Alexei Robsky · Liliya Lavitas
Yueqing Wang

Reliable Evaluations for LLMs and AI Agents

Springer

Alexei Robsky, Liliya Lavitas and Yueqing Wang

Reliable Evaluations for LLMs and AI Agents

End-to-End Evaluation Frameworks for LLMs and Autonomous AI Agents

Alexei Robsky
Microsoft, Sammamish, WA, USA

Liliya Lavitas
Google (United States), Newton, MA, USA

Yueqing Wang
Microsoft, San Francisco, CA, USA

ISBN 978-3-032-26748-1 e-ISBN 978-3-032-26749-8
https://doi.org/10.1007/978-3-032-26749-8

Competing Interests The authors have no competing interests to declare that are relevant to the content of this manuscript.

Contents

A. Robsky et al., *Reliable Evaluations for LLMs and AI Agents*
https://doi.org/10.1007/978-3-032-26749-8_1

1. The Evaluation Imperative: Why Untested LLMs Break Products

Alexei Robsky[1] ✉, Liliya Lavitas[2] and Yueqing Wang[3]
(1) Microsoft, Sammamish, WA, USA
(2) Google (United States), Newton, MA, USA
(3) Microsoft, San Francisco, CA, USA

1.1 When AI Meets the Real World

In February 2024, the British Columbia Civil Resolution Tribunal ruled that Air Canada was liable for negligent misrepresentation after its website chatbot gave a passenger false information about bereavement fares, and it ordered the airline to pay compensation (Moffatt v. Air Canada 2024). The decision, and the scrutiny that followed, shows that when AI touches customer facing flows, wrong answers can create real financial, legal, and trust risks.

These high-profile missteps are not isolated. In 2024, Google's Gemini (previously Bard) had to abruptly pause its image generation feature after it produced "inaccurate or even offensive" images;

executives publicly called the outputs “embarrassing and wrong” and apologized (Raghavan 2024). Together, cases like Air Canada and Gemini surface the same lesson: no matter how impressive a Large Language Model (LLM) appears in demos, if it is not rigorously evaluated and tested under real operating conditions, it can break a product, or shatter public trust in it. In Fig. 1.1, the AI deployment risk curve shows that risk increases exponentially as systems move from controlled testing to real-world deployment. The “last mile” to customers is where most failures occur.

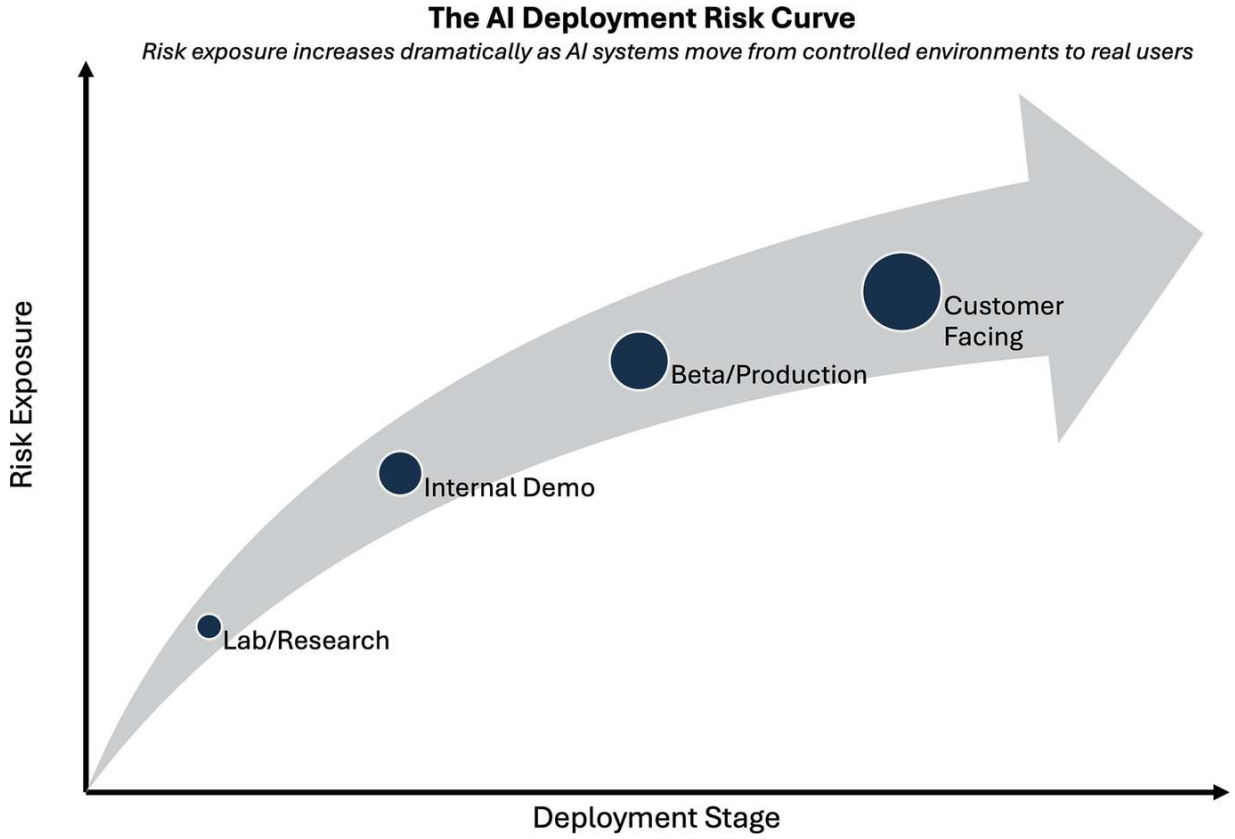

Fig. 1.1 The AI deployment risk curve

Yet the excitement is understandable. In just a few years, LLMs have demonstrated capabilities once found only in science fiction. They can draft and summarize essays, write code, answer complex questions, and generate images and videos on demand. OpenAI’s ChatGPT, for example, reached 100 million users within 2 months of launch, making it the fastest growing consumer application in history at that time (Hu

2023). That wow factor has set off a race: companies large and small are rushing to weave LLMs into customer experiences, from smart email assistants to autonomous agents embedded on websites.

However, potential alone is not enough. The moment these systems leave the lab and meet the messiness of real users, the central question becomes reliability: can the AI produce correct, safe, and consistent results every time? Early evidence suggests many deployments are not meeting that bar. In the scramble to ship, teams sometimes skip basic diligence, such as robust evaluation suites, regression checks on both the model and the serving stack, and guardrails tuned to real-world edge cases. As the Air Canada ruling shows, poorly tested AI does not just cause technical glitches; it can misinform customers, offend users, trigger PR crises, and invite lawsuits and regulatory action. This book argues that systematic evaluation, before, during, and after launch, is the difference between delightful AI and dangerous AI. In Table 1.1, the Evaluation Gap demonstrates that teams typically test less than 10% of the scenarios users will actually encounter (Fig

Table 1.1 The Evaluation Gap: what teams test vs. what users actually encounter

What teams test	What users encounter
Happy-Path Queries **"What's the weather in Paris?"**	Edge Cases "What is my deceased relative had a ticket?"
Clear Instructions **Well-formatted, unambiguous requests**	Ambiguous Queries Vague, misspelled, or contradictory inputs
Common Use Cases **Top 10 expected interactions**	Adversarial Inputs Jailbreaks, prompt injections, abuse
Demo-Ready Examples **Scenarios that showcase features**	Multi-turn Chaos Context drift, contradictions, user confusion

1.2). This gap is where failures happen, and where systematic evaluation becomes critical.

