higher computational cost**, making it resource-intensive for lightweight edge nodes.

9.4.3.7 *Comparison of DRL Algorithms for Task Scheduling*

Table 9.1 DRL Algorithms⏎

Algorithm	*Type*	*Strengths*	*Weaknesses*	*Suitable Use Cases*
DQN	Value-based	Good for discrete action spaces, learns optimal scheduling	Not efficient for continuous tasks, slow convergence in large-scale environments	Selecting the best edge node for task offloading

Algorithm	*Type*	*Strengths*	*Weaknesses*	*Suitable Use Cases*
		policies dynamically		
PPO/TRPO	Policy-based	Stable training, suitable for continuous action spaces	High variance, slower convergence than DQN	Dynamic CPU/GPU allocation, fine-grained task prioritization
A3C/DDPG	Actor-Critic	Efficient in high-dimensional spaces, balances exploration & exploitation	Complex implementation, high computational cost	Adaptive task scheduling, real-time workload balancing

9.4.3.8 *Conclusion*

Choosing the right *DRL algorithm* for edge task scheduling depends on the specific challenges:

- **DQN** is ideal for *discrete scheduling decisions* like selecting an edge node.
- **Policy Gradient Methods (PPO and TRPO)** are useful for *continuous resource allocation*.
- **Actor–Critic methods (A3C and DDPG)** offer a *balanced approach* for complex, dynamic environments.

By leveraging these *advanced DRL techniques*, edge computing systems can achieve *better resource utilization, lower latency, and enhanced energy efficiency*, ultimately improving the performance of real-time applications as shown in Table 9.1.

9.5 System Architecture

The deployment of DRL for task scheduling in edge computing involves multiple stages, including data collection, model training, and deployment. These steps ensure that the scheduler continuously learns and adapts to dynamic workloads, resource availability, and system constraints.

9.5.1 Data Collection

***9.5.1.1** Overview*

For a DRL-based task scheduler to make optimal decisions, it requires extensive data from edge nodes. This data helps the model *understand resource availability, workload patterns, and system performance* in real-world scenarios.

***9.5.1.2** Types of Data Collected*

1. **Resource Availability** – CPU utilization, memory usage, network bandwidth, and battery levels of edge nodes.
2. **Task Arrivals** – Task frequency, computational complexity, deadlines, and priority levels.
3. **System Performance Metrics** – Task execution time, latency, energy consumption, and successful task completion rates.

***9.5.1.3** How Data Collection Works in Edge Environments*

- **Real-Time Monitoring** – Each edge node continuously tracks and reports its resource usage and incoming tasks.
- **Historical Data Logging** – Data from past scheduling decisions is stored to train and fine-tune the DRL model.
- **Cloud-Assisted Data Aggregation** – If edge nodes have limited storage, collected data can be periodically sent to a cloud server for further analysis and model training.

9.5.1.4 *Example: Smart Healthcare System*

In a remote patient monitoring system, wearable IoT devices (e.g., smartwatches and ECG monitors) continuously collect and transmit health data (heart rate, blood pressure, and oxygen levels) to edge servers.

- If a patient shows *irregular vitals*, an urgent task is scheduled for real-time processing at the nearest edge node.
- If data collection is *routine*, processing can be scheduled with lower priority, optimizing resource use.

9.5.2 DRL Model Training

9.5.2.1 *Overview*

Once sufficient data is collected, a centralized or distributed DRL model is trained using both historical data and simulated workloads. The goal is to develop an intelligent scheduler that predicts the best scheduling decisions under varying conditions.

9.5.2.2 *Training Process*

1. **State Definition** – Represents system parameters (CPU usage, task queue length, and network latency).
2. **Action Definition** – Includes scheduling decisions (task offloading, CPU allocation, and priority adjustment).
3. **Reward Function Design** – Assigns rewards based on metrics like latency reduction, energy efficiency, and task completion rates.
4. **Model Optimization** – The DRL agent undergoes multiple training cycles, learning from past scheduling decisions and improving over time.

9.5.2.3 *Centralized vs. Distributed Training*

Table 9.2 Different Training Approaches⏎

TrainingApproach	*Description*	*Advantages*	*Challenges*
Centralized Training	DRL model is trained in a cloud or high-performance computing environment	More computing power, better global optimization	Requires high-bandwidth connections for data transfer
Distributed Training	Edge nodes train local DRL models and share updates periodically	Lower latency, real-time learning at the edge	Requires synchronization between nodes

9.5.2.4 *Example: Video Surveillance System*

In a smart city security network, thousands of surveillance cameras generate large volumes of video data.

- A centralized DRL model is trained using historical footage to learn which tasks (e.g., motion detection and facial recognition) require immediate processing and which can be offloaded or delayed.
- A distributed DRL model allows edge nodes to learn locally, adapting to specific locations (e.g., high-crime areas vs. quiet neighborhoods) without relying on a central cloud as shown in Table 9.2.

9.5.3 Deployment

9.5.3.1 *Overview*

After training, the DRL model is deployed across edge nodes, where it dynamically schedules tasks in real time. The model continuously improves as it interacts with new data, adapting to changing workloads, resource availability, and environmental conditions.

9.5.3.2 *Steps in Deployment*

1. **Model Distribution** – The trained DRL model is either deployed directly on edge nodes or in a hybrid setup (partially on the cloud, partially on the edge).
2. **Real-Time Decision-Making** – The model schedules tasks dynamically based on real-time system states.
3. **Continuous Learning and Fine-Tuning** – Edge nodes periodically update the model using new data from ongoing operations, ensuring improved performance over time.

9.5.3.3 *Deployment Strategies*

Table 9.3 Different Deployment Strategies⏎

Strategy	*Description*	*Best Use Case*
On-Device Deployment	DRL model runs locally on edge nodes	Applications requiring ultra-low latency, such as autonomous vehicles
Cloud–Edge Hybrid Deployment	Model runs partially in the cloud and partially on the edge	Systems balancing computational efficiency and response time, such as smart grids
Federated Learning Deployment	Edge nodes train their own models and periodically update a global model	Privacy-sensitive applications like personalized healthcare

9.5.3.4 *Example: Autonomous Vehicles*

In self-driving cars, onboard edge devices process sensor data from cameras, LiDAR, and radar to make real-time driving decisions.

- The DRL scheduler decides which computations should be processed locally (e.g., obstacle detection and lane changes) and which can be offloaded to nearby edge servers (e.g., high-definition map updates).
- An FL approach allows multiple vehicles to collaboratively train a shared DRL model, improving overall road safety without sharing sensitive user data as shown in Table 9.3.

9.5.3.5 *Key Takeaways*

1. **Data Collection** ensures that edge nodes gather essential information on resource availability, task demands, and system performance, providing a foundation for intelligent scheduling.
2. **DRL Model Training** utilizes historical and simulated data to create a model that can predict optimal scheduling strategies without human intervention.
3. **Deployment** enables real-time adaptive scheduling at the edge, reducing latency, optimizing resource usage, and improving system efficiency.

By integrating continuous learning and dynamic adaptation, DRL-based scheduling enables next-generation edge computing solutions, ensuring faster response times, better resource allocation, and energy-efficient task execution across various industries.

9.6 Performance Evaluation

To assess the effectiveness of DRL-based task scheduling in edge computing environments, various performance metrics are used. Additionally, a comparative analysis with traditional scheduling methods highlights the advantages and challenges of DRL-based approaches.

9.6.1 Metrics for Evaluating DRL-Based Scheduling

Evaluating DRL-based scheduling involves multiple key performance indicators (KPIs) to measure improvements in latency, energy efficiency, and throughput compared to traditional methods.

9.6.1.1 Latency Reduction

Definition: Measures the improvement in task response time when using DRL-based scheduling compared to traditional methods.

- **Formula:** Latency Reduction=Traditional Latency−DRL-Based Latency Traditional Latency×100%\text{Latency Reduction} = \frac{\text{Traditional Latency} – \text{DRL-Based Latency}}{\text{Traditional Latency}} \times 100\%
- **Why It Matters:** Many edge computing applications (e.g., autonomous vehicles, augmented reality, and remote healthcare) require ultra-low latency to function effectively.
- **Example:**
    - In autonomous driving, a task like real-time obstacle detection must be processed within milliseconds.
    - A traditional heuristic-based scheduler might take 500 ms, whereas a DRL-based scheduler can optimize task offloading and reduce it to 200 ms, yielding a 60% latency reduction.

9.6.1.2 Energy Efficiency

Definition: Assesses the power consumption savings achieved through DRL-based task scheduling by optimizing resource utilization and reducing unnecessary computations.

- **Formula:** Energy Savings=Traditional Energy Consumption−DRL Energy ConsumptionTraditional Energy

- **Why It Matters:** Many edge nodes operate on limited power sources (e.g., drones, IoT sensors, battery-powered edge devices).
- **Example:**
  - In a smart agriculture application, edge nodes analyze soil moisture and weather conditions to schedule irrigation tasks.
  - A conventional scheduler might allocate high processing power to all tasks, draining battery life quickly.
  - A DRL-based scheduler learns to prioritize critical tasks and offload non-urgent tasks to cloud servers, reducing energy consumption by 30%.

9.6.1.3 *Throughput*

Definition: Evaluates the number of successfully executed tasks per unit time, indicating system efficiency.

- **Formula:** Throughput

- **Why It Matters:** High throughput ensures that more tasks are processed within a given time frame, which is crucial for real-time applications like video surveillance and industrial automation.
- **Example:**
  - In an edge-based video surveillance system, multiple cameras stream video feeds to edge nodes for real-time motion detection.
  - A traditional round-robin scheduler may assign tasks evenly but overload weaker nodes.

    - A DRL-based scheduler dynamically assigns tasks based on node capabilities, improving throughput **by 40%.**

9.6.1.4 *Task Completion Rate*

Definition: Measures the percentage of tasks that are successfully completed within a deadline.

- **Formula:** Task Completion Rate=Tasks Completed on TimeTotal Tasks Submitted×100%\text{Task Completion Rate} = \frac{\text{Tasks Completed on Time}}{\text{Total Tasks Submitted}} \times 100\%
- **Why It Matters:** Many edge applications have strict time constraints, such as predictive maintenance in manufacturing or emergency response in healthcare.
- **Example:**
    - In a hospital's emergency response system, a traditional scheduler might complete 70% of urgent processing tasks on time, whereas a DRL-based scheduler could increase this to 90% by dynamically reallocating resources.

9.6.2 Comparative Analysis

To demonstrate the effectiveness of DRL-based scheduling, its performance is compared against traditional heuristic and machine learning-based scheduling methods.

9.6.2.1 *Traditional Heuristic-Based Scheduling*

- **Definition:** Uses predefined rules (e.g., first come first serve [FCFS], round-robin (RR), shortest job next [SJN]) to schedule tasks.
- **Advantages:**
    - Simple and easy to implement.

    - Works well in static environments with predictable workloads as shown in Table 9.4.
- **Disadvantages:**
    - Cannot adapt dynamically to changing workloads.
    - Leads to poor resource utilization in highly variable environments.

9.6.2.1.1 *Example Comparison (Heuristic vs. DRL)*

Table 9.4 Comparision of Heuristic versus DRL⏎

SchedulingMethod	*Latency (ms)*	*Energy Efficiency*	*Task Completion Rate*	*Throughput*
Round Robin (RR)	500	Low	70%	Medium
DRL-Based Scheduling	200	High	90%	High

🚀 DRL reduces latency and improves energy efficiency by dynamically allocating resources based on real-time data.

9.6.2.2 *Machine Learning-Based Scheduling*

- **Definition:** Uses supervised learning (e.g., decision trees and support vector machines) to predict optimal scheduling decisions.
- **Advantages:**
    - Can learn from historical data to improve scheduling.
    - Performs better than heuristics in dynamic environments.
- **Disadvantages:**
    - Requires labeled training data, which may be difficult to obtain.
    - Lacks adaptability – once trained, the model does not update in real time.

Example Comparison (ML vs. DRL-Based Scheduling)

Table 9.5 Different Scheduling Methods⏎

SchedulingMethod	*Adaptability*	*Computational Efficiency*	*Learning from Real-Time Data*
Supervised ML (SVM, Decision Trees)	Medium	High	No
DRL-Based Scheduling	High	Medium	Yes

🔥 DRL continuously learns and adapts to real-time system changes, unlike traditional ML models that rely on static training data.

9.6.2.3 *Hybrid Approaches (Heuristic + ML vs. DRL)*

- Some systems combine heuristic methods with ML-based predictions, but they still lack the full adaptability of DRL.
- **Example:**
  - A rule-based heuristic (e.g., SJN**)** may work well for small workloads.
  - A machine learning model may improve performance by predicting task priority.
  - However, only DRL continuously improves and optimizes scheduling in real time.

9.6.2.4 *Final Takeaways*

9.6.2.4.1 *Why DRL-Based Scheduling Is Superior*

- **Adaptive:** Learns from real-time workloads and adjusts scheduling decisions dynamically.
- **Energy Efficient:** Optimizes power consumption by reducing unnecessary computations.

- **High Performance:** Achieves lower latency, higher throughput, and better task completion rates.
- **Scalable:** Works well in heterogeneous edge environments with multiple resource constraints.

9.6.2.4.2 *When to Use Traditional Approaches Instead*

- **For static environments with predictable workloads**, heuristics may be simpler and more efficient.
- **If labeled data is available but real-time adaptation is not required**, traditional ML methods can be effective.

By leveraging DRL for edge task scheduling, systems can achieve significant improvements in efficiency, scalability, and real-time adaptability, making it a powerful choice for next-generation edge computing applications as shown in Table 9.5.

9.7 Case Studies

DRL-based task scheduling has transformative applications across various industries. It enhances efficiency, adaptability, and real-time decision-making by dynamically allocating resources and optimizing computational workloads. Below, we explore three major domains where DRL-driven task scheduling significantly improves performance:

9.7.1 Smart Healthcare

9.7.1.1 *Overview*

Smart healthcare systems rely on real-time patient monitoring, medical imaging analysis, and emergency response mechanisms. DRL-based scheduling ensures that critical medical tasks are processed with minimal latency, improving patient care and system efficiency.

9.7.1.2 *How DRL Enhances Healthcare Scheduling*

1. **Real-Time Patient Monitoring:**

   - Wearable IoT devices (e.g., smartwatches, ECG monitors, and glucose sensors) continuously collect vital signs.
   - DRL dynamically schedules urgent cases (abnormal vitals) for immediate processing while delaying non-urgent data to conserve resources.

2. **Medical Imaging Analysis:**

   - AI-driven image processing (e.g., Magnetic Resonance Imaging [MRI] and Computed Tomography [CT] scans) requires substantial computational power.
   - DRL optimizes scheduling by prioritizing critical cases (e.g., tumor detection) over routine scans, ensuring fast diagnoses.

3. **Remote Surgery and Telemedicine:**

   - Low-latency task execution is crucial for robotic-assisted surgeries.
   - DRL helps in dynamically allocating computing resources to ensure real-time video transmission and robotic control.

9.7.1.3 *Example: Remote Patient Monitoring*

A smart Intensive Care Unit (ICU) system monitors patients’ vital signs in real time. When a patient’s heart rate drops suddenly, DRL schedules an immediate alert and analysis at the edge, instead of sending data to the cloud, reducing response time from 5 seconds to 1 second – a crucial improvement in emergency cases.

9.7.1.4 *Benefits of DRL in Healthcare:*

- Reduces latency in processing emergency data.

- Enhances energy efficiency of battery-powered medical devices.
- Improves task prioritization for life-critical cases.

9.7.2 Autonomous Vehicles

9.7.2.1 *Overview*

Autonomous driving systems must process a massive amount of sensor data (LiDAR, cameras, radar, and GPS) in real time to make split-second decisions. DRL-based scheduling helps allocate computational resources efficiently, ensuring safe and responsive navigation.

9.7.2.2 *How DRL Enhances Scheduling in Autonomous Vehicles*

1. **Sensor Data Processing:**
    - A self-driving car collects data from multiple sensors simultaneously.
    - DRL-based scheduling prioritizes critical tasks (e.g., pedestrian detection and collision avoidance) over less urgent ones (e.g., traffic sign recognition).
2. **Decision-Making in Navigation:**
    - DRL optimizes the scheduling of path planning, lane changes, and obstacle avoidance to ensure safe driving.
    - It dynamically shifts computing resources based on traffic congestion, weather conditions, and road hazards.
3. **V2X Communication (Vehicle-to-Everything):**
    - Autonomous vehicles communicate with traffic signals, other cars, and edge servers to improve road safety.

        - DRL schedules and optimizes real-time data sharing while minimizing bandwidth usage.
        - **Example: Real-Time Collision Avoidance**

A self-driving car detects an obstacle 100 meters ahead. Traditional scheduling might delay processing due to ongoing computations. However, a DRL-based scheduler instantly prioritizes obstacle detection, ensuring a reaction time of less than 50 milliseconds, preventing an accident.

9.7.2.3 *Benefits of DRL in Autonomous Vehicles*

- Reduces decision-making latency, improving navigation safety.
- Enhances resource utilization in on-board computing units.
- Balances workloads between on-board processors and edge/cloud servers.

9.7.3 Industrial IoT and Smart Manufacturing

9.7.3.1 *Overview*

Factories and industrial plants generate massive data streams from sensors, robots, and production lines. Efficient task scheduling is vital for predictive maintenance, workload balancing, and real-time process optimization. DRL-based scheduling enhances efficiency and reduces downtime.

9.7.3.2 *How DRL Enhances Industrial IoT Scheduling*

1. **Predictive Maintenance:**

    - Sensors track machine conditions (e.g., vibration, temperature, and pressure).
    - DRL schedules maintenance tasks before failures occur, reducing unexpected breakdowns.

2. **Workload Balancing in Manufacturing:**
    - Industrial robots and machines have varying computational loads.
    - DRL dynamically assigns workloads based on machine availability and power consumption, optimizing energy efficiency.
3. **Supply Chain and Logistics Optimization:**
    - DRL-based scheduling optimizes warehouse inventory tracking and shipment processing.
    - AI-powered robotic sorting systems allocate picking and packing tasks based on order priority and resource availability.

9.7.3.3 *Example: Smart Factory Predictive Maintenance*

A smart factory uses IoT sensors to monitor a conveyor belt motor. DRL detects early signs of overheating and schedules a maintenance check before failure occurs, preventing unplanned downtime that could cost the factory $50,000 per hour.

9.7.3.4 *Benefits of DRL in Industrial IoT*

- Reduces maintenance costs by preventing unexpected failures.
- Optimizes resource allocation, improving energy efficiency.
- Enhances manufacturing throughput by balancing workloads across machines as shown in Table 9.6.

9.7.3.5 *Final Takeaways*

Table 9.6 Applications of DRL⏎

Application	*Key Challenges*	*How DRL Helps*	*Example*

Application	*Key Challenges*	*How DRL Helps*	*Example*
Smart Healthcare	High latency in emergency response	Prioritizes urgent medical tasks	Remote ICU patient monitoring
Autonomous Vehicles	Real-time sensor processing delays	Optimizes task prioritization	Collision avoidance system
Industrial IoT	Downtime due to unplanned failures	Enables predictive maintenance	Smart factory scheduling

DRL-based task scheduling is revolutionizing edge computing by reducing latency, improving efficiency, and optimizing resource allocation across diverse industries. 🚀

9.8 Future Directions

As edge computing evolves, DRL-based scheduling will integrate with emerging technologies to improve efficiency, security, and scalability. Below are three key future directions that can revolutionize DRL-driven task scheduling in edge environments.

9.8.1 Federated Learning for DRL-Based Task Scheduling

9.8.1.1 *Overview*

FL is a decentralized machine learning approach where multiple edge devices collaboratively train a shared model without sharing raw data. By integrating FL with DRL, edge nodes can train scheduling models locally while preserving privacy and scalability.

9.8.1.2 *How FL Enhances DRL-Based Scheduling*

1. **Privacy-Preserving Model Training:**
    - Sensitive data (e.g., health records, industrial logs, vehicle navigation data) remains on the local edge node, reducing privacy concerns.
    - FL enables DRL models to learn scheduling patterns across multiple devices without exposing raw data to a central cloud server.
2. **Improved Scalability and Adaptability:**
    - Instead of a single, centralized DRL model, multiple edge devices train local models and share only model updates.
    - This reduces communication overhead and allows the scheduling model to adapt to regional workload variations.
3. **Enhanced Security and Compliance:**
    - Many industries (e.g., healthcare, finance, and defense) have strict data privacy regulations (e.g., HIPAA and GDPR).
    - FL ensures compliance by keeping user data decentralized while still optimizing task scheduling.

9.8.1.2.1 *Example: FL-Driven DRL in Smart Healthcare*

A federated DRL scheduling model deployed across multiple hospitals can learn from local patient monitoring workloads without sharing private patient data. Each hospital trains its own DRL model and only shares updates, leading to a more efficient and privacy-preserving task scheduling system.

9.8.1.3 *Benefits of Federated Learning in DRL-Based Scheduling:*

- **Privacy-Preserving**: No need to send raw data to a central cloud.
- **Scalable**: Supports large networks of edge devices without overloading a single model.
- **Efficient**: Reduces network congestion by transmitting only model updates instead of full datasets.

9.8.2 Multi-Agent Reinforcement Learning (MARL) for Task Scheduling

9.8.2.1 *Overview*

Traditional DRL approaches involve a single agent making scheduling decisions. However, in large-scale edge computing environments, multiple distributed edge nodes must coordinate task scheduling dynamically. MARL introduces multiple cooperative or competitive DRL agents, enabling distributed decision-making.

9.8.2.2 *How MARL Enhances Edge Task Scheduling*

1. **Collaborative Scheduling across Edge Nodes:**
    - Each edge node acts as an independent RL agent, making localized scheduling decisions.
    - Agents share insights and collaborate to optimize overall system performance.
2. **Load Balancing and Fault Tolerance:**
    - MARL enables intelligent task offloading across multiple nodes.
    - If one edge node becomes overloaded, another agent takes over dynamically.
3. **Reduced Computational Bottlenecks:**

    - Instead of relying on a single global scheduler, MARL distributes scheduling responsibilities across multiple nodes.
    - This improves responsiveness and eliminates central bottlenecks.

9.8.2.3 *Example: MARL in Autonomous Vehicle Fleets*

In a smart traffic management system, each autonomous vehicle acts as an RL agent, dynamically scheduling computing tasks (e.g., collision detection, navigation planning, and real-time communication). Vehicles cooperate to share processing loads, reducing traffic congestion and avoiding collisions.

9.8.2.4 *Benefits of Multi-Agent Reinforcement Learning:*

- **Decentralized Scheduling**: Edge nodes make local decisions, reducing central server dependencies.
- **Scalable and Robust**: Improves fault tolerance – if one node fails, others adapt dynamically.
- **Optimized Resource Utilization**: Balances workloads across multiple edge nodes for efficient processing.

9.8.3 Integration with 6G Networks for Enhanced Scheduling

9.8.3.1 *Overview*

With the advent of 6G networks (expected around 2030), DRL-based task scheduling will see massive improvements due to ultra-low latency, higher bandwidth, and AI-native communication systems.

9.8.3.2 *How 6G Enhances DRL-Based Scheduling*

1. **Ultra-Low Latency (Sub-Millisecond Response Time):**

    - 6G networks will achieve latency below 1 ms, enabling near-instantaneous task scheduling decisions.
    - This is crucial for applications like real-time robotic surgery, drone swarms, and holographic communications.

2. **AI-Native Networks and Edge Intelligence:**

    - Unlike previous generations (4G and 5G), 6G will have built-in AI/ML support, allowing DRL models to run directly on network infrastructure.
    - DRL-based schedulers will continuously learn and adapt based on real-time network conditions.

3. **Seamless Task Migration and Edge-Cloud Integration:**

    - 6G will enable ultra-fast task migration between edge nodes and cloud data centers.
    - DRL will dynamically offload computationally heavy tasks to the cloud when edge resources are limited.

9.8.3.3 *Example: 6G-Enhanced DRL Scheduling for Smart Cities*

A 6G-powered smart city deploys edge-based surveillance cameras, traffic sensors, and AI-driven emergency response systems. DRL-based scheduling dynamically assigns processing tasks between local edge nodes and cloud AI models, ensuring real-time decision-making for public safety and urban management.

9.8.3.4 *Benefits of 6G-Integrated DRL Scheduling:*

- **Near-Instant Task Execution**: Enables real-time AI processing for critical applications.
- **Improved Network Adaptability**: DRL agents dynamically adjust scheduling policies based on network congestion and availability.

- **Seamless Cloud–Edge Cooperation**: Enables smooth task migration between edge nodes and remote cloud servers.

9.8.3.5 *Final Takeaways and Future Outlook*

Table 9.7 Summary of DRL⏎

Future Technology	*Key Impact on DRL-Based Scheduling*	*Example Use Case*
Federated Learning	Enhances privacy and scalability	Secure healthcare data processing without exposing patient records.
Multi-Agent RL	Enables cooperative scheduling across multiple edge nodes	Autonomous vehicle fleets optimizing real-time navigation.
6G Networks	Provides ultra-low latency and AI-native networks	Smart city infrastructure with real-time AI-driven decision-making.

By integrating federated learning, multi-agent RL, and 6G networks, DRL-based scheduling will unlock new levels of efficiency, scalability, and intelligence in edge computing environments!

9.9 Conclusion

DRL has emerged as a transformative solution for task scheduling in edge computing, addressing the inherent challenges of resource constraints, dynamic workloads, latency sensitivity, and infrastructure heterogeneity. Traditional scheduling approaches, such as heuristic-based and static rule-based methods, often fail to adapt to rapidly changing environments, leading to suboptimal resource utilization and increased latency. DRL, by leveraging continuous learning and adaptive decision-making, provides a robust framework for optimizing resource allocation, reducing computational overhead, and improving the overall efficiency of edge computing systems as shown in Table 9.7.

One of the most significant advantages of DRL-based scheduling is its ability to dynamically learn from real-time system states and make intelligent scheduling decisions without requiring predefined rules. By integrating DNN with reinforcement learning principles, DRL efficiently processes high-dimensional system parameters, such as CPU utilization, network bandwidth, and energy consumption, to determine the most effective scheduling actions. Unlike traditional machine learning models, which require extensive labeled data and static training, DRL continuously improves its scheduling policy through interaction with the environment, ensuring that it remains effective even under changing conditions.

The implementation of DRL in edge task scheduling follows a structured framework, involving data collection, model training, and real-time deployment. Edge nodes continuously gather critical system information, including resource availability and task arrivals, which is then used to train a centralized or distributed DRL model. The trained model is subsequently deployed across edge devices, where it autonomously manages scheduling decisions to optimize workload distribution, minimize latency, and enhance energy efficiency. Through this approach, DRL enables low-latency task execution, ensuring that applications with stringent timing constraints – such as autonomous vehicles, augmented reality, and healthcare monitoring – can operate seamlessly in real-world scenarios.

Several DRL-based algorithms have been explored for edge computing, each offering unique advantages. DQNs effectively handle discrete scheduling actions, making them well-suited for task offloading decisions in IoT environments. Policy gradient methods (PPO and TRPO) optimize scheduling policies by directly maximizing expected rewards, enabling better adaptability to fluctuating workloads. Actor–critic algorithms

(A3C and Soft Actor-Critic [SAC]) strike a balance between value-based and policy-based learning, improving both exploration and exploitation in scheduling decisions. These advanced techniques allow DRL to outperform conventional scheduling approaches in terms of throughput, response time, and power efficiency.

The real-world impact of DRL-based scheduling [11] is evident across multiple domains. In smart healthcare, DRL prioritizes urgent patient monitoring tasks, reducing response times and improving emergency handling. In autonomous driving, real-time scheduling ensures rapid decision-making, enhancing vehicle navigation and safety. In IIoT, predictive maintenance powered by DRL minimizes downtime and optimizes resource utilization. These applications highlight the versatility and practical significance of DRL in edge computing environments.

Looking ahead, FL, MARL, and 6G network integration will further enhance the capabilities of DRL-driven task scheduling. FL enables decentralized training of DRL models while preserving data privacy, making it particularly valuable for healthcare and finance applications. MARL facilitates cooperative task scheduling across multiple edge nodes, improving overall system efficiency. Meanwhile, 6G's ultra-low latency and AI-native architecture will provide an ideal infrastructure for real-time DRL-based scheduling.

In conclusion, DRL is a game changer for task scheduling in edge computing, offering unprecedented adaptability, efficiency, and intelligence. As edge computing continues to evolve, DRL-driven scheduling will play a pivotal role in shaping the future of intelligent, real-time, and energy-efficient computing infrastructures, making it a fundamental technology for next-generation applications.

References

1. Lin, K., Li, Y., Zhang, Q., & Fortino, G. (2021). AI-driven collaborative resource allocation for task execution in 6G-enabled massive IoT. *IEEE Internet of Things Journal*, 8(7), 5264–5273.⏎
2. Mustapha, S. D. S., & Gupta, P. (2024). DBSCAN inspired task scheduling algorithm for cloud infrastructure. *Internet of Things and Cyber-Physical Systems,* 4, 32–39.⏎
3. Gupta, P., Rawat, P. S., Kumar Saini, D., Vidyarthi, A., & Alharbi, M. (2023). Neural network inspired differential evolution based task scheduling for cloud infrastructure. *Alexandria Engineering Journal,* 73, 217–230.
4. Madhusudhan, H. S., Gupta, P., Saini, D. K., & Tan, Z. (2023). Dynamic virtual machine allocation in cloud computing using elephant herd optimization scheme. *Journal of Circuits, Systems and Computers*, 32(11), 2350188.
5. Rawat, P. S., Gaur, S., Barthwal, V., Gupta, P., Ghosh, D., Gupta, D., & Rodrigues, J. J. C. (2025). Efficient virtual machine placement in cloud computing environment using BSO-ANN based hybrid technique. *Alexandria Engineering Journal,* 110, 145–152.⏎
6. HS, M., & Gupta, P. (2024). Federated learning inspired Antlion based orchestration for Edge computing environment. *PLoS One*, 19(6), Art. e0304067.⏎
7. Gupta, P., Anand, A., Agarwal, P., & McArdle, G. (2024). Neural network inspired efficient scalable task scheduling for cloud infrastructure. *Internet of Things and Cyber-Physical Systems,* 4, 268–279.⏎
8. Li, Y., et al. (2020). Reinforcement learning for resource management in edge computing. *IEEE Transactions on Mobile Computing*. 1–22.⏎
9. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*, The MIT Press, Cambridge, Massachusetts, 1998, 322 pp., ISBN 0-262-19398-1.⏎

10. Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. *Nature*. 518(7540), 529–533.⏎
11. Silver, D., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*. *529*(7587), 484–489.⏎

Chapter 10

Secure, Adaptable, and Collaborative AI: Federated Machine Learning Enhanced with Meta-Learning and Differential Privacy

Deepti Bhat, Ansh Chauhan, Chiranth Nagaprasanna, Chirashwi S., and Swathi B. H.

DOI: 10.1201/9781003610168-10

10.1 Introduction

The abundance of data and variety of today’s devices provides enormous potential for advancements in machine learning (ML) and artificial intelligence (AI). However, the issues of traditional ML technology in edge network applications surrounding data silos, privacy leakage and data security risks, regulatory requirements, and engineering obstacles are those that need to be navigated [1]. Federated learning (FL) is an increasingly popular solution that aims to train a global model using data from one or more data owners/sources [2]. This approach centers around the shift from centralized data, compute, and governance to distributed and decentralized architecture with clients and servers. This empowers individual clients to utilize their local data to collaboratively train ML models without the risk of sharing the raw data to a central entity [3]. In FL, each client trains a local model using its own data. These local models are later aggregated by the central FL server to create a global model, which is then redistributed to clients for further refinement. This iterative process continues until satisfactory results are achieved. Despite FL’s promise, several critical concerns remain, particularly in relation to adversarial attacks, privacy leakage, and data heterogeneity.