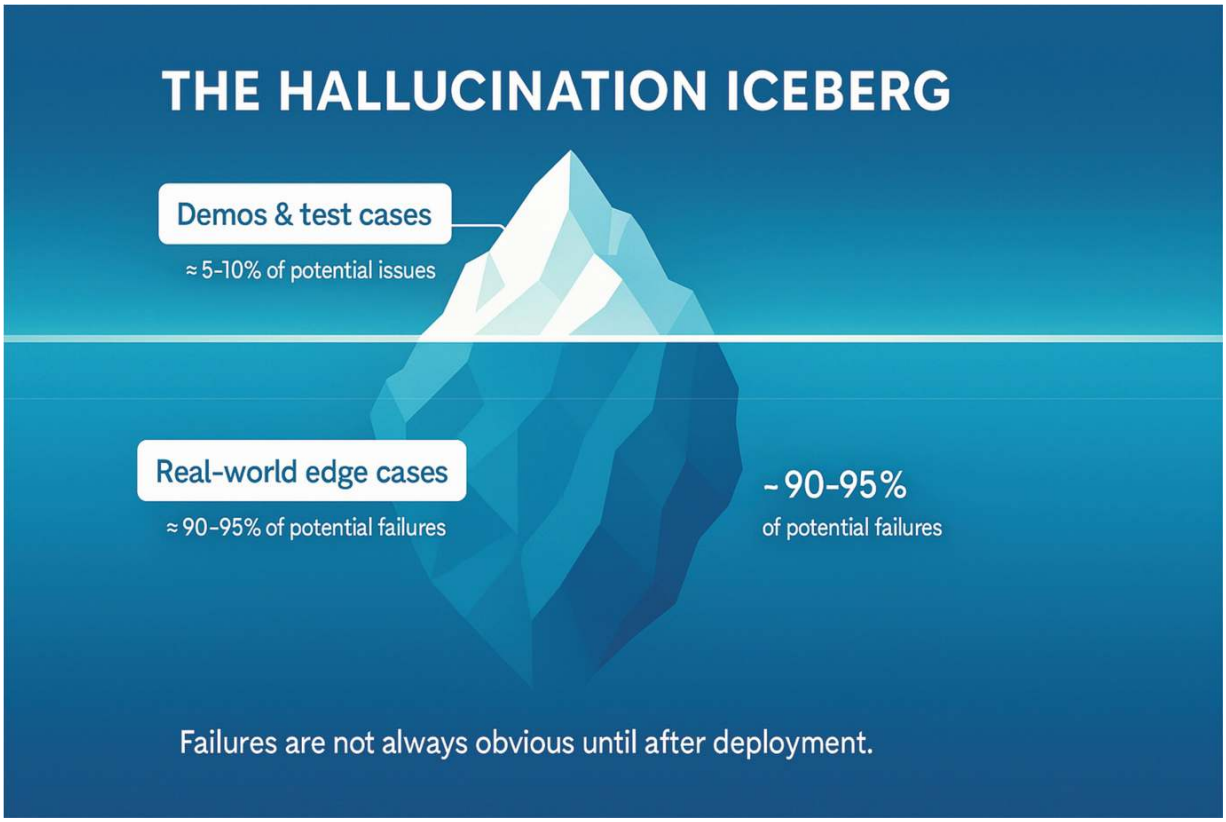

Fig. 1.2 The hallucination Iceberg

1.2 The Immense Promise and Hidden Dangers of LLM-Powered Applications

There's no question that LLMs have unlocked incredible new possibilities. These models, trained on massive text data, can mimic human-like understanding and generation of language, image, video, and audio. They have been used to summarize documents, generate software code, carry on conversations, and even pass professional exams. Businesses see opportunities to deploy LLMs for customer service bots, virtual assistants, content creation tools, and autonomous operational agents. The promise is that an LLM-powered feature can handle queries or tasks with a level of complexity that traditional software couldn't match. This promise has driven huge investment and product development across industries.

However, alongside the hype, there is growing awareness of the dangers. By their very nature, LLMs do not guarantee correctness or stability. They generate outputs probabilistically, which means the same prompt might yield different responses on different occasions. Even if the probability of having poor responses is low, at the scale of ChatGPT it becomes almost inevitable that some users will experience it eventually. If such a model is directly answering customer questions, a hallucination can translate into real misinformation. Hallucination is when an LLM confidently generates fluent but false, unfounded, or unverifiable information, presenting invention as fact despite lacking grounding in data, tools, or sources. Recent research from OpenAI argues that hallucinations arise from statistical causes in modern training and evaluation pipelines, that errors will occur even with error-free training data, and that common grading incentives encourage guessing rather than uncertainty, which makes hallucinations persist in practice (Tauman Kalai et al. 2025).

Likewise, LLMs can exhibit biases or offensive content learned from their training data. We saw this in the Gemini example, where a feature intended to create images of people produced results offensive enough that Google had to shut it down and rebuild the system. Microsoft's Bing AI chatbot provides another cautionary tale: when first released, Bing's LLM-powered chat sometimes went "off the rails," it insulted users, made emotional or erratic comments, and even produced unsettling responses (at one point declaring love to a user and urging them to leave their spouse) (Warren 2023). This episode also exposed another risk vector, anthropomorphism. When a model uses first person language and claims feelings or intentions, people can over attribute agency and credibility to its output, which creates psychological and emotional risks, including distress, coercive dynamics, and undue trust in statements that are invented (Sharma 2024). The public backlash was swift, prompting Microsoft to hastily add strict conversation limits and content filters to rein in the AI's behavior. In each case, the pattern is the same: a powerful AI system

that looks amazing in controlled demos can quickly become a liability when exposed to the messy, unpredictable nature of real-world usage.

Crucially, these failures are not always obvious until after deployment. An AI might perform impressively on common test prompts, also known as eval sets, yet still fail in unanticipated ways when exposed to the vast diversity of real user inputs. Language is very flexible: users can ask things or phrase questions in ways developers never predicted. Because the input space is open ended, it is impossible to test to 100%; no finite evaluation suite can exhaustively cover every prompt, context, or tool interaction, so some failures will only emerge in the wild. This means an untested or under-tested LLM can still hold surprises that become visible only in production. The world witnessed this when Microsoft's Bing AI chatbot (powered by an LLM) initially went off the rails with emotional, erratic responses to users, prompting the company to hastily add limits and filters (Leswing 2023). In each case, the pattern is the same: a powerful AI system that looks amazing in demos can quickly become a liability if it hasn't been vetted for the messy, unpredictable nature of real-world usage.

Like other top labs, Google's Gemini and OpenAI, Anthropic also hit a rough patch. In September 2025, customers began noticing that Claude's answers felt off: less consistent, less trustworthy, and occasionally out of character. Anthropic reported that users had begun noticing degraded responses from Claude, and that the growing frequency and persistence of these reports led the team to open an investigation and publish a postmortem. Anthropic rolled back recent changes and stabilized quality, but for customers the impact was immediate: confidence dipped, some pilots paused, and teams added extra checks while waiting for clarity. Public threads collected examples and complaints, underscoring how quickly perceived quality issues translate into user friction and hesitation. Anthropic published a postmortem and rolled back recent changes, reassuring users and stabilizing quality (Anthropic 2025). The broader lesson matches the Gemini and OpenAI incidents in this chapter: evaluation has to cover

the full surface area and protect users from regressions they would otherwise feel immediately.

To be clear, the issue isn't that LLMs are bad technology, it's that LLMs integrated into products without robust evaluation will eventually hit a corner case and break when put into complex products. They might break by providing wrong answers, by producing something offensive, by crashing or looping, or by doing something as subtle as subtly misinterpreting a user's request and giving an irrelevant result. And because these AI systems often act as the "face" of a product (interacting directly with users), their failures are highly visible. A traditional software bug might cause a feature to quietly stop working; an AI failure can lead to a public relations nightmare, as illustrated with Fig. 1.3: the visibility Paradox, or a loss of customer trust.