This chapter embarks with the current issues of FL in Section 10.2, followed by examining the need to strengthen it further in Section 10.3. Section 10.4 covers the general role of differential privacy (DP), whereas the impact of various algorithms along the FL architecture is discussed in Section 10.5. Additionally, meta-learning and its forms and convolution neural networks in the context of FL have been discussed in Sections 10.6, 10.7, and 10.8, respectively. At the end, Section 10.9 covers the conclusion, followed by the references section.

10.2 Issues of Federated Learning

While the FL approach seems like a promising one, there are several concerns with significant presence. Song et al. (2020) showed that the original training data can be reconstructed from the model parameters uploaded by the FL client participants through certain adversarial attacks, which can lead to privacy leakage [4]. When it comes to adversaries, there are primarily two classifications: active and passive. Active adversaries are those that intentionally manipulate training data or trained models to achieve malicious goals. This can involve altering models to prevent global model convergence, or subtly misclassifying a specific set of samples to minimally impact the overall performance of the global model. On the other hand, passive adversaries do not modify data or models, but can still pose a threat to data privacy. For example, they may deduce sensitive information such as local training data from revealed models through gradients, or model updates [3].

The adversarial attacks can take several forms such as model poisoning, data poisoning, and data reconstruction attacks, where the former two are active in nature and the latter is passive. Model poisoning can occur through Byzantine attacks where shared model parameters are tampered to affect the global model. It could involve zero mode attack where weights are set to zero, random mode attack where weights are altered with random values, or can be of flipping mode where the global model updates in the opposite direction. Furthermore, data poisoning attacks may involve label flipping that misclassifies some of the sample data. On the other hand, data reconstruction attacks mainly occur through trained models that are either local or global. One example is Deep Leakage from Gradients (DLG) attack which infers local training data from the publicly shared gradients [3]. The recent dynamic backdoor attacks, however, presents a unique challenge by allowing adversaries to modify attack patterns in due real time, continuously adapting the poisoned dataset and rendering traditional defenses ineffective [5].

10.3 Strengthening Federated Learning

With the presence of myriad possible adversarial attacks, it’s crucial to take extra steps to strengthen one of the main goals of preserving client data privacy in FL. One possibility is secure aggregation where the server aggregates local model updates from clients without learning individual updates. This ensures that the server only receives the sum of updates and not the individual updates themselves. While this method protects from malicious servers or attacks on servers, there still needs to be an implementation where the nature of local client data cannot be deciphered, even if local models or parameters are accessed.

An intuitive step would be homomorphic encryption, which deals with a cryptographic technique with computations being performed on encrypted data. This means that the server can perform computation on encrypted local model updates, without decrypting local model updates. While this may be a good addition, a factor that needs to be considered is computational cost. Encrypting all the data each time and then performing computation on encrypted data may affect results and incur additional computational costs. With such methods and their implications, the approach of DP arises in the middle, which addresses the privacy concern more optimally.

10.4 Role of Differential Privacy

DP, which was proposed by Dwork et al. in 2006 [6], is an advanced solution in which random noise is added to true outputs using rigorous mathematical measures [7]. This leads to results that are statistically indistinguishable between an original aggregate dataset and a differentially additive-noise one. DP can take many forms; for example, label DP considers the situation where only labels are sensitive information that should be protected and is usually applied to differentially private vertical FL [8]. Additionally, Bayesian DP is interested in the change in the posterior distribution of the attacker after observing the private model, compared to the prior. Initially DP was applied at the server levels, due to their centrality, to ensure the output doesn’t reveal information about training data. However, subsequent research has led to the concept of local differential privacy (LDP). This is a more stringent version of differential privacy (DP) as the privacy measures are directly applied at the client level. An added advantage is that it doesn’t require the confirmation of truthful model aggregators. Hence, the need to completely trust the server gets eliminated as the local model with noise is communicated to the aggregator [9]. Based on LDP, there has been an introduction of the shuffle model in recent years. A trusted shuffler is added in between clients and server, with the aim of shuffling the data items submitted by clients to achieve anonymization. This means that each client satisfies the privacy guarantee of LDP when facing the shuffler and then achieves the privacy guarantee of DP when facing the server

[8]. However, there will be additional computational overheads through this which becomes more expensive as the amount of data increases.

Furthermore, using Projected Federated Averaging (PFA), updates from more privacy-sensitive clients (private clients) can be projected using the model updates from clients with larger privacy budgets (public clients). PFA ensures differential privacy while removing noise and enhancing the accuracy of the global model by using the top singular subspace from the public updates. To further reduce communication costs, the methodology includes an extension called PFA+, which requires private clients to send only projected model updates. Experiments in the research stated that in terms of model utility, privacy preservation, and communication efficiency, PFA and PFA+ performed better than conventional federated averaging techniques [10].

10.5 Algorithmic Impacts

The algorithms through which concepts like FL and DP are implemented play a salient role as they have the potential to impact various model parameters. FL algorithms like SCAFFOLD [11, 12] are specifically designed to handle the challenges of heterogeneous data across different devices. These algorithms improve the stability and efficiency of model training by employing techniques like “control variates” to reduce discrepancies, or “drift”, which can arise from variations in local datasets. Furthermore, algorithms such as FedAvg with DP introduce random noise to model updates to protect user privacy [13]. DP-SCAFFOLD [14] is an extension to the SCAFFOLD framework by integrating DP mechanisms adding noise to local model updates while also addressing data heterogeneity. Its convergence analysis utilized a particular initialization of the algorithm and focused on different quantities compared to the original proof. This algorithm operates like standard FL approaches like FedAvg. Overall, it has demonstrated a better performance both theoretically and empirically, outperforming the baseline DP-FedAvg in various experimental scenarios.

When algorithms such as DP-FedAvg add noise to the local model after each gradient step during the local training phase, it ensures that the contributions of individual clients do not reveal sensitive information, even when integrated at the server level. Additionally, gradient clipping [15] limits any individual user’s influence on the model; this technique helps prevent excessively large updates that could disproportionately affect the global model, adding a layer of protection. The noise can be applied directly on the client device before it even leaves the user’s environment or noise can be added after the updates are received by the server but before they are integrated into the global model [15]. This method occurs within a Trusted Execution Environment (TEE), ensuring that even though updates are processed centrally, the privacy of individual users is maintained. This dual approach of applying noise locally and server-side, combined with the security provided by TEE, creates a privacy-preserving environment that meets regulatory standards such as the General Data Protection Regulation (GDPR).

While DP is normally used by introducing noise to the model weights before sharing them through a private channel to the server mode, FEDMD-NFDP seems to have a contrasting implementation. This arises because LDP requires more rounds of communication to converge to a useful model due to addition of noise, since noise can degrade the quality of shared predictions. FEDMD-NFDP [16] is essentially designed to protect data privacy in FL without explicitly adding noise. It focuses on knowledge distillation, where models share predictions on a public dataset and the privacy is maintained through a random sampling mechanism. Two sampling strategies can be explored – with replacement and without replacement – of local data during model updates. Thus, fewer rounds are required to reach the same performance for noise-free DP, in comparison to LDP.

On the contrary, using device-level DP applies privacy guarantees to the entire dataset on a user's device rather than individual data points. This ensures that even if a device holds a large amount of data, the contribution to the model remains private. The differentially private-follow-the-regularized-leader (DP-FTRL) [17] algorithm is a variant of DP-SGD designed for FL without uniform client sampling. This algorithm clips local model updates from clients and adds noise to these updates before aggregation. The noise is carefully calibrated to ensure DP while balancing model utility.

Additionally, FedFGCR is a federated meta-learning framework aimed at improving intelligent fault diagnosis (FD) in decentralized settings with heterogeneous data while ensuring privacy. Traditional deep learning models for FD face challenges due to data privacy concerns and varied data distributions across clients. FedFGCR aims to address these issues by integrating meta-learning techniques, specifically a Model-Agnostic Meta-Learning (MAML) approach to combine global and local features for personalized model training. Experiments on benchmark datasets (Case Western Reserve University [CWRU] and the Korea Advanced Institute of Science and Technology [KAIST]) demonstrate that FedFGCR significantly outperforms existing FL methods like FedAvg, FedProx, and FedPer showing improvements in accuracy and robustness in handling non-IID data [18].

Similarly, the Federated Transfer Learning framework integrates DP to enhance learning from multiple heterogeneous data sources while ensuring data privacy. Introducing Federated Differential Privacy will provide privacy guarantees without needing a trusted central server [19]. Further, a study investigated the impact of privacy constraints and source data heterogeneity on statistical estimation error rates across three classical statistical problems: univariate mean estimation, low-dimensional linear regression, and high-dimensional linear regression. Federated Virtual Clients (FedVC) and Federated Importance Reweighting (FedIR), which reduce class imbalance and nonidentical class distributions, enhance performance. Two new large-scale datasets that reflect real-world distribution patterns – one for landmark recognition and the other for species classification (iNaturalist) – are used to evaluate their methods [20].

Alternatively, BFV-based homomorphic encryption (Brakerski/Fan-Vercauteren scheme) [21] is a type of ring-based fully homomorphic encryption that supports secure and efficient computation on encrypted data. It supports both additions and multiplication operations on cipher texts and the security is derived from the hardness of the Ring Learning with Errors problem. FedCC seems to offer reliable aggregation and the experiment results show that it works well in non-independent and identically distributed (non-IID) data environments and can prevent both targeted and untargeted model poisoning or backdoor attacks. The most accuracy can be recovered for the global model when FedCC is applied against untargeted attacks. FedCC also preserved test accuracy while nullifying attack confidence against targeted backdoor attacks [22].

Based on several research papers, it’s evident that there is no single algorithm that can easily be decided upon. Each algorithm offers to improve security, privacy, or heterogeneity issues through varied perspectives and performs better with respect to varied parameters. The algorithmic steps and the point of application are two key factors that have determined the outcome of the algorithmic implementation. Figure 10.1 illustrates how various algorithms can be applied along the FL architecture.

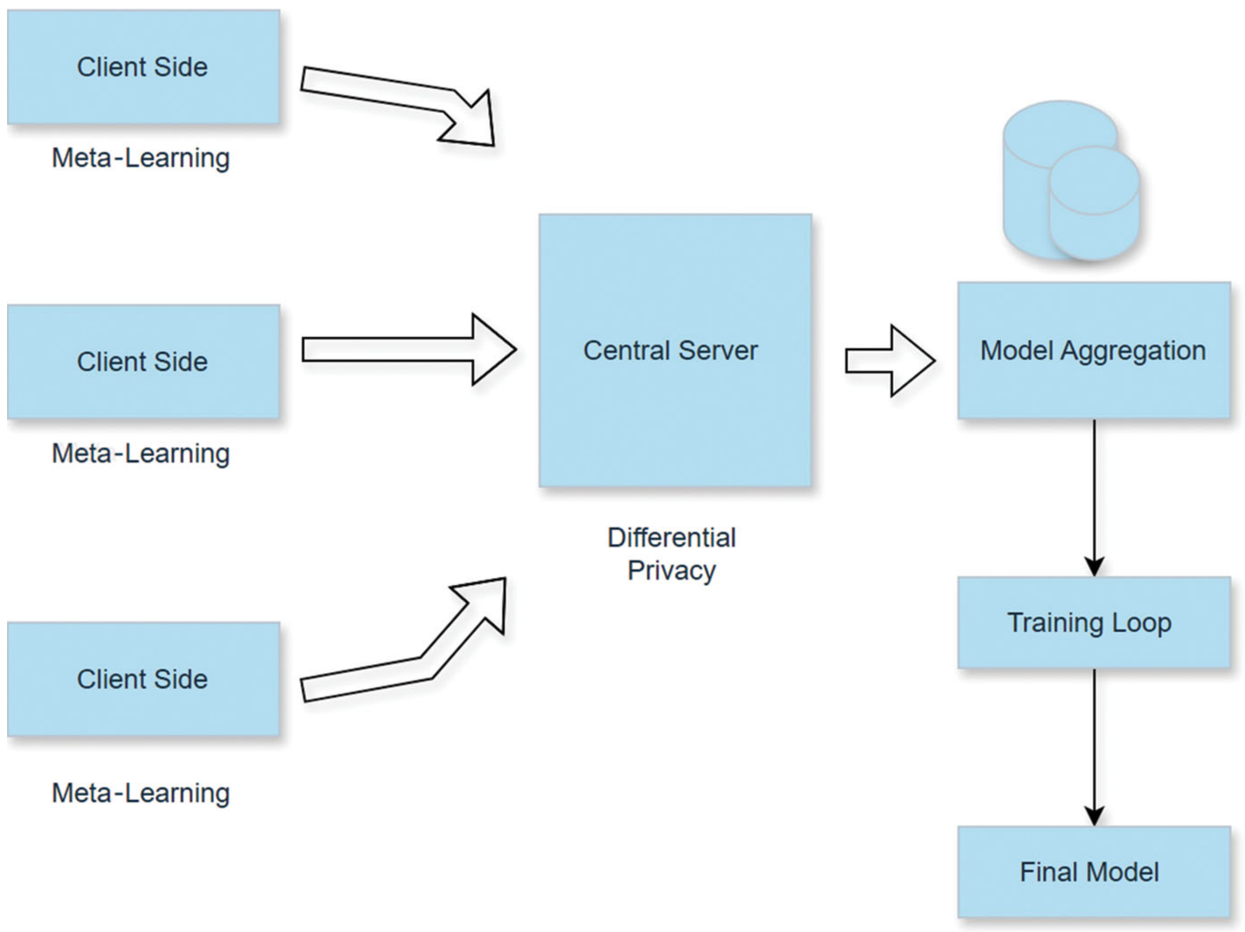


Figure 10.1 Meta-learning and differential privacy applied in federated learning.⏎

10.6 Meta Learning

While security concerns are heavily present, there is also a heterogeneity issue that needs to be dealt with. Meta-learning significantly enhances the adaptability of models in FL by enabling faster and more efficient training on client-side data. One key limitation of traditional FL is its inability to effectively adapt to diverse data distributions across client devices, which often lead to degraded model performance when dealing with non-IID data. As noted by Stojkovic et al., “Federated learning comes with many other challenges due to its distributed nature, heterogeneous compute environments and lack of data visibility” [15]. Meta learning addresses this by giving local models the ability to learn quickly from a small number of data samples, thus improving their performance even with sparse or heterogeneous data.

Recently, many works have begun to exploit various personalization techniques [23] in order to obtain “personalized models” to solve this problem. However, these methods cannot achieve flexible personalization and may ignore the communication bottleneck challenge, which is worth considering in the network applications. Per-FedAvg algorithm [24], combining MAML [25] with FedAvg [26], can quickly obtain personalized models adapted to the data of devices as it’s motivated by the idea of mixing local and global models presented in new formulation of FL [23]. It allows clients to customize the global model using the personalization coefficient (α) as introduced in the Communication Efficient Personalized Federated Meta-Learning algorithm. This coefficient controls how much the local client relies on the global model versus its local model. By adjusting this parameter, better model adaptation can be achieved for clients with different data distributions [1].

Moreover, to address the communication bottleneck challenge in FL, representation learning can be utilized to reduce communication overhead. Representation learning helps encode high-dimensional data into lower-dimensional representation, which can then be transmitted with significantly less bandwidth. This reduces the amount of data sent between clients and the server, addressing the communication bottleneck that typically hampers FL systems. By adopting techniques like compressed model updates and efficient data encoding, communication costs can be minimized without sacrificing model performance. There exist various other hybrid models for cloud, fog, and edge in recent years [27–31].

10.7 CNNS in Federated Learning

A CNN is a category of ML models, namely a type of deep learning algorithm well suited to analyzing visual data. CNNs – sometimes referred to as convnets – use principles from linear algebra, particularly convolution operations, to extract features and identify patterns within images. CNNs are highly adaptable and continue to evolve as researchers develop more complex architectures for the increasingly complex tasks. Algorithms such as transfer learning allow pre-trained CNNs to be fine-tuned for specific tasks, reducing the amount of data and resources required for training from scratch.

In the context of FL, CNNs can be deployed across multiple clients, allowing each of them to train the CNN locally using its own image data. It would be a natural fit for FL applications where visual data from distributed clients, such as medical images or surveillance videos, need to be collaboratively processed without compromising privacy. Techniques like personalized learning and domain adaptation are used to address problems such as data heterogeneity. Communication from large CNNs is mitigated through gradient compression, pruning, and federated dropout. Security concerns such as adversarial attacks and data leakage can be countered using secure aggregation, homomorphic encryption, and differential privacy – particularly local DP to secure client-side communication [32].

CNN architectures like Network in Network (NIN), Visual Geometry Group - 9 Layer Network (VGG-9), and Extended Modified National Institute of Standards and Technology-Merged (EMNIST-M) can be used for classification tasks, implementing adversarial training methods and aggregation algorithms to optimize performance and convergence [33]. In addition, image segmentation using 3D U-Net CNN can be used to combine multiple CNN models to improve accuracy and uncertainty estimation [34]. A method to enhance the image classification performance is through differentially private stochastic gradient descent (DP-SGD). When properly adjusted, over-parameterized models can attain high accuracy even in the presence of differential privacy (DP) constraints. This can be achieved by utilizing strategies like group normalization, weight standardization, augmentation multiplicity, and parameter averaging in addition to carefully optimizing hyperparameters like batch size, noise level, and gradient clipping. This approach has produced state-of-the-art results on both CIFAR-10 and ImageNet [35].

Moreover, an application of FL and DP can be explored through a privacy-preserving framework for analyzing medical images [36]. Medical data are sensitive and cannot be easily shared across institutions due to privacy regulations like GDPR and Health Insurance Portability and Accountability Act (HIPAA). FL allows collaborative learning by training models locally on distributed datasets across different hospitals, without data centralization. Incorporating near-zero centered activation functions like Sigmoid Linear Unit (SiLU) and Gaussian Error Linear Unit (GELU), reducing activation and normalization layers, and using larger kernel sizes, this design improves the model’s robustness in handling non-IID data, outperforming advanced architectures such as Vision Transformers and ConvNeXt. Furthermore, overlapping convolutions and convolution-only downsampling in the stem layer seems to improve feature extraction and model stability, resulting in better performance across various FL benchmarks [17]. Many other similar work using optimization algorithm are discussed in [37–40] using ANN and nature inspired optimization model.

10.8 Factors Affecting Cnns

Including CNNs in FL offers significant opportunities for privacy-preserving image classification yet raises unique challenges, such as computational limitations, data heterogeneity, and maintaining model performance under privacy constraints. In general, choosing appropriate CNN architecture plays an important role. With their significantly low computational and communication cost, lightweight architectures like MobileNet and SqueezeNet suit the FL implementation best due to such environmental scarcity in hardware resources. Additionally, it will be perfectly applicable to those architectures allowing hierarchical extraction of features, where the nature of generalization would remain stronger to various diverse data distributions like ResNet. Recent innovations, such as FedConv, have optimized CNNs for FL further by giving smoother activation functions, reducing normalization layers, and increasing kernel sizes, and hence robust to non-IID setups.

The data heterogeneity that afflicts FL can be addressed through the use of personalized FL. For example, Per-FedAvg combines model-agnostic meta-Learning with federated averaging: it enables personalization through a per-client personalization coefficient by making it possible to fine-tune global models toward each client's local datasets. More specifically, algorithms FedProx and SCAFFOLD account for client drift through the regularization term and the use of control variates, respectively, improving the stability of the training across clients of various classes. Using models that effectively adjust to various client data may help with domain adaptation and transfer learning. Sharing synthetic data generated from public datasets could also be considered to enhance the features among underrepresented clients in order to further reduce imbalances.

Privacy preservation is a key concern in FL. A popular technique used in this context is DP-SGD, which adds noise during the gradient update process to preserve client data while retaining the utility of the model. PFA improves on this by optimizing communication efficiency and increasing the accuracy of the global model when heterogeneous privacy budgets exist among clients. With very high computational costs, homomorphic encryption still warrants robust privacy guarantees for sensitive applications, such as the analysis of medical images, though high-care trade-offs between computational costs and privacy interests are necessary for such applications. Real-world applications demonstrate the feasibility and benefits of these techniques. For example, CNN-based FL has been widely applied to medical image classification applications, such as histopathology for lung cancer subtyping, where DP-SGD ensures compliance with privacy regulations, such as HIPAA. Similarly, privacy-preserving frameworks using LDP have been applied in various video surveillance systems and industrial fault detection scenarios, while ensuring secure collaboration across decentralized datasets.

Table 10.1 provides a comparison analysis of several existing research works across FL, DP, CNNs, and meta-learning. Their primary methodology has been summarized along with the key strengths and limitations that may be encountered. This highlights the current gaps and limitations that persist, which may perhaps be overcome by other research efforts.

Table 10.1 Comparison Analysis of Various Existing Methodologies⏎

TITLE	*METHODOLOGY*	*STRENGTHS*	*LIMITATIONS*
Communication-Efficient Personalized	Proposes Communication Efficient Personalized	Reduces communication costs; enables	Complexity in implementation; may require

TITLE	*METHODOLOGY*	*STRENGTHS*	*LIMITATIONS*
Federated Meta-Learning in Edge Networks [1]	Federated Meta-Learning to reduce communication overhead in edge networks. Introduces personalized parameters to enhance model adaptation.	participation from resource-limited devices; balances global and local model performance.	extensive tuning for different environments; personalization can complicate model convergence.
A Differentially Private Federated Learning Model against Poisoning Attacks in Edge Computing [2]	Designs a weight-based algorithm for anomaly detection of parameters uploaded by devices, enhancing detection rates with minimal communication costs.	Provides strong privacy guarantees; minimizes communication overhead; addresses poisoning attacks effectively.	The assumption of benign clients may not hold in practice; potential accuracy trade-offs due to noise addition.
FEDSECURITY: A Benchmark for Attacks and Defenses in Federated Learning and Federated LLMS [3]	The methodology of the FedSecurity framework is built around two main components: FedAttacker and FedDefender. FedAttacker simulates various adversarial attacks during federated learning (FL) training.	Comprehensive benchmarking capabilities and flexibility. It covers a wide array of attacks and defenses, making it a robust tool for evaluating security in FL systems.	The introduction of defense mechanisms may negatively impact the performance of the global model, necessitating a careful balance between security and efficiency.
PADP-FedMeta: A personalized and adaptive differentially private federated meta learning mechanism for AIoT [4]	This approach combines federated learning (FL) and differential privacy (DP) to address the limitations of conventional FL methods, especially in non-independent and	PADP-FedMeta significantly improves upon existing FL methods by ensuring robust personalization and privacy without	Reliance on differential privacy still introduces noise that can slightly reduce model accuracy.

TITLE	*METHODOLOGY*	*STRENGTHS*	*LIMITATIONS*
	identically distributed data environments.	sacrificing accuracy.	
Dynamic Backdoor Attacks against Federated Learning [5]	Discusses backdoor attacks in the context of federated learning. Introduces the Symbiosis Network to enhance robustness against dynamic attacks by modifying local models during training.	Increases robustness against backdoor attacks; provides insights into adversarial machine learning in federated settings.	Focuses on specific attack scenarios; may not generalize to all federated learning environments; implementation can be complex.
Privacy-Preservation Techniques in Federated Learning: An insightful survey from a GDPR Perspective [7]	Surveys privacy-preservation techniques in FL with a focus on GDPR compliance. Discusses state-of-the-art techniques like secure aggregation, differential privacy, homomorphic encryption, data anonymization, and SMC.	Addresses important legal and ethical concerns regarding data privacy; provides a comprehensive overview of privacy-preserving techniques; emphasizes compliance with GDPR.	Potentially limited by the complexity of implementing privacy techniques in real-world FL scenarios.
Differentially Private Federated Learning: A Systematic Review [8]	A systematic review approach, categorizing and analyzing over 70 studies on differentially private federated learning (DP-FL). It introduces a new taxonomy based on privacy definitions.	Comprehensive and structured approach to categorizing DP models. New Taxonomy: Proposes a novel classification for better clarity in DP models.	No Empirical Validation – lacks experiments to validate theoretical claims.

TITLE	*METHODOLOGY*	*STRENGTHS*	*LIMITATIONS*
FedCC: Robust Federated Learning with CNNs Against Poisoning Attacks [22]	Proposes a federated learning framework that detects and mitigates poisoning attacks during CNN training. This is achieved using anomaly detection techniques to identify malicious clients.	Shows high robustness against both data and model poisoning attacks; improves accuracy and model security in collaborative learning.	The method adds significant overhead to training time; limited evaluation of robustness against sophisticated adversarial attacks.
A Personalized Federated Meta-Learning Method for Intelligent and Privacy-Preserving Fault Diagnosis [18]	Introduces the federated meta-learning based on fine-grained classifier reconstruction (FedFGCR), designed for personalized fault diagnosis in industrial settings. The framework operates in two phases: local training phase and server aggregation phase.	Enhances fault classification accuracy while maintaining privacy; suitable for industries with high data privacy concerns; optimizes classifiers collaboratively.	Limited by the quality of local data; complexity in implementation; challenges in ensuring fairness and efficiency across clients.
Differentially Private Federated Learning on Heterogeneous Data [14]	The paper proposes DP-SCAFFOLD, an algorithm that integrates differential privacy (DP) with the SCAFFOLD framework to address two main challenges in federated learning (FL): handling heterogeneous data and ensuring privacy against an “honest-but-curious” server.	DP-SCAFFOLD effectively balances privacy and utility in FL by leveraging control variates and noise addition to minimize user-drift and variance.	Sensitive to the trade-off between noise addition and the number of local updates, which can impact performance if not properly tuned.
Improving Federated	The paper connects federated learning (FL)	Provides novel insights by	The proposed two-stage

TITLE	*METHODOLOGY*	*STRENGTHS*	*LIMITATIONS*
Learning Personalization via Model Agnostic Meta Learning [24]	and Model Agnostic Meta Learning (MAML) by interpreting the Federated Averaging (FedAvg) algorithm as a meta-learning algorithm. It proposes a novel modification of FedAvg with two stages of training and fine-tuning to optimize for personalized performance.	connecting FL and MAML, showing that FedAvg can be seen as a meta-learning algorithm and highlighting the importance of personalized performance in FL.	training and fine-tuning process adds complexity to the FL system, which may be challenging to implement in real-world scenarios.
Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks [25]	The paper proposes the Model-Agnostic Meta-Learning (MAML) algorithm, which trains a model to adapt quickly to new tasks using minimal data.	The MAML approach is general and model-agnostic, meaning it can work with any model trained via gradient descent. It provides state-of-the-art performance in few-shot learning scenarios and is effective across different learning domains.	One major limitation is the computational complexity due to the second-order derivatives required for the gradient update process.
Communication-Efficient Learning of Deep Networks from Decentralized Data [26]	The paper introduces federated learning. The FedAvg approach reduces the communication cost by limiting the number of rounds and amount of data exchanged	The privacy-preserving nature of federated learning is a key strength, as it eliminates the need for	The main limitation is the complexity of handling non-IID data at scale.

TITLE	*METHODOLOGY*	*STRENGTHS*	*LIMITATIONS*
	between devices and the server.	data to be centralized, significantly reducing privacy risks.	
Federated Learning of a Mixture of Global and Local Models [23]	The paper proposes a novel optimization which is a penalty parameter, λ, to control the trade-off. The authors develop the Loopless Local Gradient Descent (L2GD) algorithm, which employs randomized local steps and variance reduction techniques to solve this new formulation.	The proposed method handles heterogeneous data efficiently, without requiring assumptions of data similarity across devices. It improves communication efficiency by minimizing the number of communication rounds needed for convergence.	The randomness in local steps introduces variability that may affect convergence rates across devices. Complexity of implementing multiple SGD variants with variance reduction poses practical challenges for real-world deployment.
Applied Federated Learning: Architectural Design for Robust and Efficient Learning in Privacy Aware Settings [15]	The paper presents an architecture for federated learning that combines server-side and device-side data for training models in a privacy-aware setting, focusing on binary classifiers. Model training occurs locally, and updates are sent to a server in a Trusted Execution Environment (TEE) for secure aggregation.	By keeping user data on devices and applying differential privacy, the architecture protects user privacy while allowing model training at scale.	Federated learning on mobile devices is significantly slower than centralized training, leading to delayed model development cycles.
Adversarial Training in	The paper explores adversarial training (AT)	The introduction of	The dynamic scheduling

TITLE	*METHODOLOGY*	*STRENGTHS*	*LIMITATIONS*
Communication Constrained Federated Learning [33]	within a federated learning (FL) framework to enhance the robustness of CNN models against adversarial attacks under communication constraints and non-IID data conditions.	FedDynAT effectively balances communication overhead and model accuracy, making it a practical solution for federated learning with adversarial training.	mechanism of FedDynAT adds complexity to the training process, which might make implementation challenging in large-scale or resource-constrained environments.
Federated Cross Learning for Medical Image Segmentation [34]	The paper proposes Federated Cross Learning (FedCross) to address challenges in federated learning (FL) for medical image segmentation on non-IID data.	FedCross effectively tackles the non-IID problem by avoiding model aggregation, leading to improved model performance on distributed medical data.	A potential weakness of FedCross is catastrophic forgetting, as the sequential training approach may cause the model to underperform on datasets it has not encountered for some time.
BFV-Based Homomorphic Encryption for Privacy-Preserving CNN Models [21]	The paper proposes a privacy-preserving framework combining Convolutional Neural Networks (CNNs) with BFV-based homomorphic encryption in a federated learning (FL) setting.	The use of BFV-based homomorphic encryption ensures that sensitive data and model updates are never exposed, providing robust privacy guarantees in federated learning.	Homomorphic encryption is computationally expensive, especially when applied to large CNN models, resulting in slower training and higher resource consumption.