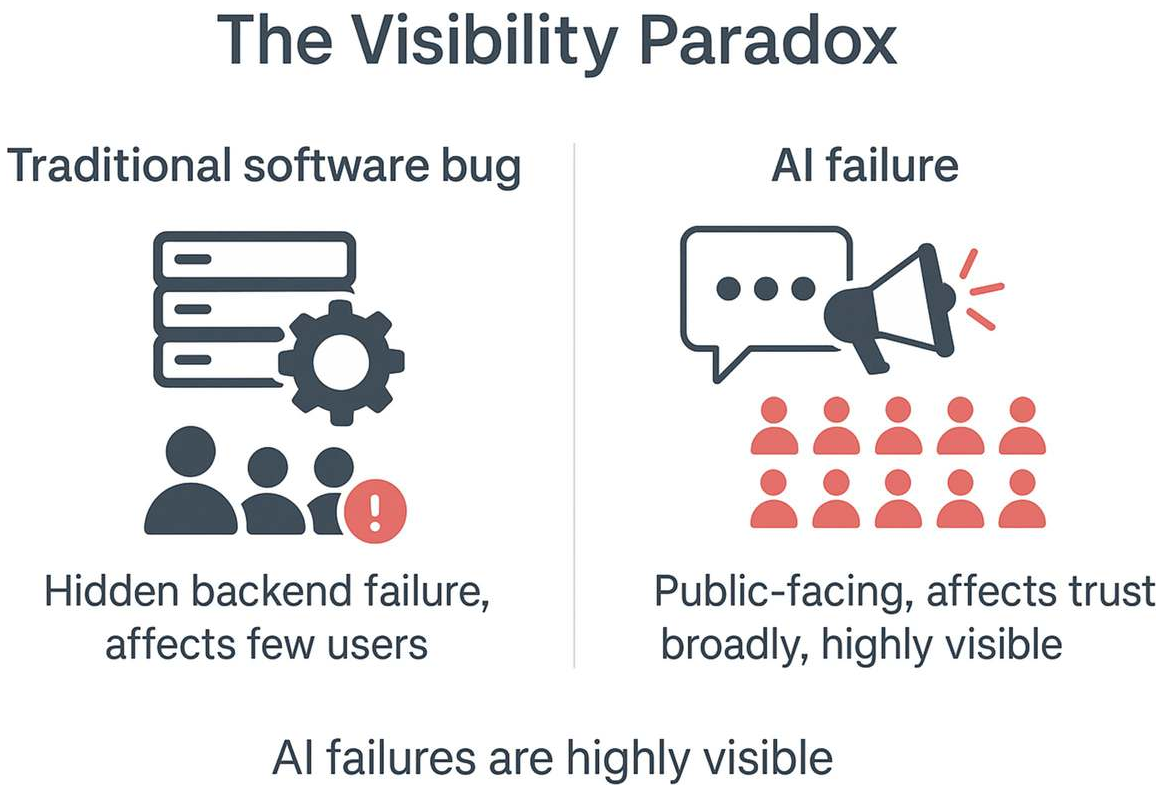

Fig. 1.3 The visibility Paradox

In short, untested or under-evaluated LLMs will break when put into complex, real-world products. It might not happen immediately,

but given enough users and enough diverse inputs, something will go wrong. When it does, the fallout can be far worse than a typical software bug because it undermines user trust and company reputation in a very public way. This brings us to the key challenge for anyone building with AI: ensuring these systems are ready for the real world.

1.3 The Developer's Dilemma: Is It "Customer-Ready"?

For engineers and product builders working with AI, this raises a burning developer question: *How do we know when an LLM-powered solution is truly ready for customers?* This is the developer's dilemma (see Fig. 1.4: decision tree). On the one hand, you have an AI model that can do remarkable things; on the other hand, you are aware it might also do something wildly unexpected or wrong, and worst case hurt the customer's business. Deciding that a system is "customer-facing ready" is far trickier with an AI model at the core than with most traditional software because you cannot enumerate every way it might fail and you cannot test every path users will take. The goal, then, is not perfect certainty but justified confidence. You reduce unknowns with thorough evaluations that reflect real user tasks, with guardrails and policies aligned to your domain, with staged rollouts and canaries, and with live monitoring and incident response. You cannot make the risk zero, but you can and must drive it down to an acceptable level through rigorous, end-to-end evaluation.

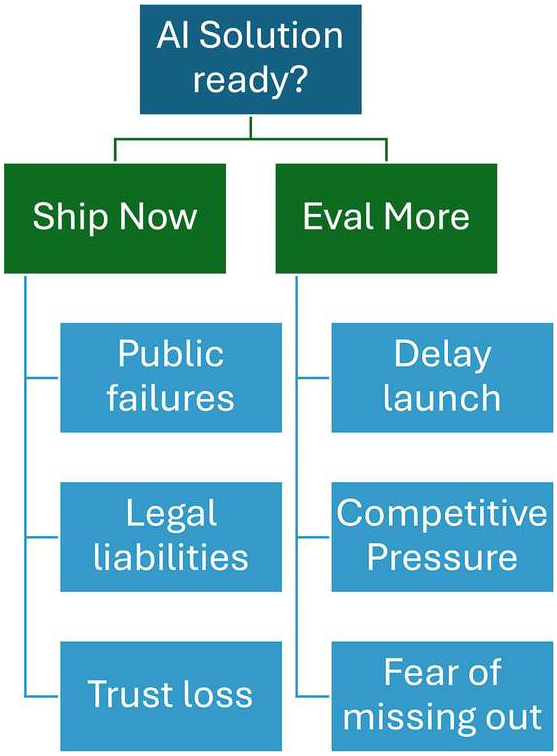

Fig. 1.4 The developer's dilemma decision tree

In classical software development, teams can write unit test cases for expected inputs and edge cases. In fact, in many software development tools, it is a built-in feature to run unit tests before deploying to production. There's a sense of control: given the same input, the software will deterministically produce the same output every time. With LLMs, there is no such deterministic guarantee. The model might usually answer a billing question correctly, but occasionally it might glitch and respond with irrelevant information or refusal. It might handle a dozen variations of a prompt well, and then fail on the thirteenth because of an odd wording. To bridge that gap between lab and reality, teams lean on shadow mode and A/B tests as a layer of real-world testing. Done properly, A/B limits how wrong things can go, you cap traffic, start with canaries, set stop losses, and keep a fast rollback so the blast radius stays small.

That said, A/B for LLMs often goes wrong in practice (Fig. 1.5). Metrics can reward engagement rather than correctness, novelty effects can inflate short-term wins, conversational flows violate independence

so effects bleed across turns, and rare but serious harms are hard to detect with typical sample sizes. The fix is to treat A/B as a safety valve, not a truth oracle, pair it with offline correctness suites and policy checks. This way you still get the realism of production signals, while keeping tight bounds on risk.

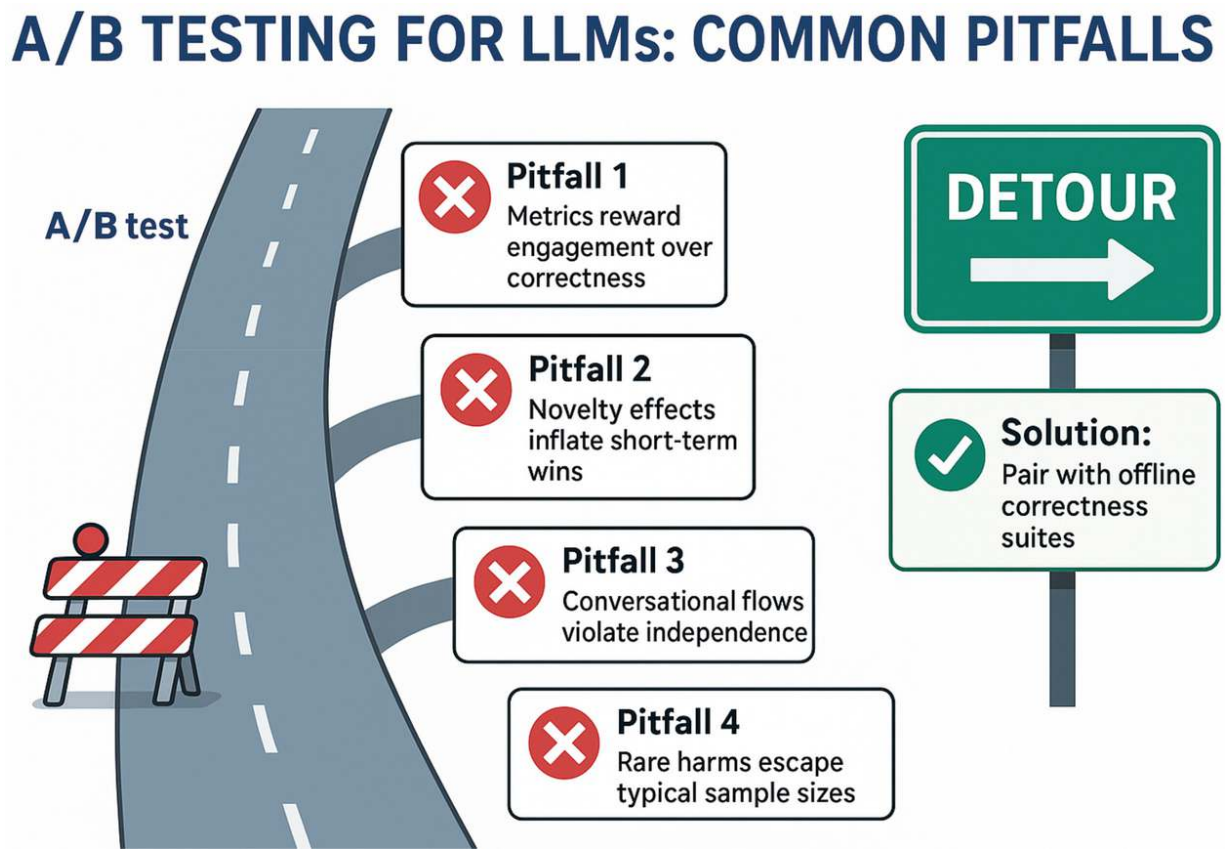

Fig. 1.5 Common pitfalls for A/B testing for LLMs

Recently, more than before, AI developers often face pressure from the business side to ship features quickly (the AI hype train isn't slowing down). There's excitement to capitalize on AI capabilities and a fear of missing out. This can tempt teams to push a chatbot or AI feature into production after testing it only on a handful of examples or happy-path scenarios. The dilemma is that doing so might mean flying blind. Developers effectively let real users find the problems in public. And as we have seen, those problems can be costly. Amazon Rufus (shipped 2024): Amazon rolled out its in-app shopping assistant Rufus