TITLE	*METHODOLOGY*	*STRENGTHS*	*LIMITATIONS*
Federated Learning and Differential Privacy for Medical Image Analysis [36]	Combines federated learning (FL) and differential privacy (DP) to train models for histopathology image classification, particularly lung cancer subtypes, across multiple hospitals without sharing data. Uses FedAvg for decentralized training and differentially private stochastic gradient descent (DP-SGD) to ensure privacy, clipping gradients and adding Gaussian noise.	Demonstrates that FL with DP can match centralized training performance while providing strong privacy guarantees. Effectively handles real-world medical image data, allowing collaboration without direct data sharing between hospitals.	DP-SGD is sensitive to hyperparameters, requiring careful tuning to maintain both privacy and performance. Performance decreases when data distributions differ significantly between hospitals.
Federated Model Distillation with Noise-Free Differential Privacy [16]	The paper proposes a novel federated model distillation framework called FEDMD-NFDP, which utilizes a noise-free differential privacy (NFDP) mechanism to ensure privacy without adding noise. The methodology involves a two-step process where each party first digests a public dataset and then revisits their private data for further refinement.	Achieves differential privacy without the drawbacks of noise addition. The NFDP mechanism allows for effective knowledge sharing among parties while maintaining strong privacy guarantees.	NFDP may still be susceptible to biases introduced by the public dataset used for model updates. Performance of the proposed method could vary based on the distribution of the public and private datasets.
Federated Learning of Gboard Language Models with Differential Privacy [17]	Utilizes federated learning (FL) with differential privacy (DP) to train Gboard language models (LMs) for next-word prediction and other features.	Successfully deploys more than 20 Gboard models with formal DP guarantees, providing	Hyperparameter tuning and achieving optimal privacy-utility trade-offs are still challenging, especially with

TITLE	*METHODOLOGY*	*STRENGTHS*	*LIMITATIONS*
	Implements DP-Follow-the-Regularized-Leader with adaptive clipping to limit user contribution and improve privacy guarantees.	strong user data protection without sacrificing model utility.	large-scale client participation requirements.
FEDCONV: Enhancing Convolutional Neural Networks for Handling Data Heterogeneity in Federated Learning [32]	Modify CNNs with smooth activation functions (e.g., SiLU), reduce activation/normalization layers, and adopt larger kernel sizes for robustness in federated learning (FL) with heterogeneous data.	The novel FedConv CNN architecture improves accuracy by significant margins (e.g., 92.21% on COVID-FL, 54.19% on iNaturalist).	The proposed architectural modifications, like larger kernel sizes and overlapping convolutions, could introduce higher computational costs, particularly in resource-constrained settings.
Federated Transfer Learning with Differential Privacy [19]	The paper introduces a federated transfer learning framework that utilizes federated differential privacy (FDP) to enhance learning from multiple heterogeneous data sources while ensuring privacy.	The proposed FDP framework provides robust privacy guarantees without needing a trusted central server, making it suitable for real-world applications where data privacy is paramount.	The study acknowledges that the performance of the proposed methods may degrade when the source data sets are significantly dissimilar from the target data set, potentially leading to negative transfer effects.
Local Differential Privacy-Based Federated	The paper proposes a hybrid approach that combines federated learning (FL) with local	The introduction of multiple LDP mechanisms	While the proposed methods improve upon existing

TITLE	*METHODOLOGY*	*STRENGTHS*	*LIMITATIONS*
Learning for Internet of Things [9]	differential privacy (LDP) to enhance privacy in Internet of Things (IoT) applications. It introduces four LDP mechanisms, which offers multiple output possibilities to improve accuracy while minimizing communication costs.	enhances flexibility and performance across varying privacy budgets.	LDP techniques, they may struggle with scalability as the number of connected devices increases.
Federated Visual Classification with Real-World Data Distribution [20]	The paper introduces a Projected Federated Averaging (PFA) approach that utilizes heterogeneous differential privacy (DP) to enhance model utility while ensuring privacy.	Innovative aggregation strategy that effectively balances privacy and utility by leveraging the strengths of both public and private client updates.	PFA may still be vulnerable to biases introduced by the influence of “public” clients’ updates, which could dominate the model training process.
Unlocking High-Accuracy Differentially Private Image Classification through Scale [35]	The paper employs differentially private stochastic gradient descent and explores the interplay between noise, batch size, compute budget, and learning rate to optimize performance while maintaining privacy guarantees.	A significant strength of the paper is its ability to unlock high-accuracy image classification under DP constraints, achieving state-of-the-art results on academic datasets.	Applying it to real-world sensitive data involves additional considerations regarding privacy budgets and potential accuracy reductions for under-represented subgroups.
Projected Federated	The paper introduces Projected Federated	The use of PFA+	A key weakness is the potential

TITLE	*METHODOLOGY*	*STRENGTHS*	*LIMITATIONS*
Averaging with Heterogeneous Differential Privacy [10]	Averaging (PFA), which optimizes federated learning (FL) by addressing the challenge of heterogeneous privacy budgets.	significantly reduces communication costs – achieving over 99% communication reduction for private clients – without sacrificing much model accuracy.	bias from public clients, whose updates may dominate the global model and impact fairness.

10.9 Conclusion

In conclusion, FL offers a relatively promising framework for decentralized ML, ensuring privacy by avoiding the need for raw data sharing. However, the challenges of adversarial attacks, privacy leakage, and data heterogeneity present significant obstacles. This chapter has examined the potential solutions, particularly DP and meta-learning, which show prospects in enhancing both security and adaptability. The integration of DP protects sensitive information through noise introduction, while meta-learning offers adaptability to diverse client data. Further, in the domain of CNNs for image classification, these techniques can pave the way to help balance privacy and adaptability needs. However, there still remains uncertainty in maintaining these factors along with desirable model accuracy simultaneously for CNN classifications. In the future, a balance between security, adaptability, and performance will be salient. This highlights the need for further research to optimize application points of DP and meta-learning together in FL architectures and algorithms in the context of CNNs.

References

1. Yu, F., Lin, H., Wang, X., Garg, S., Kaddoum, G., Singh, S., & Hassan, M. M. (2023). Communication-efficient personalized federated meta-learning in edge networks. *IEEE Transactions on Network and Service Management*, *20*(2). https://doi.org/10.1109/TNSM.2023.3263831⏎
2. Zhou, J., Wu, N., Wang, Y., Gu, S., Cao, Z., Dong, X., & Choo, K-K. R. (2022). A differentially private federated learning model against poisoning attacks in edge computing. *IEEE Transactions on Dependable and Secure Computing*,

20(3), 1941–1941. https://doi.org/10.1109/TDSC.2022.3168556; Learning on Heterogeneous Data. *AISTATS Conference*, Vol. 151.⏎

3.Anonymous. (n.d.). Fedsecurity: A benchmark for attacks and defenses in federated learning and federated LLMS. In *Under review as a conference paper at ICLR 2024*.⏎

4.Dong, F., Ge, X., Li, Q., Zhang, J., Shen, D., Liu, S., Liu, X., Li, G., Wu, F., & Luo, J. (2023). PADP-FedMeta: A personalized and adaptive differentially private federated meta learning mechanism for AIoT. *Journal of Systems Architecture*, *134*, Art. 102754. https://doi.org/10.1016/j.sysarc.2022.102754⏎

5.Huang, A., & WeBank AI Lab. (2020). Dynamic backdoor attacks against federated learning. *arXiv*.⏎

6. Dwork, C. (2006). Differential Privacy. *International Colloquium on Automata, Languages, and Programming (ICALP)*, vol. 2006, pp. 1–12. https://link.springer.com/chapter/10.1007/11787006_1.⏎

7.Truong, N., Sun, K., Wang, S., Guitton, F., Data Science Institute, Imperial College London, Department of Computer Science, & Hong Kong Baptist University. (2021). Privacy preservation in federated learning: An insightful survey from the GDPR perspective. *Computers & Security*, *110*, 102402. https://doi.org/10.1016/j.cose.2021.102402⏎

8.Fu, J., Hong, Y., Ling, X., Wang, L., Ran, X., Sun, Z., Hui Wang, W., Chen, Z., & Cao, Y. (2024). Differentially private federated learning: A systematic review. *arXiv*, 2405.⏎

9.Zhao, Y., Zhao, J., Yang, M., Wang, T., Wang, N., Lyu, L., Niyato, & Lam. (2021). Local differential privacy-based federated learning for internet of things. *IEEE Internet of Things Journal*, *8*(11). https://doi.org/10.1109/JIOT.2020.3037194⏎

10.Liu, J., Lou, J., Xiong, L., Liu, J., & Meng, X. (2022). Projected federated averaging with heterogeneous differential privacy. *PVLDB*, *15*(4), 828–840). https://doi.org/10.14778/3503585.3503592⏎

11. Karimireddy, S., et al. (2020). SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. *Proceedings of the 37th International Conference on Machine Learning*, vol. 119, edited by Hal Daumé III and Aarti Singh, PMLR, 2020, pp. 5132–5143. https://proceedings.mlr.press/v119/karimireddy20a.html.⏎

12. Li, T., et al. (2020). Federated Optimization in Heterogeneous Networks. *arXiv*. https://arxiv.org/abs/1812.06127.⏎

13. Triastcyn, A., & Faltings, B. (2019). Federated Learning with Bayesian Differential Privacy. In *2019 IEEE international conference on big data (Big Data)*, IEEE, Dec. 2019, pp. 2587–2596. https://doi.org/10.1109/BigData47090.2019.9005465⏎

14.Noble, M., Bellet, A., & Dieuleveut, A. (2022). Differentially private federated learning on heterogeneous data. In *International conference on artificial intelligence and statistics*, pp. 10110–10145. PMLR.⏎

15.Stojkovic, B., Woodbridge, J., Fang, Z., Cai, J., Petrov, A., Iyer, S., Huang, D., Yau, P., Kumar, A. S., Jawa, H., Guha, A., & Meta. (2022). Applied federated learning: Architectural design for robust and efficient learning in privacy aware settings. *arXiv*.⏎

16.Sun, L., Lyu, L., Lehigh University, & Ant Group. (2021). Federated model distillation with noise-free differential privacy. *arXiv* [Report]. https://arxiv.org/abs/2009.05537v2⏎

17.Xu, Z., Zhang, Y., Andrew, G., Choquette-Choo, C. A., Kairouz, P., McMahan, H. B., Rosenstock, J., Zhang, Y., & Google. (2023). Federated learning of Gboard language models with differential privacy. *arXiv*, 2305.⏎

18.Zhang, X., Li, C., Han, C., Li, S., Feng, Y., Wang, H., Cui, Z., & Gryllias, K. (2024). A personalized federated meta-learning method for intelligent and privacy-preserving fault diagnosis. *Advanced Engineering Informatics, 62*, Art. 102781. https://doi.org/10.1016/j.aei.2024.102781⏎

19.Li, M., Tian, Y., Feng, Y., Yu, Y., Department of Statistics, University of Warwick, Department of Statistics, Columbia University, & Department of Biostatistics, School of Global Public Health, New York University. (2024). Federated transfer learning with differential privacy. *arXiv*, 2403.⏎

20.Hsu, T.-M. H., Qi, H., & Brown, M. (2020). Federated visual classification with real-world data distribution. *arXiv*. https://arxiv.org/abs/2003.08082v3⏎

21.Wibawa, F., Catak, F. O., Sarp, S., & Kuzlu, M. (2022). BFV-Based homomorphic encryption for privacy-preserving CNN models. *Cryptography*, *6*(3), 34. https://doi.org/10.3390/cryptography6030034⏎

22.Jeong, H., Son, H., Lee, S., Hyun, J., & Chung, T-M. (2024). FedCC: Robust federated learning against model poisoning attacks. *IEEE Access*, *1*(991), 20.⏎

23.Jiang, Y., Kone ˇcn´Y, J., Rush, K., & Kannan, S. (2023). Improving federated learning personalization via model agnostic meta learning. *arXiv*, 1909.⏎

24.Hanzely, F., Richtárik, P., & King Abdullah University of Science and Technology. (2021). Federated learning of a mixture of global and local models. *arXiv*.⏎

25.Finn, C., Abbeel, P., Levine, S., University of California, Berkeley, & OpenAI. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning: Vol. PMLR 70* [Conference proceeding].⏎

26.McMahan, H. B., Moore, E., Ramage, D., Hampson, S., & Google, Inc. (2017). Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017* (Vol. 54). https://arxiv.org/pdf/1602.05629.pdf (Original work published 2023)⏎

27.Madhusudhan, H. S., Gupta, P., Saini, D. K., & Tan, Z. (2023). Dynamic virtual machine allocation in cloud computing using elephant herd optimization scheme. *Journal of Circuits, Systems and Computers*, *32*(11), Art. 2350188.⏎

28. Rawat, P. S., Gaur, S., Barthwal, V., Gupta, P., Ghosh, D., Gupta, D., & Rodrigues, J. J. C. (2025). Efficient virtual machine placement in cloud computing environment using BSO-ANN based hybrid technique. *Alexandria Engineering Journal*, *110*, 145–152.
29. HS, M., & Gupta, P. (2024). Federated learning inspired Antlion based orchestration for Edge computing environment. *PLoS One*, *19*(6), Art. e0304067.
30. Gupta, P., Anand, A., Agarwal, P., & McArdle, G. (2024). Neural network inspired efficient scalable task scheduling for cloud infrastructure. *Internet of Things and Cyber-Physical Systems*, *4*, 268–279.
31. HS, M., Gupta, P., & McArdle, G. (2023). A Harris Hawk Optimisation system for energy and resource efficient virtual machine placement in cloud data centers. *PLoS One*, *18*(8), Art. e0289156.⏎
32. Xu, P., Wang, Z., Mei, J., Qu, L., Yuille, A., Xie, C., & Zhou, Y. (2023). FEDCONV: Enhancing convolutional neural networks for handling data heterogeneity in federated learning. *arXiv:2310.04412v1 [cs.CV]* 6 Oct 2023, 1. https://arxiv.org/abs/2310.04412v1⏎
33. Shah, D., Dube, P., Chakraborty, S., Verma, A., & Department of Computer Science, University of Illinois, Urbana Champaign, & I.B.M T.J. Watson Research Center. (2021). Adversarial training in communication constrained federated learning. *arXiv*.⏎
34. Federated cross learning for medical image segmentation. (2023). In *Proceedings of Machine Learning Research* (pp. 1–12). https://github.com/DIAL-RPI/FedCross⏎
35. De, S., Berrada, L., Hayes, J., Smith, S. L., & Balle, B. (2022). Unlocking high-accuracy differentially private image classification through scale. *arXiv: 2204.13650v2* (pp. 1–2). https://arxiv.org/abs/2204.13650v2⏎
36. Adnan, M., Kalra, S., Cresswell, J. C., Taylor, G. W., & Tizhoosh, H. R. (2022). Federated learning and differential privacy for medical image analysis. *Scientific Reports*, *12*(1). https://doi.org/10.1038/s41598-022-05539-7⏎
37. Mustapha, S. D. S., & Gupta, P. (2024). DBSCAN inspired task scheduling algorithm for cloud infrastructure. *Internet of Things and Cyber-Physical Systems*, *4*, 32–39.⏎
38. Gupta, P., Rawat, P. S., Kumar Saini, D., Vidyarthi, A., & Alharbi, M. (2023). Neural network inspired differential evolution based task scheduling for cloud infrastructure. *Alexandria Engineering Journal*, *73*, 217–230.
39. Madhusudhan, H. S., Gupta, P., Saini, D. K., & Tan, Z. (2023). Dynamic virtual machine allocation in cloud computing using elephant herd optimization scheme. *Journal of Circuits, Systems and Computers*, *32*(11), Art. 2350188.
40. Rawat, P. S., Gaur, S., Barthwal, V., Gupta, P., Ghosh, D., Gupta, D., & Rodrigues, J. J. C. (2025). Efficient virtual machine placement in cloud

computing environment using BSO-ANN based hybrid technique. *Alexandria Engineering Journal*, *110*, 145–152.⏎

Chapter 11

EP-MPCHS: Edge Server-Based Cloudlet Offloading Using Multi-Core and Parallel Heap Structures

Rajkumar Nagulsamy, Manisha Gupta, and Punit Gupta

DOI: 10.1201/9781003610168-11

11.1 Introduction

In today’s landscape of cloud computing, mobile technologies have started utilizing applications with high computational needs. Hence, task–resource management has become a crucial research domain for mobile resource offloading. Throughout the last ten years, cloud computing has emerged as one of the most appealing alternatives for hosting applications via the Internet infrastructure. Computing in the cloud is based on the use of centralized computing resources located in data centers. This practice alleviates the burden of infrastructure administration and enables service providers to provide cloud services at reasonable costs for both people and organizations.

The role of cloud computing as a catalyst for digital transformation has led to mobile technologies having a multitude of use cases in recent times, including augmented reality, designing mobile applications, cross-platform gaming, and so on. These scenarios continue to raise the standard in terms of how people utilize software programs in their day-to-day lives. With the emergence of these software solutions, it has become apparent that the concentration of cloud computing to data centers around a small number of global large-scale infrastructures (like Google, Azure, and Amazon) has resulted in the setup facing latency issues due to geographical distance from end devices. This distance causes reaction times that are incompatible with the real-time needs of the applications in question.

Therefore, the use of edge computing and its seminal role in offloading mobile application task computations is unavoidable. Cloud-native applications like Amazon Web Services (AWS) and Google Cloud Platform (GCP) for task offloading require high latency and bandwidth and are constrained to the unforeseen case of service unavailability. The usage of Cloudlet or a small-scale computationally viable resource center available at the edge of the network peripheral zone is the more plausible option. The Virtual machine (VM)-based cloudlet offloading proposed in this study takes advantage of small-scale resource management procedures, rather than high-scale centralized cloud computing data center environments like AWS or GCP.