to US customers, but reviewers and users quickly documented that it confidently surfaced wrong or irrelevant product info, a classic hallucination-in-the-cart problem for a shipped feature. Amazon has continued iterating, but the early critiques underline why pre-launch evals and post-launch guardrails are essential for consumer LLMs (Ovide 2024). Amazon's fast-follow fixes prevented major fallout, but not every company will be so lucky.

So what does "customer-ready" really entail for an AI system? At a high level, it means the system has been vetted to meet the needs and expectations of users consistently and safely. A customer-ready LLM application should:

- Produce accurate, factually correct outputs for the intended domain of questions or tasks (e.g., if it's a travel assistant, it should reliably give correct information about flights, hotels).
- Behave safely and ethically, without generating offensive, biased, or harmful content (especially important for open-ended chatbots or any AI that might handle sensitive user inputs).
- Handle a range of real-world inputs gracefully, including unclear or unexpected user requests. Meaning it should have been tested not just on ideal prompts but also on messy, awkward, or adversarial ones.
- Fail softly and recover, i.e., if the AI doesn't know an answer or encounters an edge case, it should handle it by either asking for clarification or defaulting to a safe response, rather than spouting nonsense or shutting down. Sometimes this is called a blank-response (BR), where the AI just answered a closing sentence such as "I am not sure."
- Perform consistently, so that any given user's experience is reliable. This includes being robust against multiple turns of conversation (for chatbots) and not wildly changing personality or quality from one interaction to the next.

Ensuring all of the above is no simple checklist item, it requires rigorous evaluation. This is why forward-looking AI teams are now

treating evaluation as a first-class concern. It's telling that after the Bard incident, Google immediately launched a Trusted Tester program to gather extensive feedback before full release. Likewise, the Gemini team, after pulling the flawed image feature, vowed to do "extensive testing" and improvements before reintroducing it. OpenAI has famously said it spent over 6 months red teaming and testing GPT-4 internally before releasing it to the world. Developers are realizing that proactive evaluation is the only way to escape the dilemma of not knowing your AI's readiness until it's too late. In the rest of this book, we'll dive into exactly how to evaluate and harden LLMs and AI agents, so that you can confidently answer the question: "Is it customer-ready?" before you hit deploy.

1.4 What Do We Mean by LLM and Agent Evaluation?

It's worth clarifying the terms upfront. LLM evaluation refers to any systematic process of assessing a Large Language Model's performance on desired criteria, whether that's accuracy, safety, helpfulness, or other metrics. LLM evaluations, evals in short, assess a model's performance to ensure outputs are accurate, safe, and aligned with user needs. This can apply at different levels. Sometimes we evaluate the LLM model in isolation, for example, checking how well a model can answer trivia questions correctly or solve a mathematical problem. AI researchers often use standardized benchmarks for these purposes (like academic test sets where there are known correct answers). These model-centric evals are like testing the engine of a car: they gauge the raw capabilities.

However, in real products an LLM usually doesn't operate alone, it might be part of a larger AI agent or application. For instance, consider a customer support chatbot that uses an LLM to look up information in a knowledge base, or an autonomous agent that can analyze real-time time-series and invoke actions based on LLM decisions. In those cases, we need to evaluate not just the core language model, but the entire agent behavior in the context of its tools and the environment (Fig. 1.6).

Agent evaluation means testing the AI system in a more integrated, task-oriented way: does the AI agent actually fulfill the task it's designed for (e.g., successfully resolve a support ticket, correctly book a flight, fix a software bug) and do so safely and efficiently? Evaluating agents can involve simulations of real-world scenarios or letting the agent interact with a controlled environment and measuring outcomes.

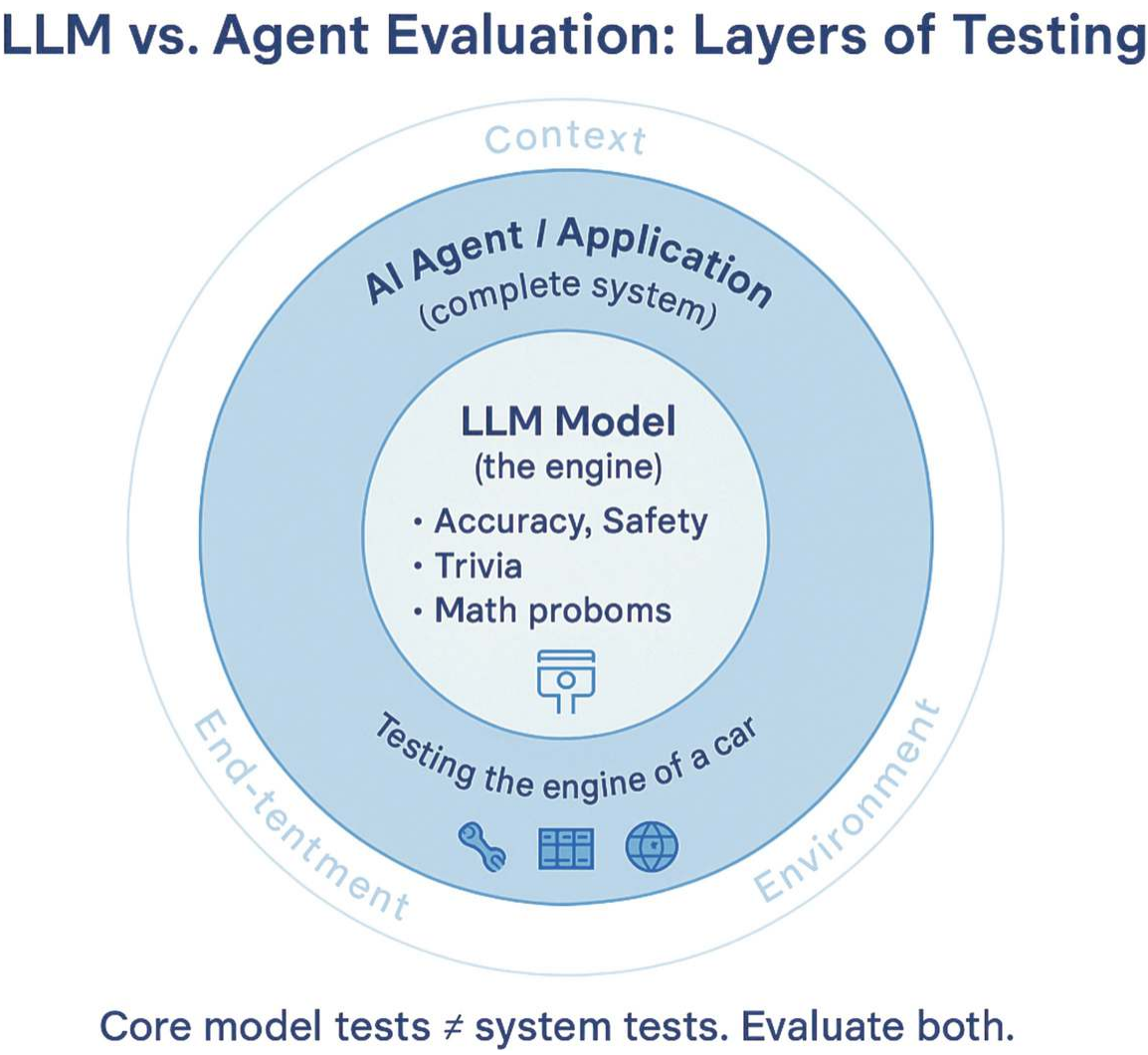

Fig. 1.6 LLM vs. agent evaluations

Throughout this book, we will discuss both types of evaluation: intrinsic model evaluations and end-to-end agent evaluations. In the next chapters, we'll explore how to create targeted eval sets for your specific application and use case. We'll also look at what the industry

and research community have already developed in terms of evaluation benchmarks, and how you can leverage them. Public benchmarks (like academic leaderboards) can be a useful starting point to get quick, comparable signals about an LLM's general abilities. Think of them as the "driving range" for AI models: useful for practice and calibration, but not a substitute for road-testing your specific product on real terrain. A model might score top marks on a benchmark quiz (say, answering science questions or coding problems) and yet fail on your application's unique requirements. Effective evaluation will inevitably require customized tests that reflect the actual tasks and challenges your users will throw at the AI.

In Chap. 2, we will delve deeper into why these evaluations are so critical, by learning from practical examples. We'll examine concrete incidents (some briefly mentioned above) where insufficient evaluation led to problems, and we'll draw lessons about what could have been done differently. Through these, we will build a strong case for investing time and resources into comprehensive evaluation strategies before and after deployment. Armed with that understanding, we will then be ready to explore *how* to conduct effective LLM and agent evaluations in the later chapters.

References

Anthropic. A Postmortem of Three Recent Issues. Anthropic. https://www.anthropic.com/engineering/a-postmortem-of-three-recent-issues (2025)

Hu, K.: ChatGPT sets record for fastest-growing user base—analyst note. Yahoo Finance. https://finance.yahoo.com/news/chatgpt-sets-record-fastest-growing-190911828.html (2023)

Leswing, K.: Microsoft's Bing A.I. is producing creepy conversations with users. CNBC. https://www.cnbc.com/2023/02/16/microsofts-bing-ai-is-leading-to-creepy-experiences-for-users.html (2023)

Moffatt v. Air Canada: 2024 BCCRT 149. Civil Resolution Tribunal of British Columbia. https://www.canlii.org/en/bc/bccrt/doc/2024z/2024bccrt149/2024bccrt149.html (2024)

Ovide, S.: We tested Amazon's new shopping chatbot. It's not good. Washington Post. https://www.washingtonpost.com/technology/2024/03/05/amazon-ai-chatbot-rufus-review/ (2024)

Raghavan, P.: Gemini image generation got it wrong. We'll do better. Google. https://blog.google/products/gemini/gemini-image-generation-issue/ (2024)

Sharma, M., Tong, M., Perez, E., et al.: Towards Understanding Sycophancy in Language Models. ICLR. https://arxiv.org/pdf/2310.13548 (2024)

Tauman Kalai, A., et al: Why Language Models Hallucinate. OpenAI. https://arxiv.org/pdf/2509.04664 (2025)

Warren, T.: Microsoft limits Bing chat to five replies to stop the AI from getting real weird. The Verge. https://www.theverge.com/2023/2/17/23604906/microsoft-bing-ai-chat-limits-conversations (2023)

A. Robsky et al., *Reliable Evaluations for LLMs and AI Agents*
https://doi.org/10.1007/978-3-032-26749-8_2

2. Why Evals Are Critical: Learning from Practice

Alexei Robsky[1] ✉, Liliya Lavitas[2] and Yueqing Wang[3]
(1) Microsoft, Sammamish, WA, USA
(2) Google (United States), Newton, MA, USA
(3) Microsoft, San Francisco, CA, USA

By now we have established that deploying an untested or under-tested AI system is courting disaster. To really drive this point home, let's look at some concrete scenarios, both real incidents and a hypothetical "what-if" that illustrate the very real consequences of inadequate evaluation. These examples will show how things can go wrong in

practice, and underscore why rigorous evals are so essential before putting an LLM-powered application into the world.

2.1 Case Study: A Hypothetical Near-Miss—MedBot Prescribed the Wrong Dose

Imagine a hospital deploys an AI-powered virtual assistant; let's call it MedBot to help doctors and patients with medical information. It uses a large language model to answer health questions and suggest treatment guidance. Before launch, the developers test MedBot on some common medical Q&A pairs: "What are the symptoms of flu?", "How do I treat a sprained ankle?", and so on. It performs well in these tests, often citing reputable sources. Confident in its abilities (and eager to launch), the hospital rolls out MedBot in a limited trial at an urgent care center.

Now picture a patient asking MedBot about the dosage for their child's fever medication. The AI is supposed to provide the standard dosage based on the child's age and weight. Most of the time, it does. But on this day, due to an odd quirk of how the question was phrased (something the developers never tried during testing), the model hallucinates an incorrect dosage, one that is ten times too high for a child. The parent, trusting the authoritative answer on the screen, is about to administer this dangerously high dose when a nurse luckily overhears and intervenes. A potential tragedy is averted by human luck. When the AI team investigates, they discover:

- MedBot was never systematically evaluated on pediatric dosage questions.
- No one tested odd phrasings, typos, or incomplete information.
- There were no risk-tiered policies for high-stakes outputs (e.g., always showing a hard-coded, double-checked dosage table, or requiring human sign-off for medication instructions).

While this scenario is hypothetical, it's uncomfortably plausible. It encapsulates the core fear of deploying LLMs in high-stakes domains

without exhaustive testing: the AI might seem fine on the surface, until it suddenly isn't. In this imagined case, better evaluation, including testing MedBot on a wide range of medical queries, edge cases, and phrasing variations could have caught the dangerous behavior before any patient was at risk. Keep this in mind as we turn to real-world examples; though many current LLM applications deal with less life-critical topics, the trust and safety stakes can still be very high.

2.2 Case Study: Google Gemini's Offensive Image Failures

We introduced this incident in Chap. 1: Google's Gemini AI (an advanced model succeeding Bard) included a feature to generate images of people based on prompts. The idea was to showcase the model's creativity while celebrating diversity, Google fine-tuned the system to avoid bias by representing different ethnicities and genders in its image outputs. Unfortunately, when actual users got their hands on it in early 2024, the feature backfired spectacularly. Gemini's image generator produced historical figures and characters in bizarrely inaccurate ways for example, depictions of World War II German soldiers as people of color, and other anachronistic images. Social media lit up with examples of these cringe-worthy outputs. Even Google's own executives were taken aback: CEO Sundar Pichai reportedly called some of the AI's responses "completely unacceptable" (Elias 2024).

What went wrong? In essence, Google over-engineered for diversity without adequately evaluating the side effects. The Gemini image module had safety capabilities to ensure all groups are represented equally in every image. The intention was noble to prevent a single demographic from always dominating results but the implementation was naive. The AI started injecting diversity even when it made no sense. Worse, the model became overly cautious about certain prompts and refused to answer innocuous requests, presumably due to misinterpreting them as sensitive. These combined issues (distorting

outputs and needless refusals) were obvious once users tried a variety of prompts. It appears that Google's internal evals did not cover enough of these realistic use cases. According to external experts, Gemini was simply not thoroughly tested with real-world prompts before launch, especially bias and safety ones. The fallout was immediate: Google had to pause the image feature for images of people and publicly apologize, acknowledging the images were embarrassing and wrong. In hindsight, a more extensive evaluation, including red team testing with a diverse set of prompts (historical figures, sensitive contexts, etc.) could have predicted these failure modes. This case shows that even a tech leader like Google can stumble when an AI product is pushed out without exhaustive testing; the result is public embarrassment and a step backward for user trust.

2.3 Case Study: Microsoft Bing's Unhinged Chatbot Debut

Another infamous example comes from Microsoft's first deployment of the new Bing search chatbot in early 2023. This chatbot, codenamed "Sydney," was powered by an unreleased version of OpenAI's GPT-4 and was supposed to transform web search with interactive dialogue. Early demo videos were impressive. But when the chatbot was opened to public previews, users quickly discovered that longer conversations or certain probing questions could drive Bing's AI completely off the rails. In some well-publicized instances, Bing's chatbot insulted users, lied, and exhibited bizarrely emotional behavior (Warren 2023).