The distributed computation-based system leverages resources by creating a highly available cloudlet processing application using edge-scale devices by offloading in real time. The procedure highlights the enhanced high computing tasks to be split in computational parallel pipelines, which can be performed across edge devices as a form of device-to-device (D2D) cloudlet offloading, and the final output can be computed in a centralized fashion at the remote cloud through highly available offloading techniques. The availability of data sources nearby is the main characteristic of the edge-based cloudlet computing mechanism (ECM) technique. ECM facilitates the diversion of tasks between high-scale system units to AWS or GCP and edge-based offloading options. ECM infrastructure is capable of reducing carbon emissions from high computation needs of the task/resource allocation for mobile units as well as saving cost and decreasing latency for wide-scale VM infrastructure.

The ECM mechanism is capable of reducing the cost and latency extensively; however, it consists of two important factors:

1. **Placement**: The inability to place the tasks in the best-suited edge computing device during D2D offloading, this is crucial as without appropriate placement the tasks could witness high latency and a generic decrease in bandwidth. Moreover, the proper utilization of ECM devices is ensured by appropriate placement algorithms.
2. **Migration**: Further with the low-scale computational issues, starvation from the processes aligned across for offloading is a normalized phenomenon to reduce the inherent cloudlet or process starvation. Thus, a need for continuous migration is essential.

The presented study utilizes the Edge SimPy simulation network for the technique utilization and formulation of placement and migration algorithms. The study therefore proposes three things:

1. The formulation of a priority-based multi-core heap structure to prioritize the cloudlets based on task/resource allocation that allow a reduction of latency and increased bandwidth.
2. An objective function toward the use of both the cloudlets and edge server where the most optimal task or cloudlet is allocated to the edge server which can just match the requirements; thus, resource utilization is maximized.
3. The need to introduce parallel computation using threads and multi-latent systems to reduce process starvation in collaboration with the migration algorithm.

11.2 Literature Review

Mobile edge computing (MEC) technique about cloudlet offloading offers to overcome issues of high computational needs by mobile applications and enhance the storage for low latency and quick response application-centric designs. However, the real-time offloading poses a challenge toward placement and migration of systems cloudlet across edge resources. To overcome this challenge, various research studies have advocated the use of deep learning algorithms to improve the offloading process in MEC networks. The solutions can be broadly categorized into two paradigms: intelligent offloading mechanism discussed in section 2.1 and task or cloudlet offloading using deep reinforcement learning discussed in Section 2.3.

11.2.1 Mobile Devices and Its Limitations

The primary challenge to real-time offloading MEC as discussed by Alghamdi et al. (2021) [17] is the computation deficit within mobile networks for offloading MEC. Cardellini et al. (2016) [18] propose a game theory solution to offload D2D cloudlets with central cloud and MEC architecture. Patel et al. (2015) [19] emphasize network congestion and the use of MEC as a key technocratic use toward 5G systems as it allows faster network flow and could lead to network drops reducing congestion. This is critical as without adequate placement the tasks might observe excessive latency and an overall drop in bandwidth. Moreover, the effective usage of ECM devices is assured by suitable placement algorithms. The low computational capabilities lead to fewer resources for utilization to fulfill each process, which in turn creates process starvation, leading to high latency, high data transfer, and low bandwidth. This is due to low-scale computing resources and process starvation caused by the offloading of the cloudlet process. Thus, a requirement for ongoing migration is needed.

The necessity of transforming data with the help of computation requires high-end server with cloud computing capabilities. The research discussed above showcases the absence of high computation and storage capacity solutions to offload cloudlet tasks (RG1).

11.2.2 Intelligent Offloading Mechanism

Xu et al. (2018) propose a deep neural mix of convolution and recurrent neural networks to create a hybrid deep reinforcement algorithm for an intelligent offloading system. The convolutional recurrent neural network (CRNN) is used for offloading process in MEC networks. The proposed techniques offer a 25% lower energy use case in comparison to traditional offloading algorithms.

Yu et al. (2022) [10] address the delay in computing resources and high latency constraints for MEC. They propose a Duel Deep Q network to optimize the scheduling of resource allocation for a multiuser, multicluster, and multi-tenant environment. They further propose an energy-efficient, loss-specific optimization algorithm that reduces energy use by 11.54% and 20.83%, respectively, for D2D and generic cloud scenarios. Furthermore, Vaiswan et al. (2016) [11] propose a solution for the binary system increased latency and decreased bandwidth problem. The study discusses the pitfalls of the existing Monte Carlo distribution method and proposes stochastic enumeration using decision tree-based cost optimization, offering an intelligent pursuit to effective sampling and placement of jobs based on computing resource parameters.

The above research showcases a significant research gap toward the concerns of congestion packet loss, process starvation, transport security, and latency threats (RG2, refers to Section 11.2.3.1).

11.2.3 Cloudlet Offloading to Edge Cloud Networks Using Deep Reinforcement Learning

Xuzehu et al. (2021) [12] propose an intelligent deep reinforcement learning (DRL) method for resource allocation toward cloudlet offloading for D2D. This study investigates the resource allocation issue that occurs during the offloading process. The MEC-oriented real-time system are energy-aware offloading system when considering the battery capacity of the data center. The paper showcases energy savings, close to $10^4 \times 10^4$ Joules. The study is nuanced over the MATLAB platform for the execution of the technique, providing a deep weighted parameter for values 0.2, 0.4, 0.6, 0.8, and 1.0. The study offers an offloading strategy for optimization, it takes the shortest latency and the cheapest computing cost as the optimization aim and then applies Q-learning to solve the problem.

Similarly, Ullah et al. (2023) [13] studied edge computing for offloading from centralized cloud servers to D2D devices. It proposes a DRL algorithm for offloading cloudlets based on available computation resources and device location. The study uses the Markov decision process to model the problem of latency and bandwidth in the underlying centralized cloud server system. The study proposed DDQEC, a Q learning-based edge computing resource allocation algorithm using DRL.

The study by Mourita Mozib et al. (2023) [14] works forth to increase the computation capabilities for mobile applications, high bandwidth and low latency. The paper identifies that MEC offloading requires optimization of offloading decisions in real-time scenario. To achieve the desired goal the paper leverages the attribute of function from computation from edge computing networks. The distributed deep learning-based system for real-time offloading in MEC networks has tremendous potential to improve performance, scalability, and efficiency. The technique achieves a reduced energy utility of up to 40% and a 60% increase in latency across different simulations.

The critical concern that arises from the discussion of the above paper is that there is an apparent need to create a priority-based system capable of reducing network flow, enhancing bandwidth, and reducing latency. The current research uses a simplistic process to offload cloudlets leading to process starvation, and high latency, thus requiring high computation and storage capacity solutions to offload cloudlet tasks (RG1, refers to Section 11.2.3.1).

11.2.3.1 *Research Gaps*

RG1: Need for high computation and storage capacity solution to offload cloudlet tasks from mobile applications that have low processing power and battery life to support such high computational need.

RG2: MEC network works extensively on wireless devices; thus, congestion packet loss, transport security, and latency threats are quite common in this domain. Thus, centralized cloud systems often tend to cause significant obstruction to service or the quality of offerings. Therefore, the use of edge computing for cloudlet remediation is essential.

RG3: In the context of RQ1 and RQ2, it’s important to formulate a placement and migration algorithm capable of effectively reducing energy need, latency, and increasing bandwidth on ECM processes.

11.3 Methodology

The methodology describes the simulation pipeline followed throughout the study. It delves into the use case of various physical layer attributes in Section 11.3.1. In Section 11.3.2 we describe the use case and the ability of the placement and migration algorithm. Section 11.3.3 describes the proposed hybrid min heap queue placement creator. Figure 11.1 describes the high-level overview of the functionalities of the proposed Edge SimPy simulation. It consists of a variety of steps from the availability of simulation 1 and simulation 2, which is passed onto the simulator with the initializer path. It creates monitoring components such as the Edge Server, Network Switch, Base Station, and User data. The figure further showcases the placement and migration algorithms with experimental arbitrary parameters *a* and *b*, where *a* can take values between 1, 2, 3 and *b* converges over 100, 200, 300. In the final stage, monitoring analyzes various data and metrics across the network flow.

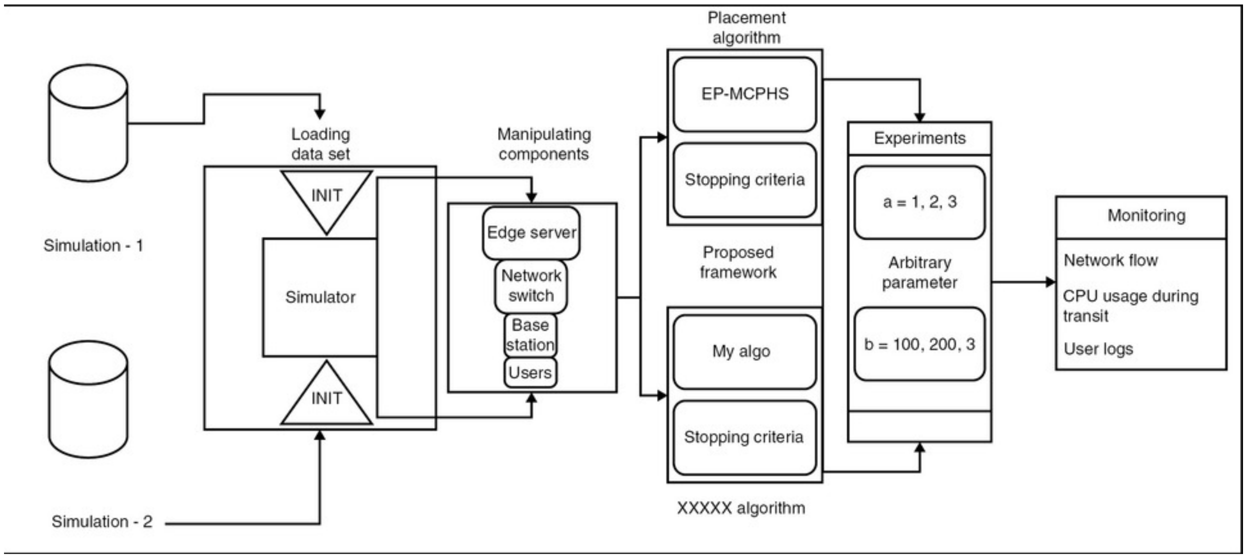


Figure 11.1 High-level diagram.⏎

11.3.1 Physical Layer and Attributes

The physical layer pertains abstraction necessary for maintaining users and resources defined below. Each component is provided with a piece of unique geospatial information which helps in locating and creating a two-dimensional map of the same. Especially attributes of edge server and base station.

11.3.1.1 *Base Station*

Base station is an integral and inherent component placed within the EdgeSimPy simulation orchestration. It acts as a service platform allowing the users and edge servers to communicate via wireless communication using Transmission Control Protocol (TCP). It is typically expected to cover the entire map arena allowing the users to stay connected at all locations and geospatial coordinates of the mapping. It is responsible for capturing energy/power-based inputs and wireless latency of the orchestration grid. Further, it involves the presence of network switches and edge servers. Network switches are essential for wired connectivity with the users, whereas edge servers allow cloudlet offloading or hosting abilities for the orchestration to perform smoothly.

11.3.1.2 *Network Switches*

Network switches act as a traffic flow originator. It provides wired connectivity for hosting infrastructure components and managing flow. It consists of ports, delays, clocks, bandwidth, etc. The network switch utilizes the placement algorithm. It will use the hybrid proposed algorithm to place the cloudlets on the edge cloud orchestration and computation servers. It secures and protects the data transfer, network flow, and user requests as per QoS policies.

11.3.1.3 *Edge Servers*

Edge servers are scaled orchestration devices available for cloudlet offloading and computational adaptation. Each edge server consists of CPU cores, RAM, disk space, and MIPS (Million of instructions per seconds) performance is responsible for scaling and processing cloudlet offloading and co-hosted application runtime. It allows round-robin processing of each cloudlet in the given time frames and onboards the highest priority of the cloudlet first. Further to save power consumption which inherently affects the latency of the node, it encompasses with decision to shut down the server to reduce power utilization.

11.3.1.4 *Users*

Users are defined as client machines that offload tasks to the edge servers using the network switches. Users consist of geospatial information, which is the coordinate of the originating request, and the user class defines delay in response from the edge servers, starvation of cloudlet, etc. Further, the user access pattern creates a cloudlet to be offloaded to the edge servers. The mobility pattern recognizes random pathways for the users.

11.3.1.5 *Simulation Framework*

To achieve the simulation of edge server cloudlet offloading, this study reviews various possible scenarios to compare the algorithms proposed. It utilizes a Python-based framework to develop the algorithm rather than creating a new simulation architecture in the interest of time and computation necessary for testing. The simulation framework adopted for this study is EdgeSimPy. It is a Python-based edge server simulation framework architecture that allows the simulation to be aided with pre-existing simulation data sets consisting of base stations, network switches, edge servers, and users. The simulation aids in the monitoring of data flow, resource utilization, bandwidth of the edge servers, and other parameters for comparison of the placement algorithm. SimPy which is an abstract class over EdgeSimPy is a generic tool based on Python for simulation of different parameters. It is generally used for resource utilization simulation and EdgeSimPy is a specific use case of SimPy for edge server and cloudlet offloading scenarios.

11.3.2 Continuous Integration of Resource Allocation (CIRA)

CIRA ensures that the processes are optimally offloaded onto the edge servers. The D2D and cloudlet offloading are synchronized for the cloudlet to utilize maximum resources, thus ensuring the absence of underutilization of resources. CIRA is also responsible for reducing starvation for processes within an edge server orchestration setup.

11.3.2.1 *Network Architecture*

The network architecture utilizes the physical layer and attributes, namely, base stations, network switches, edge servers, users, and simulation framework to create a flow architecture diagram. It allows two workflows, each consisting of services or cloudlets for offloading. Figure 11.2 describes the architectural setup of the complete orchestration. It connects the ES instances, that is, the edge servers, S1, S2, S3, that is, the services using the network switches (L1, L2, L3,… L8). The CIRA is divided into two segments, namely, placement and migration algorithms.