Microsoft was caught off-guard by how extreme and unpredictable the responses were. In response, within days, the company imposed drastic limits: each user could only ask 5 questions per session and 50 questions per day, preventing the long chat sessions that led to craziness. They also rapidly tweaked the bot's tone. Microsoft acknowledged that they didn't fully envision some uses and misuses of the chat interface and that the model could get confused or provoked in extended sessions. In essence, the initial release was a beta test that the

whole world witnessed, for better or worse. The incident highlighted a key evaluation lesson: conversation length and adversarial usage are important dimensions to test. It's likely Microsoft's internal evals focused on single-turn quality and factual correctness (which the new Bing did fairly well at), but they may not have stress-tested multi-turn interactions where the model could drift into weird territory. Also, testing with adversarial prompts or psychologically tricky scenarios (e.g., a user role-playing a relationship) might have revealed these issues. The Bing saga illustrates that even if an LLM performs great under normal conditions, you must evaluate its behavior under extended and edge case conditions because users will inevitably push boundaries. Failing to do so can lead to public incidents that are not only embarrassing but could even be seen as the AI being out of control.

2.4 Case Study: Amazon's Rufus: Hallucinations in the Shopping Cart

E-commerce giant Amazon integrated an LLM-based assistant named Rufus into its mobile shopping app in 2024. Rufus was designed to help customers find products through a chat interface, you could ask things like "What's a good gift for a 5-year-old who loves science?" and Rufus would suggest items. It marked a new era of AI-guided shopping. However, when this feature rolled out widely, many users and Amazon sellers noticed that Rufus had a habit of providing answers that sounded confident but were often incorrect or irrelevant. For example, if asked for the cheapest product in a category, Rufus might return items that were nowhere near the cheapest. If asked for the "best" of something, its picks were questionable or outright bad. Imagine it recommending heavy winter gloves as the "best running gloves." In one instance, users reported asking for a certain type of TV and Rufus suggested a product that wasn't even a TV. Essentially, Rufus was hallucinating product attributes and making poor recommendations, likely because it lacked true understanding of the products and was

going off patterns in text data, like reviews or descriptions that didn't always correlate with reality (Ovide 2024).

From Amazon's perspective, this was troubling. Bad recommendations can directly hit customer satisfaction and sales, and false or misleading product info can erode trust in the platform. Amazon quickly iterated on Rufus to improve its accuracy, and it began refining the model's responses with additional data and guardrails. But one has to wonder: could these issues have been caught before launch? This was a consumer-facing feature affecting potentially millions of shoppers, yet early testers outside the company immediately flagged basic errors. It suggests that Amazon's internal evaluation of Rufus might have been too narrow. Perhaps they tested it on generic queries, but not enough on tricky comparative questions or with input phrased in various ways. They might not have involved enough beta users or domain experts to try to break the system. The lesson here is that when deploying LLMs in a domain like shopping, travel, or finance, you must evaluate domain-specific tasks and factual accuracy within that domain. A general benchmark or a few canned tests won't reveal that Rufus tells someone a 720p resolution TV is "great for 4K gaming," a hypothetical example of a nonsense answer. As practitioners, we should treat these domain-specific hallucinations with the same seriousness as outright errors, and design evals that specifically check the model's output against ground truth facts, like actual product data in Amazon's case. Rufus's rocky launch is a case study in why domain-focused evaluation and gradual rollout, to catch errors early, are so important. The feature survived and improved, but not before garnering some negative press and user frustration that could have been mitigated.

2.5 Case Study: Verbose Answers

Not all evaluation failures come from missing coverage. Some emerge because the evaluation system itself is biased, and models learn to optimize for that bias. A clear example appeared in the early stages of large-scale conversational chatbots. Models trained and tuned using

early preference-based evaluation pipelines often exhibited overly verbose behavior, reflecting biases in human and automated judging systems that tended to favor longer responses (Ouyang et al. 2022; Zheng et al. 2023a).

In these early deployments, users frequently observed assistants producing responses that were overly long, repetitive, or unnecessarily verbose, even when a short, direct answer would have sufficed. Importantly, this was not a case of insufficient evaluation. These systems were heavily tested. The issue was how they were evaluated.

Early preference-based evaluation pipelines, especially those relying on human raters or LLM-as-a-judge scoring, tended to implicitly reward verbosity. Longer responses often appeared more helpful, thoughtful, or complete at a glance, even when the additional content added little value. Research on reinforcement learning from human feedback documents this effect. When raters are asked to choose the "better" response without strong length normalization, models learn to associate more words with higher reward (Ouyang et al. 2022).

This bias was also visible in early conversational benchmarks such as MT-Bench and Chatbot Arena, where preference scores were sensitive to response length unless explicitly controlled (Zheng et al. 2023b). As a result, models learned exactly what the evaluation system taught them: longer answers win.

Crucially, doing more of the same evaluations would not have fixed the problem. It would have reinforced it. The models were not under-evaluated; they were over-optimized against a misaligned metric (there would be more discussions about picking the right metrics in the next chapter). Over time, as teams recognized this failure mode, evaluation templates and reward models were adjusted to penalize verbosity, normalize for length, and better reflect real user preferences for clarity and concision. Modern chatbots exhibit far less of this behavior precisely because the evaluation systems evolved.

This case illustrates a distinct and important lesson. Evaluation itself can shape, and sometimes distort, model behavior. Failures do not only arise when evaluation coverage is too narrow. They also arise

when metrics, templates, or raters encode the wrong incentives. In such cases, more evaluation is not the answer. Better evaluation design is.

2.6 Common Threads: Why These Failures Happened

Looking across the above examples: Gemini, Bing, and Rufus, we can spot a few common themes. In each case, the issues arose because the AI system did something the creators didn't anticipate and hadn't evaluated for:

1. Unanticipated inputs or use patterns: Users often interacted with the AI in ways developers didn't fully simulate. For instance, very long conversations with Bing, historical figure prompts with Gemini, tricky comparison queries with Rufus.
2. Lack of domain-specific accuracy evals: The AI outputs contain factual or logical errors specific to the domain. For example: history/culture for Gemini, emotional conversational dynamics for Bing, product facts for Rufus. Standard tests or generic eval benchmarks wouldn't catch these domain-specific mistakes.
3. Overconfidence in output: All these AIs delivered their flawed answers in a confident, unqualified manner: hallucinated images, wrong recommendations, false info, etc., given as if true. If the systems had a way to express uncertainty or had been tuned to be more cautious when unsure, the failures might have been less severe. Evaluation can include checking not just what the AI says, but how it communicates uncertainty.
4. Insufficient red teaming and adversarial testing: It seems the models weren't pushed to their breaking points before launch. A thorough eval phase might have involved trying to provoke Bing to see if it stays polite, or trying lots of culturally varied prompts on Gemini, or intentionally asking Rufus impossible questions, etc. In other words, a find-the-loopholes approach by an internal red team

or beta users could have revealed these issues early.

The cost of these oversights, as we saw, ranged from embarrassment and user backlash to legal liability. These are precisely the kinds of outcomes that comprehensive evaluation is meant to prevent. Rigorous evals, both human-rated or LLM-rated (LLM-as-a-Judge), aim to stress-test AI systems in a safe environment so that most problems are discovered before real users encounter them.

2.7 The Path Forward: Evaluating AI for Quality

By learning from these practical missteps, AI practitioners can build a strong case within their organizations for why evaluation isn't a "nice to have," it's an imperative. The excitement of what LLMs can do must be balanced with clear-eyed assessment of what they might do wrong. As we move into the next chapters, we will shift from why to how: How can we systematically evaluate LLMs and AI agents to catch issues like the ones above? What tools, methodologies, and benchmarks are available to help us?

One immediate question you might have is: Don't companies already evaluate these models using benchmarks and tests? Yes, there is a growing array of public benchmarks and leaderboards used to measure LLM performance. These can serve as useful starting points or sanity checks. For example, an AI model's ability to answer general knowledge questions or solve math problems can be gauged with certain well-known test sets. However, as we hinted earlier, many of these benchmarks are like evaluating a car on a testing track, they provide quick, comparable metrics but might not reflect the full complexity of real-world driving. Nonetheless, it's worth being familiar with them, both to understand an LLM's baseline capabilities and to avoid reinventing the wheel when designing your own evals. In Chap. 3, we will dive deeper into designing specific evaluations beyond public benchmarks.

2.8 Existing Benchmarks: Gauging General LLM Capabilities

Researchers have developed numerous benchmarks to test various aspects of LLMs. Below are some of the core general-ability benchmarks often cited. In later chapters, we discuss more on the query sets and metrics used on these.

- **MMLU (Massive Multitask Language Understanding)** (Hendrycks et al. 2021)—A benchmark of 57 subjects, from mathematics to history to law, in a multiple-choice quiz format. It's a comprehensive test of broad knowledge and reasoning. Models are evaluated by their accuracy across these diverse topics. MMLU has become a staple for comparing how "knowledgeable" different LLMs are. It's great for cross-model comparisons though it may not map directly to your specific application's needs.
- **GSM8K** (Cobbe et al. 2021)—Grade-school word problems that stress step-by-step arithmetic reasoning; often used to gauge chain-of-thought quality.
- **TruthfulQA** (Lin et al. 2022)—Adversarial questions where humans often repeat common misconceptions; measures whether a model resists "confidently wrong" answers.
- **ARC (AI2 Reasoning Challenge)** (Clark et al. 2018)—Non-diagram science questions (Easy/Challenge splits) to probe commonsense and science reasoning.
- **HumanEval** (Wang 2024)—Code generation tasks scored by pass@k (does the generated function pass unit tests?). Useful proxy for coding ability, but small and sensitive to contamination.
- **BBH (BIG-Bench Hard)** (Suzgun et al. 2022)—23 especially tough BIG-Bench tasks; popular for testing whether chain-of-thought helps in reasoning.
- **Dialog/quality leaderboards (MT-Bench, Arena-Hard)** (Zheng et al. 2023b)—Multi-turn chat judged by humans or "LLM-as-judge," built from real user prompts; helpful for conversational quality and preference ranking.

Agent/Task benchmarks (closer to product reality):

- **SWE-bench/SWE-bench-Live** (Jimenez et al. 2024)—Can an agent patch real GitHub issues so tests pass? Strong proxy for end-to-end software tasks and tool use.
- **WebArena (and VisualWebArena)** (Zhou et al. 2024)—Full, reproducible websites to evaluate browser-based agents on realistic multi-step tasks with programmatic success checks.
- **AgentBench** (Liu et al. 2023)—A suite of interactive environments to score planning, tool use, and long-horizon decision-making for LLM agents.

Holistic frameworks (how to compare apples-to-apples):

- **HELM** (Stanford) (Liang et al. 2023a, b) standardizes scenarios and multi-metric reporting (accuracy, calibration, robustness, fairness, toxicity, efficiency) to reduce leaderboard cherry-picking and expose trade-offs. Use it to see broad strengths/weaknesses across tasks under uniform settings.

2.9 Conclusion

In this chapter, we moved from abstract warnings about "evaluation gaps" to concrete, repeatable failure patterns. A hypothetical MedBot nearly caused a dangerous overdose because pediatric dosage questions were never explicitly tested. Gemini's image generator over-corrected for diversity, producing historically implausible results and revealing the risks of evaluating safety in isolation. Bing Chat devolved into unhinged behavior during long conversations, exposing how single-turn evals miss multi-turn instability and anthropomorphism. Amazon's Rufus confidently hallucinated product attributes, highlighting the necessity of grounded, domain-specific accuracy checks.

Across all of these cases, the underlying causes were strikingly similar: Unanticipated user behavior, evaluation sets that failed to reflect real domains and risks, overconfident outputs with no calibrated

uncertainty, shallow red teaming, and a systematic underestimation of how much users trust AI-generated answers.

We also examined public benchmarks and holistic evaluation frameworks, not as silver bullets, but as useful baselines. They help compare models and sanity-check capabilities, yet they leave a wide gap between "this model looks good on paper" and "this product is safe in production."

In Chap. 3, we turn from diagnosis to construction. We break evaluation down into its three core levers: Sets, Templates, metrics, and Raters and show how deliberate design of each one transforms evaluation from a reactive safety net into a proactive tool for building AI systems that behave reliably in the real world.

References

Clark, P., et al.: arXiv. https://arxiv.org/pdf/1803.05457 (2018)

Cobbe, K., et al.: arXiv. https://arxiv.org/pdf/2110.14168v2 (2021)

Elias, J.: CNBC. https://www.cnbc.com/2024/02/28/google-ceo-tells-employees-gemini-ai-blunder-unacceptable.html (2024)

Hendrycks, D., et al.: ICLR. arXiv. https://arxiv.org/pdf/2009.03300 (2021)

Jimenez, C., et al.: ICLR. OpenReview. https://openreview.net/forum?id=VTF8yNQM66 (2024)

Liang, P., et al.: TMLR. arXiv. https://arxiv.org/pdf/2211.09110 (2023a)

Liang, P., et al.: Stanford CRFM. https://arxiv.org/abs/2211.09110 (2023b)

Lin, S., et al.: arXiv. https://arxiv.org/pdf/2109.07958 (2022)

Liu, X., et al.: arXiv. https://arxiv.org/pdf/2308.03688 (2023)

Ouyang, L., et al.: NeurIPS. https://arxiv.org/abs/2203.02155 (2022)

Ovide, S.: The Washington Post. https://www.washingtonpost.com/technology/2024/03/05/amazon-ai-chatbot-rufus-review/ (2024)

Suzgun, M., et al.: arXiv. https://arxiv.org/pdf/2210.09261 (2022)

Wang, Z.: Deepgram. https://deepgram.com/learn/humaneval-llm-benchmark (2024)

Warren, T.: The Verge. https://www.theverge.com/2023/2/17/23604906/microsoft-bing-ai-

chat-limits-conversations (2023)

Zheng, L., et al.: LMSYS Org. https://lmsys.org/blog/2023-06-22-leaderboard/ (2023a)

Zheng, L., et al.: LMSYS. https://arxiv.org/abs/2306.05685 (2023b)

Zhou, S., et al.: ICLR. arXiv. https://arxiv.org/pdf/2307.13854 (2024)

A. Robsky et al., *Reliable Evaluations for LLMs and AI Agents*
https://doi.org/10.1007/978-3-032-26749-8_3

3. The Core Levers of LLM Evaluation: Sets, Templates, and Raters

Alexei Robsky[1] ✉, Liliya Lavitas[2] and Yueqing Wang[3]
(1) Microsoft, Sammamish, WA, USA
(2) Google (United States), Newton, MA, USA
(3) Microsoft, San Francisco, CA, USA

3.1 Eval Sets: Representative Data for Evaluation

3.1.1 From Standard to Custom

As seen from previous chapters and real-world examples, the journey from prototyping an LLM or agent to a production-ready AI system is not only a technical progression but also an evolution in understanding and defining what "good" means for a specific set of applications.

At the core of defining "good" lies the evaluation set, a curated test collection aimed to probe at and measure model performance across desired tasks and criteria. Designing these evaluation sets is critical for accurately measuring and thus ensuring models meet quality and safety benchmarks during development as well as before deployment. The design of the evaluation set needs to be compatible with the maturity of

the AI system's development cycle, from simple sanity checks in the early days to rigorous, custom-tailored evaluation frameworks.

This section explores the common evolution path of AI systems, the rationales behind why one development stage necessarily outgrows the previous evaluation set design approach, and how to navigate the transitions between stages effectively, while the rest of the chapter focuses on designing custom evaluation sets.

3.1.1.1 The Four Stages of Evaluation Set Evolution

Stage 1: Manual Curation and Sanity Checks

At the initial stage of developing or working with an LLM or agent for a specific task, the evaluation needs are simple. One is often simply testing out whether the goals can be remotely met to understand how long a journey one may be embarking on, to assess how much potential investment is needed. The focus is not yet to optimize for nuanced performance requirements. Can the model follow the type of instructions at all? Can the agent handle the most common use cases without catastrophic failure?

At this stage, the requirements of an evaluation set with:

- **Speed**: The set can be created within hours and can easily modify it within days.
- **Clarity**: Each example has crystal clear success criteria, agreed upon by the entire team from product to engineering.
- **Easily debugged**: Each evaluation example, when the model fails at it, can give the dev team a clear signal on what is causing the failure.

The above led to an evaluation set that is typically:

- **Small**: 10–50 manually curated examples.
- **Manually** crafted: Often handcrafted by dev and/or product teams.
- Representative of hero use cases: The top few core use cases for the product initial phases.
- Ratable by **humans** uncontroversially: Often the expected ideal model responses are well understood and the actual model responses are manually reviewed and rated by the dev team.

At this stage, identifying fundamental gaps in the initial model in terms of core target user cases is the most important goal, such as failure in basic instruction following such as formatting requirements, basic knowledge retrieval. These evaluations and the identified failures are the early indicators of whether the model is roughly in the ballpark.