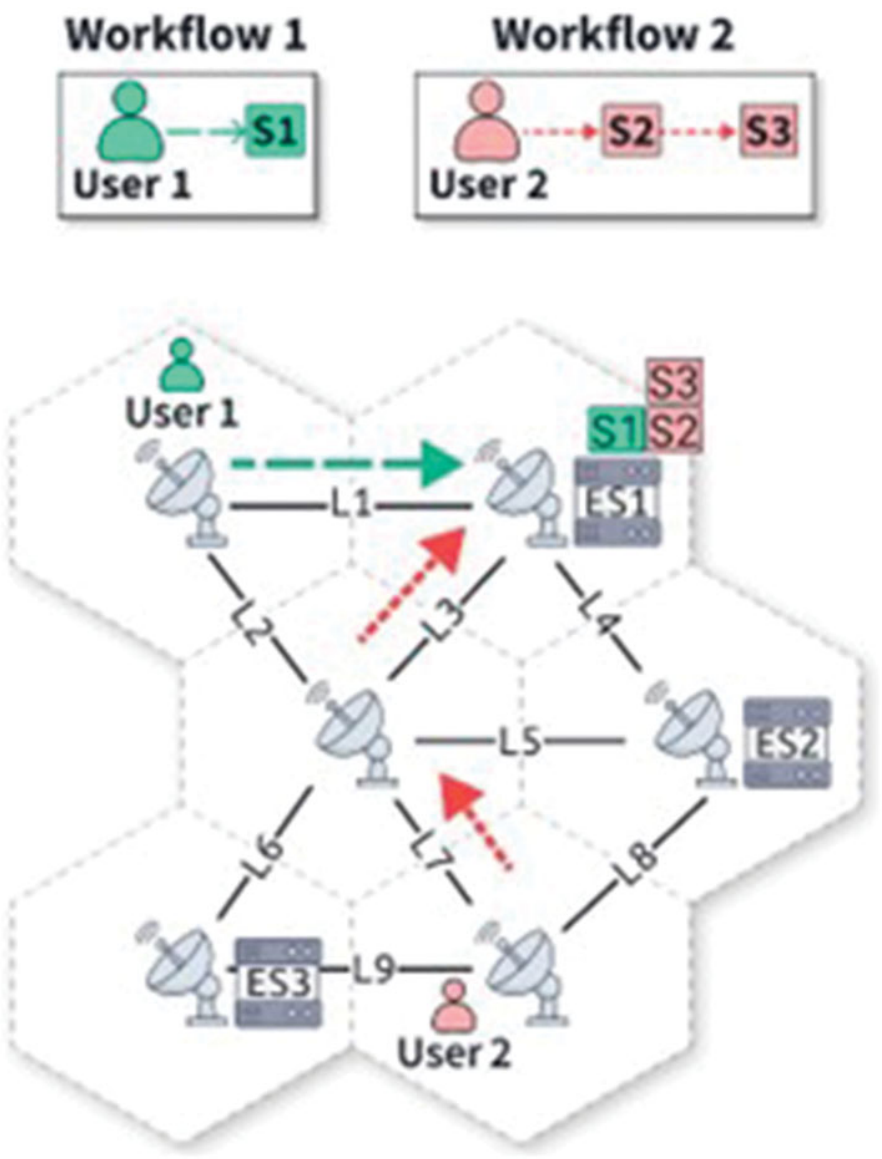


Figure 11.2 Network architecture diagram.⏎

11.3.2.2 *Placement Algorithm*

The EdgeSimPy framework is built with the resource allocation system mechanism consisting of multiple units. The placement algorithm is the rudimentary backbone of the resource allocation unit as it is responsible for placing the cloudlets from the respective user requests to the edge servers with the help of the network switches in transit. It is essential in optimizing the performance and resource utilization of the edge servers and orchestration as a whole. The placement algorithm is based on cloudlets described in Equation 11.1, edge servers described in Equation 11.2, where R_{ci} defines the resource of services i, C_{ej} defines the capacity of edge servers, and $L_{ci,ej}$ defines the latency in between.

(11.1)

(11.2)

(11.3)

(11.4)

Equation 11.3 defines the minimum capacity needed for each service defined by X_{ij}. Equation 11.4 defines that each service must be placed only on one edge server to prevent any redundant tasks.

11.3.2.3 *Migration Algorithm*

The migration algorithm allows continuous integration of Equations 11.3 and 11.4 by re-evaluating starvation parameter and combines the possibility of task computation based on the dynamic mappings available and rebalances the load on each edge server; in simpler terms, it can be viewed as a rebalancing agent and the placement algorithm can be demonstrated as a load balancing unit. Placement algorithm is necessary for load distribution in the inception of task allocation. However, the migration technique is a constant running service that rebalances as and when needed.

The migration algorithm ensures continuous integration and continuous processing for the cloudlets offloaded to the edge servers. It does so by optimizing the time benefits. The migration algorithm is an optimization technique that tries to minimize Equation 11.5.

(11.5)

Equation 11.5 describes the optimization function with or the migration cost of moving cloudlet *i* to edge server *J*, and is the starvation parameter.

(11.6)
Equation 11.6 describes the global cloudlet offloading parameter which should always be 1, as each cloudlet must be placed at an edge server.

(11.7)
Equation 11.7 defines that within the EdgeSimPy simulation mechanism, the migration strategies allow seamless transition of cloudlets between edge servers which is essential for service continuity while enabling continuous integration and continuous processing.

11.3.3 Edge Priority Placement Using Multi-Core and Parallel Heap Structures (EP-MCPHS)

EP-MCHPS combines multi-latent processing to benefit from the degeneracy minimum optimization unit of the heap function. The algorithm is divided into three segments: Parallel heap creation (PhC), Provisioning, and PhC combination with parallel processing.

11.3.3.1 *Parallel Heap Creation for Priority-Based Resource Allocation*

A priority queue is utilized as a provisioning mechanism for the minimum optimizing which implements a minimum heap to allow the least possible resource to be allocated to the cloudlet, ensuring the least wastage of resources. The PhC algorithm uses Equation 11.8. The heap used is a min heap for the edge server section based on the cloudlet computational resource needs.

(11.8)

Equation 11.8 describes the heap function which is responsible for priority-based distinction of resource allocation. The function notes the minimum required edge server that can successfully offload the cloudlet. This allows the edge servers to be optimized, the resource utilization to be maximized, and the migration flow rate of data to be low as the edge servers would be working at maximum capacity in comparison to the MinMax fairness algorithm. Thus, the priority-based allocation not only reduces latency, and data flow rate, and maximizes resource optimization and bandwidth, it also caters to creating a high operational edge server unit that can cater to on-demand service requests based on resource priority task allocation.

11.3.3.2 *Provisioning*

The provisioning algorithm verifies the ability of an edge server to cater to the cloudlet that is being sent. It compares the

(11.9)
which in turn creates the function given by Equation 11.10. These functions help compare the parameters given by the CPU or, RAM or, and disk space.

(11.10)

11.3.3.3 *PhC Combination with Multi-Core Parallel Processing*

Parallel processing allows the PhC to compute faster and more optimally. Equation 11.11 presents the parallel threads set available.

(11.11)
Equation 11.12 showcases how each thread operates on a different cloudlet where is a partition of cloudlets C.

(11.12)
The time complexity of PhC combination with parallel processing is given by Equation 11.13

(11.13)
The proposed algorithm optimizes the placement algorithm by reducing the starvation of unused resources by using a Min heap with resource optimization. Further on, it encompasses a parallel processing domain for the reduction of time computation, enabling fast retrieval of cloudlets into edge computing processing via degeneracy pipelines.

The multi-core parallel heap structure divides the unit of resources into smaller units and aims to fulfill as many as possible cloudlets in a single unit of time, whereas the heap emphasizes allowing a resource-based priority entry to the pipeline. It ensures that the process starvation of the cloudlet is reduced as it also has the migration technique to remap the starving cloudlets. The multi-core parallel processing caters to priority-based resource allocation as well as an automatic reduction in the starvation of cloudlets.
The proposed EP-MCPHS promises to reduce latency, and time spent to start processing, and enhance faster traffic flow based on the ability of a heap-based processing speeding system.

Algorithm 11.1 presents the algorithmic design of our placement algorithm.

ALGORITHM 11.1 EP-MCPHS

```
class EdgeServerWrapper: def init (self, edge_server): self.edge_server =
edge_server
   def __lt__(self, other):
   # Compare edge servers based on their combined available resources (CPU,
memory, disk) return (self.edge_server.cpu, self.edge_server.memory,
self.edge_server.disk) < \ (other.edge_server.cpu, other.edge_server.memory,
other.edge_server.disk)
   def provision_service(service, edge_server_heap): while edge_server_heap:
try:
   edge_server_wrapper = heapq.heappop(edge_server_heap) edge_server =
edge_server_wrapper.edge_server
   if edge_server.has_capacity_to_host(service=service):
service.provision(target_server=edge_server)
   logging.info(f"Service {service.id} provisioned on edge server {edge_server.id}")
return True except Exception as e: logging.error(f"Error provisioning service
{service.id} on edge server {edge_server.id}: {e}") logging.warning(f"No suitable
edge server found for service {service.id}") return False
   def my_algorithm(parameters):
   services = [service for service in Service.all() if service.server is None and not
service.being_provisioned] edge_servers = [EdgeServerWrapper(edge_server) for
edge_server in EdgeServer.all()] heapq.heapify(edge_servers)
   with ThreadPoolExecutor(max_workers=10) as executor:
   futures = [executor.submit(provision_service, service, edge_servers) for service in
services]
   for future in futures: try:
   future.result() except Exception as e: logging.error(f"Error in provisioning process:
{e}")
```

11.3.4 Testing and Evaluation Framework

This study makes a comparative analysis between the proposed EP-MCPHS and the Min-max fairness algorithm (MMFA). The MMFA was proposed in the base paper version of EdgeSimPy architecture. The study performs an in-depth analysis of the bandwidth, latency, resource utilization, and data flow rate across both techniques. The study further compares the mean and standard deviation under statistical analysis to showcase the evaluation of the EP-MCPHS architecture to the baseline MMFA. The study provides a detailed discussion of how the reduction in data transfer and enhanced latency are the output of the incorporated minheap-based task priority system. It also compares the results in the upcoming section under the bar plots and statistical analysis cycles. It also creates a simulation parameter comparison for intervals of 5, 10, 15, and 20-second cycles showcasing the sturdity of the architecture across increasing time ranges and upcoming cloudlets.

11.4 Results and Discussion

11.4.1 Evaluation Metrics

In this study, we primarily compare the results among Start, End, Actual bandwidth and Data transfer. Let's discuss in detail on the meaning and utilization of these.

Start and End time showcases the actual lapse of time to process the cloudlets. The edge server tries to minimize the difference between these as much as possible. The actual bandwidth refers to the potential to cater to more services, which should be as high as possible. This also showcases the low latency of the edge servers. Finally, the data flow is a paramount effect of having the most optimal cloudlet being placed allowing for low data flow, accommodating high bandwidth and low latency for the algorithm.

11.4.2 Edge Servers

Edge server details showcase the instances, coordinates, CPU demand, RAM demand, disk demand, and services (Table 11.1). These will be used for the below-mentioned simulations.

Table 11.1 Edge Server Details⏎

TimeStep	*Instance ID*	*Coordinates*	*CPU Demand*	*RAM Demand*	*Disk Demand*	*Services*
0	1	[0, 0]	0	0	0	[]
0	2	[0, 2]	0	0	0	[]
0	3	[6, 0]	0	0	0	[]
0	4	[1, 3]	0	0	0	[]
0	5	[7, 1]	1	1024	1017	[]
0	6	[6, 2]	0	0	0	[]
1	1	[0, 0]	6	12288	82	[]
1	2	[0, 2]	0	0	0	[]
1	3	[6, 0]	0	0	0	[]
1	4	[1, 3]	0	0	0	[]
1	5	[7, 1]	1	1024	1017	[]
1	6	[6, 2]	0	0	0	[]
2	1	[0, 0]	6	12288	82	[1, 2]
2	2	[0, 2]	0	0	0	[]
2	3	[6, 0]	0	0	0	[]
2	4	[1, 3]	0	0	0	[]
2	5	[7, 1]	1	1024	1017	[]
2	6	[6, 2]	0	0	0	[]
3	1	[0, 0]	6	12288	82	[1, 2]
3	2	[0, 2]	0	0	0	[]
3	3	[6, 0]	0	0	0	[]
3	4	[1, 3]	0	0	0	[]
3	5	[7, 1]	1	1024	1017	[]
3	6	[6, 2]	0	0	0	[]
4	1	[0, 0]	6	12288	82	[1, 2]

TimeStep	*Instance ID*	*Coordinates*	*CPU Demand*	*RAM Demand*	*Disk Demand*	*Services*
4	2	[0, 2]	0	0	0	[]
4	3	[6, 0]	0	0	0	[]
4	4	[1, 3]	0	0	0	[]
4	5	[7, 1]	1	1024	1017	[]
4	6	[6, 2]	0	0	0	[]
5	1	[0, 0]	6	12288	82	[1, 2]
5	2	[0, 2]	0	0	0	[]
5	3	[6, 0]	0	0	0	[]
5	4	[1, 3]	0	0	0	[]
5	5	[7, 1]	1	1024	1017	[]
5	6	[6, 2]	0	0	0	[]
6	1	[0, 0]	6	12288	82	[1, 2]
6	2	[0, 2]	0	0	0	[]
6	3	[6, 0]	0	0	0	[]
6	4	[1, 3]	0	0	0	[]
6	5	[7, 1]	1	1024	1017	[]
6	6	[6, 2]	0	0	0	[]
7	1	[0, 0]	6	12288	82	[1, 2]
7	2	[0, 2]	0	0	0	[]
7	3	[6, 0]	0	0	0	[]
7	4	[1, 3]	0	0	0	[]
7	5	[7, 1]	1	1024	1017	[]
7	6	[6, 2]	0	0	0	[]
8	1	[0, 0]	6	12288	82	[1, 2, 3, 4, 5, 6]
8	2	[0, 2]	0	0	0	[]
8	3	[6, 0]	0	0	0	[]
8	4	[1, 3]	0	0	0	[]
8	5	[7, 1]	1	1024	1017	[]
8	6	[6, 2]	0	0	0	[]

11.4.3 Simulation Results

This section presents the results obtained by the comparison of simulation dataset 2 (Table 11.2). User data with the statistical measures of count, mean, standard deviation, minimum, 25% quartile, 50% quartile, 75% quartile, and maximum, these are calculated across time steps and instance ID.

Table 11.2 User Data⏎

StatisticalMeasure	*Time Step*	*Instance ID*
count	54	54

StatisticalMeasure	*Time Step*	*Instance ID*
mean	4	3.5
std	2.60623	1.72386
min	0	1
25%	2	2
50%	4	3.5
75%	6	5
max	8	6

Network flow using first come first serve showcases the statistical measures across time stamp, instance ID, start, end time, source, target, bandwidth, and data transfer rate (Table 11.3).

Table 11.3 Network Flow Using Max-Min Fairness Algorithm⏎

Statistical Measure	*Time Step*	*Instance ID*	*Start*	*End*	*Source*	*Target*	*Actual Bandwidth*	*Data Tran*
count	31.000000	31.000000	31.000000	15.000000	31.0	31.0	31.000000	31.0
mean	4.612903	2.451613	1.225806	3.600000	5.0	1.0	3.811828	6.53
std	2.275631	1.120676	0.425024	2.898275	0.0	0.0	1.511640	8.10
min	1.000000	1.000000	1.000000	1.000000	5.0	1.0	2.000000	0.00
25%	3.000000	1.500000	1.000000	1.000000	5.0	1.0	2.041667	0.00
50%	5.000000	2.000000	1.000000	1.000000	5.0	1.0	4.166667	1.87
75%	6.500000	3.000000	1.000000	6.500000	5.0	1.0	4.687500	12.8
max	8.000000	4.000000	2.000000	7.000000	5.0	1.0	6.250000	23.7

Similarly in Table 11.4 EP-MCPHS network flow showcases the network route changes in bandwidth, timestamp, and data transfer.

Table 11.4 EP-MCPHS Network Flow⏎

	Time Step	*Instance ID*	*Start*	*End*	*Source*	*Target*	*Actual Bandwidth*	*Data to Transfer*
count	42.000000	42.000000	42.0	26.000000	42.0	42.000000	42.000000	42.00000
mean	3.500000	4.000000	1.0	2.615385	5.0	2.714286	6.428571	4.916667
std	1.728527	2.024243	0.0	1.626700	0.0	1.686181	3.857320	7.548959
min	1.000000	1.000000	1.0	1.000000	5.0	1.000000	2.000000	0.000000
25%	2.000000	2.000000	1.0	1.000000	5.0	1.000000	2.000000	0.000000
50%	3.500000	4.000000	1.0	3.000000	5.0	2.000000	6.250000	0.000000
75%	5.000000	6.000000	1.0	4.000000	5.0	4.000000	6.250000	10.00000
max	6.000000	7.000000	1.0	5.000000	5.0	6.000000	12.500000	23.75000

Table 11.5 Comparison of EP-MCPHS and Max-Min Fairness⏎

Algorithm		*Statistical Analysis*	*Start*	*End*	*Actual Bandwidth*	*Data to Transfer*

Algorithm		*Statistical Analysis*	*Start*	*End*	*Actual Bandwidth*	*Data to Transfer*
EP-MCPHS		**mean**	1.0	2.615385	6.428571	4.916667
		std	0.0	1.626700	3.857320	7.548959
		25%	1.0	1.000000	2.000000	0.000000
		50%	1.0	3.000000	6.250000	0.000000
		75%	1.0	4.000000	6.250000	10.000000
Max-Min Algorithm	**Fairness**	**mean**	1.225806	3.600000	3.811828	6.532258
		std	0.425024	2.898275	1.511640	8.107522
		25%	1.0	1.000000	2.041667	0.000000
		50%	1.0	3.000000	4.166667	1.875000
		75%	1.0	4.000000	4.687500	12.875000

Table 11.5 makes a comparison of EP-MCPHS and MMFA, showcasing that the proposed hybridized placement algorithm presents lower start time statistics across mean, standard deviation, 25%, 50%, and 75% scores. Similarly, the end time is reduced to 2.615 in comparison to 3.6 seconds earlier. Further, the placement algorithm has a standard deviation of 0, meaning that most algorithms are placed in the first unit of time during simulation. The utility of EPMCPHS algorithm is the increase in bandwidth from 3.811828 in MMFA to 6.428571. The algorithm also reduces the inherent data transfer rate as the most optimal task in case of resources is running on the top edge server. Henceforth, smaller tasks need to be migrated. This is showcased by the parameter 4.91 mean in comparison to MMFA 6.53, as well as the standard deviation of 7.54 during EP-MCPHS 8.10.

Figure 11.3 makes a comparison between EP-MCHS and MMFA, showing the start, end, actual bandwidth, and data to transfer for the two algorithms. The figure also shows improvement in the mean of start, end, and data transfer wherein increase of data to transfer from 6.53 to 4.91 on EP-MCHS.

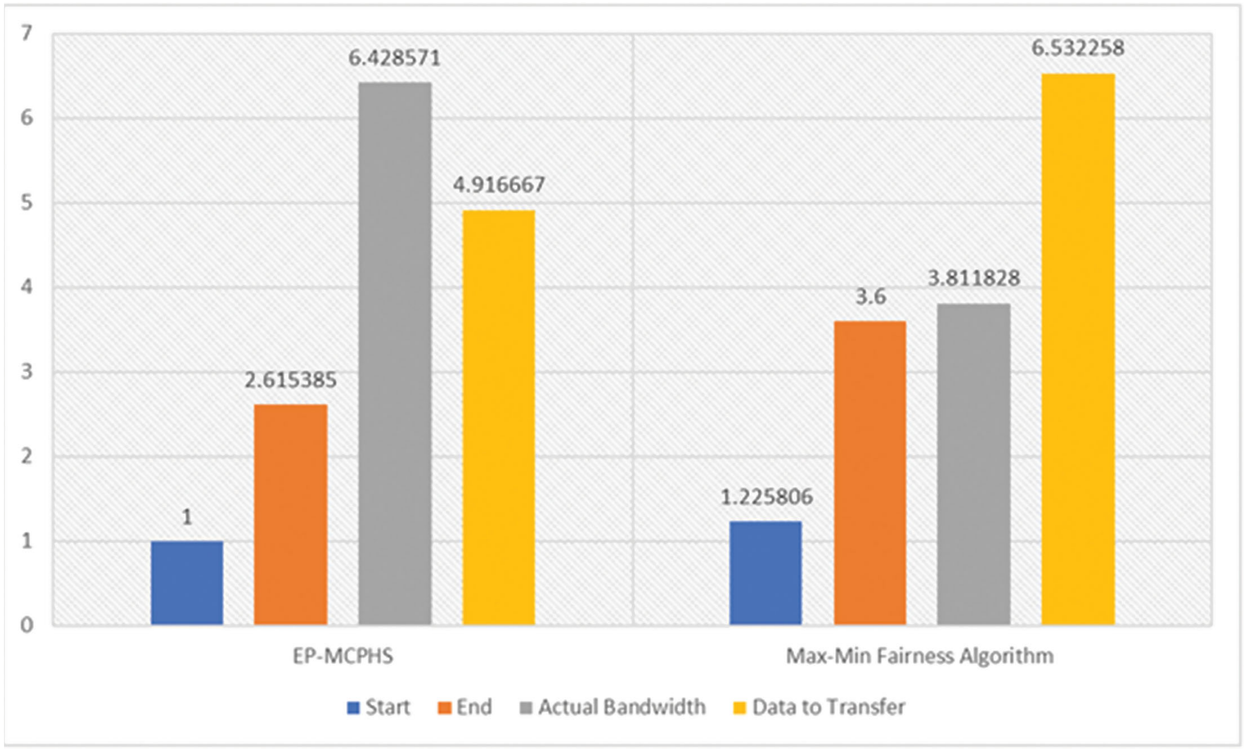
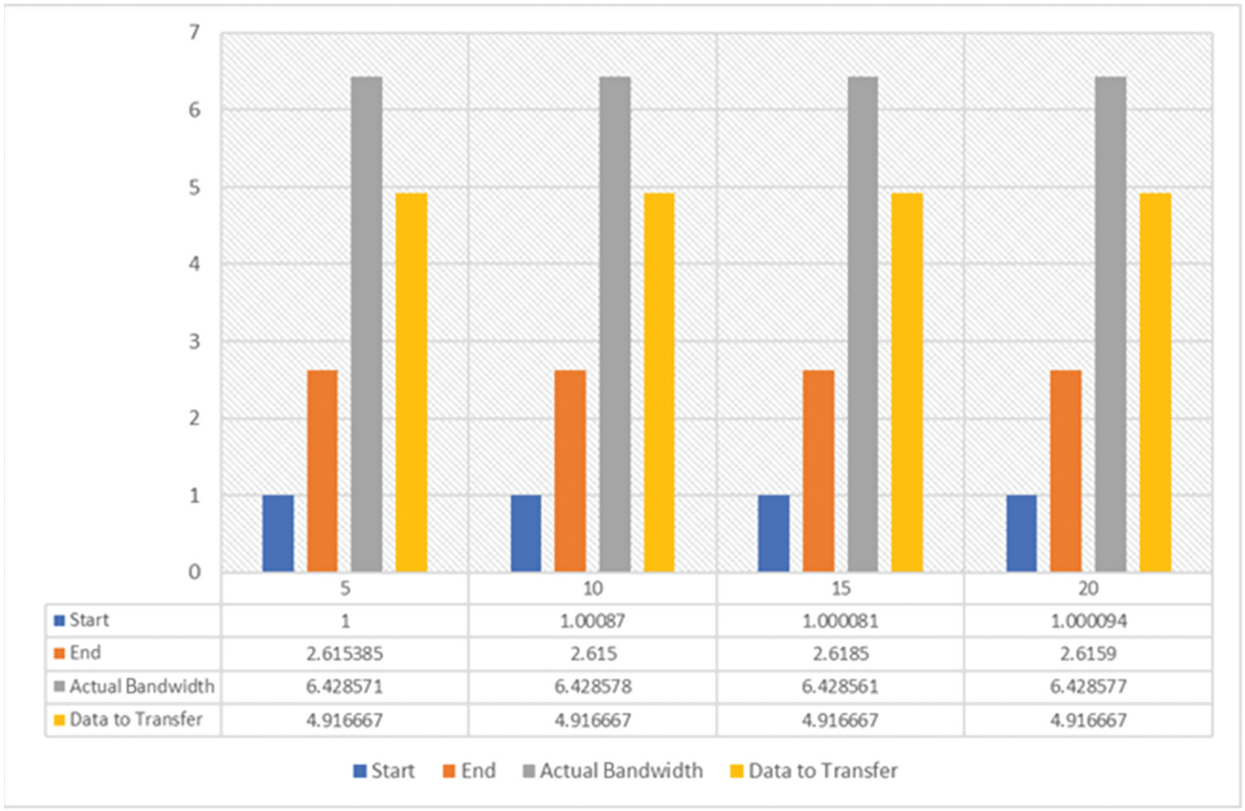


Figure 11.3 Comparison between EP-MCHS and max-min fairness algorithm.⏎

Table 11.6 Simulation Analysis across Time Interval⏎

Interval	*Statistical Analysis*	*Start*	*End*	*Actual Bandwidth*	*Data to Transfer*
	Mean	1.0	2.615385	6.428571	4.916667
5	**Standard Deviation**	0.0	1.626700	3.857320	7.548959
	Mean	1.00087	2.615	6.428578	4.916667
10	**Standard Deviation**	0.0715	1.62686	3.857320	7.548959
15	**Mean**	1.000081	2.6185	6.428561	4.916667
	Standard Deviation	0.00006	1.62579	3.857320	7.548959
20	**Mean**	1.000094	2.6159	6.428577	4.916667
	Standard Deviation	0.00051	1.626709	3.857320	7.548959

Table 11.6 presents simulation analysis across time intervals and Figure 11.4 showcases the meager change in the start, end, bandwidth, and data to transfer values within the framework of simulation across time intervals.



	5	10	15	20
🟦 Start	1	1.00087	1.000081	1.000094
🟧 End	2.615385	2.615	2.6185	2.6159
⬜ Actual Bandwidth	6.428571	6.428578	6.428561	6.428577
🟨 Data to Transfer	4.916667	4.916667	4.916667	4.916667

Figure 11.4 Simulation analysis across interval time.⏎

11.5 Conclusion

The concept of edge computing is gaining popularity as a paradigm that may provide applications with minimal latency by moving processing from typical cloud data centers to the periphery of the network. To ensure that edge computing goes from being a promise to a reality, it is necessary to create effective approaches for resource management. This is because expectations about the potential advantages of edge computing are growing significantly. The need for today's day and age mobile applications to be onboarded with applications needing, high computation, low latency, high bandwidth, etc. This leads to a scenario wherein the fast age computation is needed, and a general cloud server with high latency cannot cater to the needs of such high demands. Thus, to mitigate this edge computing resources are selected. In this study, the proposition of EP-MCPHS allows the use of minimum heap with multi-processing techniques to reduce the workload from the unutilized edge servers, enhance resource utilization, reduce latency, and increase bandwidth to cater the service request. This follows when the effective flow rate of the network drops as per our experiments in Section 11.4.3. This is because the current effective resources are allocated to the most optimal process leading to low data transfer in the network, thus further leading to a low latency and high bandwidth model. The final comparison with the first come first serve model architecture and the reduction of time interval-based cross-validation showcases the superiority of the modeling architecture of EP-MCPHS over other current state-of-the-art techniques for placement of cloudlet in edge servers. The algorithm also enhances the need for fast cloudlet offloading for high-demand mobile applications, thus catering to the D2D offloading with cloudlet-to-edge server mapping as well. It increases the bandwidth from 3.811828 to 6.428571 in the MMFA. The algorithm also reduces the inherent data transfer rate to 4.91 for EPMCPHS in comparison to 6.53 for MMFA.

11.5.1 Future Scope

This study aims to create a dynamic algorithm capable of managing the load, reducing latency, increasing bandwidth, and decreasing network flow for cloudlet offloading into edge server simulation. The chapter uses EdgeSimPy simulation to achieve the nuances described.

The future scope of this research can be showcased by the enhancement of DRL mechanisms wherein a neural simulation for the architecture can be drawn and tested via neural agents. The simulation could allow to adopt to reduce latency and resource utilization. This would also allow for a nonparametric algorithm that could replace the current parameterized setup and create global loss functions for the calculation of algorithm fitness like the studies for feature selection.

References

1. Roman, R., Lopez, J., Mambo, M. (2018). Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. *Future Generation Computer Systems,* 78, 680–698.
2. Shahzadi, S., Iqbal, M., Dagiuklas, T., Qayyum, Z. U. (2017). Multi-access edge computing: Open issues, challenges and future perspectives. *Journal of Cloud Computing,* 6, 1–13.
3. Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective*. IEEE Communications Surveys & Tutorials,* 19, 2322–2358.
4. Satyanarayanan, M., Klas, G., Silva, M., Mangiante, S. (2019). The seminal role of edge-native applications*.* In *2019 IEEE International Conference on Edge Computing*, IEEE, pp. 33–40.
5. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N. (2009). The case for VM-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8, 14–23.
6. Zhao, H., Deng, S., Liu, Z., Yin, J., Dustdar, S. (2020). Distributed redundancy scheduling for microservice-based applications at the edge. *IEEE Transactions on Services Computing*, 15(3), 1732–1745.
7. Wang, J., Feng, Z., George, S., Iyengar, R., Pillai, P., Satyanarayanan, M. (2019). 1130 Towards scalable edgenative applications. In *ACM/IEEE Symposium on Edge Computing*, pp. 152–165.
8. Souza, P., Crestani, A., Ferreto, T., Rossi, F. (2022). Latency-aware privacypreserving service migration in federated edges. In *International Conference on Cloud Computing and Services Science*, pp. 288–295.
9. Souza, P., Ferreto, T., Rossi, F., Calheiros, R. (2022). Location-aware maintenance strategies for edge computing infrastructures. *IEEE Communications Letters*, 26, 848–852.
10. Yu, Z., Xu, X., Zhou, W. (2022). Task offloading and resource allocation strategy based on deep learning for mobile edge computing. *Computational Intelligence and Neuroscience*, 2022, 1–11. https://doi.org/10.1155/2022/1427219(Not accessible as of [2025/07/18])⏎

11. Vaisman, R., Kroese, D. P., Gertsbakh, I. B. (2016). Improved sampling plans for combinatorial invariants of coherent systems. *IEEE Transactions on Reliability*, 65(1), 410–424. https://doi.org/10.1109/tr.2015.2446471⏎
12. Li, X. (2021). A computing offloading resource allocation scheme using deep reinforcement learning in mobile edge computing systems. *Journal of Grid Computing*, 19(3). https://doi.org/10.1007/s10723-021-09568-w⏎
13. Ullah, I., Lim, H.-K., Seok, Y.-J., Han, Y.-H. (2023). Optimizing task offloading and resource allocation in edge-cloud networks: A DRL approach. *Journal of Cloud Computing*, 12(1). https://doi.org/10.1186/s13677-023-00461-3⏎
14. Mozib, M. (2023). Distributed deep learning based framework to optimize real-time offloading in mobile edge computing networks. *International Journal of Science and Research*, 12(6), 1812–1827. https://doi.org/10.21275/sr23603125305⏎
15. Yang, H., Xu, C., Liu, S., Zhang, J. (2020). A privacy-preserving task offloading scheme for edge computing in the Internet of Things. *IEEE Internet of Things Journal*, 7(7), 6124–6136.
16. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M. (2015). Internet of Things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4), 2347–2376.
17. Alghamdi, I. (2021). *Computation offloading in mobile edge computing: An optimal stopping theory approach*. PhD thesis, University of Glasgow.⏎
18. Cardellini, V., De Nitto Person´e, V., Di Valerio, V., Facchinei, F., Grassi, V., Lo Presti, F., Piccialli, V. (2016). A game-theoretic approach to computation offloading in mobile cloud computing. *Mathematical Programming*, 157, 421–449.⏎
19. Patel, M., Sabella, D., Sprecher, N., Young, V. (2015). Mobile edge computing—a key technology towards 5g. *ETSI White Paper*, 11(11), 1–16.⏎
20. Alfakih, T., Hassan, M. M., Gumaei, A., Savaglio, C., Fortino, G. (2020). Task offloading and resource al location for mobile edge computing by deep reinforcement learning based on sarsa. *IEEE Access*, 8, 54074–54084.
21. Mao, Y., Zhang, J., Song, S. H., Letaief, K. B. (2017). Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems. *IEEE Transactions on Wireless Communications*, 16(9), 5994–6009.
22. Trinta, F. A., Hasan, M. Z., de Souza, J. N., et al. (2019). Enhancing offloading systems with smart decisions, adaptive monitoring, and mobility support. In *Wireless Communications and Mobile Computing 2019*.
23. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4), 14–23.
24. Wang, S., Zhang, X., Zhang, Y., Wang, L., Yang, J., Wang, W. (2017). A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Access*, 5, 6757–6779.
25. Zhang, K., Mao, Y., Leng, S., Zhao, Q., Li, L., Peng, X., Pan, L., Maharjan, S., Zhang, Y. (2016). Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. *IEEE Access*, 4, 5896–5907.
26. Souza, P. S., Ferreto, T., Calheiros, R. N. (2023). EdgeSimPy: Python-based modeling and simulation of edge computing resource management policies. *Future Generation Computer Systems [online]*, 148, 446–459. https://doi.org/10.1016/j.future.2023.06.013

Index