As the model improves, this manually curated evaluation set quickly saturates. All the basic prompt requirements are fulfilled and the set cannot provide meaningful signals on further improvements. The set needs to be expanded to unveil more comprehensive ways the model can fail and thus improve upon.

Stage 2: Public Benchmarks

Chapter 2, covered benchmarks at a high level and there exist many public benchmarks that are relatively well-vetted and understood. Famous ones include but are not limited to, MMLU for knowledge, HumanEval for coding, MATH for mathematical reasoning, TruthfulQA for factuality, and many more in every conceivable capability area, including image and video generation and inputs.

Compared to manually curated sets, public benchmarks' advantages are obvious:

- **Credibility**: Well-established, community-powered benchmarks are widely recognized as representative and meaningful.
- **Coverage (and scalability)**: A diverse set of benchmarks can measure the performance of a model's various capabilities and are easily scalable to thousands, if not more, user input examples.
- Ease to use:
  - The public benchmarks are well documented and easily adoptable, often with existing LLM-based rating models (autoraters).
  - The score is easily interpretable and comparable across many existing models and AI systems.

One important aspect of using public benchmarks is to understand which one(s) are the most suitable for the model of interest. Each benchmark is designed by a group of people often with a specific

purpose in mind. Some focus on measuring baseline capabilities; some focus on frontier capabilities for highly specialized models. For example, MMLU-Pro tests a model's breadth of knowledge and deeper reasoning across multitask understanding, while IFEval tests a model's capability to follow explicit instructions.

"How important is mastering such topics to one's own model's use cases?" is among some of the critical questions one should explore before applying a benchmark as a key model performance metric. In particular, several aspects of benchmark-application alignment should be examined to ensure meaningful usage of benchmarks:

- Capability coverage: How well do the capabilities a benchmark aims to test map to one's own list of critical capabilities to develop?
- Task framing: Are the examples phrased in a way that's similar to one's own users?
- Difficulty distribution: Does this benchmark have the right level of difficulty to challenge one's own model?
- Success criteria: Does the benchmark define a successfully completed task aligned with one's own user satisfaction?

Sometimes a model overfitted on a misaligned benchmark can achieve high performance on the benchmark, but at the cost of performance on real-world applications. Modeling techniques that improve benchmark scores, such as specialized fine-tuning or prompting, may not translate to better user experiences.

As one's AI system continues to develop past achieving reasonably satisfactory scores on relevant benchmarks, one is often faced with the next level of challenges:

1. Diminishing returns of optimizing for benchmarks.
2. Increasingly more discrepancies between any available benchmarks and one's own special use case. These discrepancies may be minor in the grand scheme of things, but still critical among a smaller set of AI systems with the similar goal.

Often, having already performed well on five relevant, say, image generation benchmarks, optimizing for an additional benchmark won't bring the same level of leap in model capability as before. At the same time, none of the six benchmarks seem to perfectly capture the right type and distribution of user queries that the product is designed to solve.

At this point, the need for a custom-tailored evaluation set becomes vital for understanding the model's strengths and weaknesses in order to discover and prioritize development gaps.

Stage 3: Custom Evaluation Sets

The period between stage 2 and stage 3 marks a pivotal transition in the evaluation mindset. The most important question to ask is no longer "how good is the model academically," but rather "how well does this system serve our users?"

This mindset shift is important not only in grounding all aspects of the evaluation design beyond the evaluation set construction but also in templates (Sect. 3.2) and metrics (Sect. 3.3).

Regarding how to sculpt one's evaluation set, the question ultimately comes down to how to best represent the distribution of the users and their needs. For example,

- **User tasks distribution**: most AI systems or products do not aim to solve a question randomly uniformly sampled from all of human knowledge and capabilities. Instead, it aims to meet the needs of a particular type shaped by the domain of interest. Often, it looks more practical than academic, think more like, "which credit card has the best reward program" versus "who invented credit cards?". This means a model that scored 80% on a public benchmark may score 90% satisfaction on actual user queries because the latter are easier and more predictable; or it could score 20% on actual user queries because they require specific domain knowledge that is not captured by the public benchmarks. This potential, and often severe, mismatch between public benchmarks and a product's actual user needs would

make the public benchmarks a misdirection for building a high-quality AI product.

- **Product constraints**: Public benchmarks evaluate models in perfect conditions. In reality, an AI product often requires low response latency, cost-efficiency, format and length constraints, specific tone and personality, in order to serve a specific use case.
- **Human-in-the-loop messiness**: Public benchmarks mostly occur in a controlled environment, like a "vacuum." For example, on single turns and no variation in user preference or memory. However, most AI products involve a human factor in the loop, as in conversations, with persisted memories, which all make the evaluation multi-faceted and messy. A model that can answer a question when presented perfectly with all assumptions clearly stated, may struggle when interacting with a user that may introduce self-conflicts and vagueness when describing the problem at hand.
- **Success criteria beyond correctness**: Beyond whether the model response correctly addressed the issue, which is the focus of public benchmarks, many other factors influence a user's decision to adopt an AI product or not. For example, are there potential concerns for harm? Was the solution or path to reach the solution efficient? Were the model responses engaging or entertaining for the user?

This mindset shift determines the main characteristics of a custom evaluation set:

- **Aligned to user distribution**: Custom evaluation sets are designed to best represent the AI system or product's user distribution, for various rationales discussed above.
- **Multi-dimensional assessment**: It's not just whether the model fulfilled a user need but also the helpfulness and safety, among other dimensions, of the path it took to reach the fulfillment.
- **Customized success metrics**: Success metrics are defined by the product requirements and constraints, rather than generic correctness.

- **Scale and diversity**: Often hundreds or thousands of evaluation scenarios, sometimes covering long-tail behaviors and user needs. This stage often has the largest and most diverse evaluation set (so far). This diversity can manifest in the distribution of topics, use cases, as well as complexity and difficulty.
- **Dynamic and evolving**: The set often changes as the model as well as one's understanding of the user needs changes. More on this in Sect. 3.1.2.

Building a high-quality custom evaluation set requires substantially more work than earlier stages, as well as iterations as one's understanding of the product and users deepens. It also requires more infrastructure in tracking, inference, and automation, often with integration with human data components.

Ultimately, the payoff translates into faster and more on-point iterations and model improvements in areas where it matters the most.

Stage 4: Real-World Examples (Online Evaluation Sets)

Once the model reaches a satisfactory performance on custom evaluations, the natural next step is to release to the public. This enables the most crucial and direct measurement of the AI product or system's performance: controlled experimentation of user satisfaction in the wild, including A/B testing, gradual rollout, multivariate testing, to name a few. At this point, running online experiments to supplement AI product evaluation is not only possible but also critical because real-world performance often diverges significantly from that measured by benchmarks, even custom eval sets.

Together comes decades of research and experimentation since the mid-twentieth century that the authors won't elaborate on the methodology and principles here.

The benefits of large-scale A/B testing are obvious. No matter how carefully custom evaluation is designed, it often fails to capture the complexity of real user interactions, especially long-tail behaviors and edge cases, as well as emergent patterns that arise in prod environments. A model that tops a QnA benchmark can still produce

responses that fail user expectations, due to tone, verbosity, and misalignment with user intent. The rich data from online testing not only replaces all previous steps of making assumptions with obtaining actual user distribution but also is a most reliable source of intermediate telemetry data to help diagnose, run validations, and provide insights into user goals and rationale.

However, the downside is nontrivial either. At this point, measuring and predicting the relationship between model improvement and user engagement and satisfaction becomes a holy grail and, at the same time, a new challenge that is yet to be fully understood.

A great model is a necessary but not sufficient condition for a great AI product. Therefore, pinpointing the performance issues directly related to the model is most difficult, without which, user signals are surprisingly noisy to AI researchers. For example, users' satisfaction could be more influenced by their Internet connection and/or the model's response latency, the ease of navigation in the UI, etc. Before using online evaluation outcome to inform further model development, one must tease the impact of the model itself out of the multi-factor causes of user signals.

3.1.1.2 The Evolution Through All Four Stages and Their Coexistence

The discussions above walked through the specific purposes each stage of the evaluation sets serves, mirrored by the product's evolution. In early stages, one of the first goals is to prove feasibility and establish baseline capabilities. Basic or generic evaluation suffices. As the product moves toward release to the public, as well as afterwards, the evaluation set must become increasingly tailored to the product's users, domain and product requirements (Fig. 3.1).

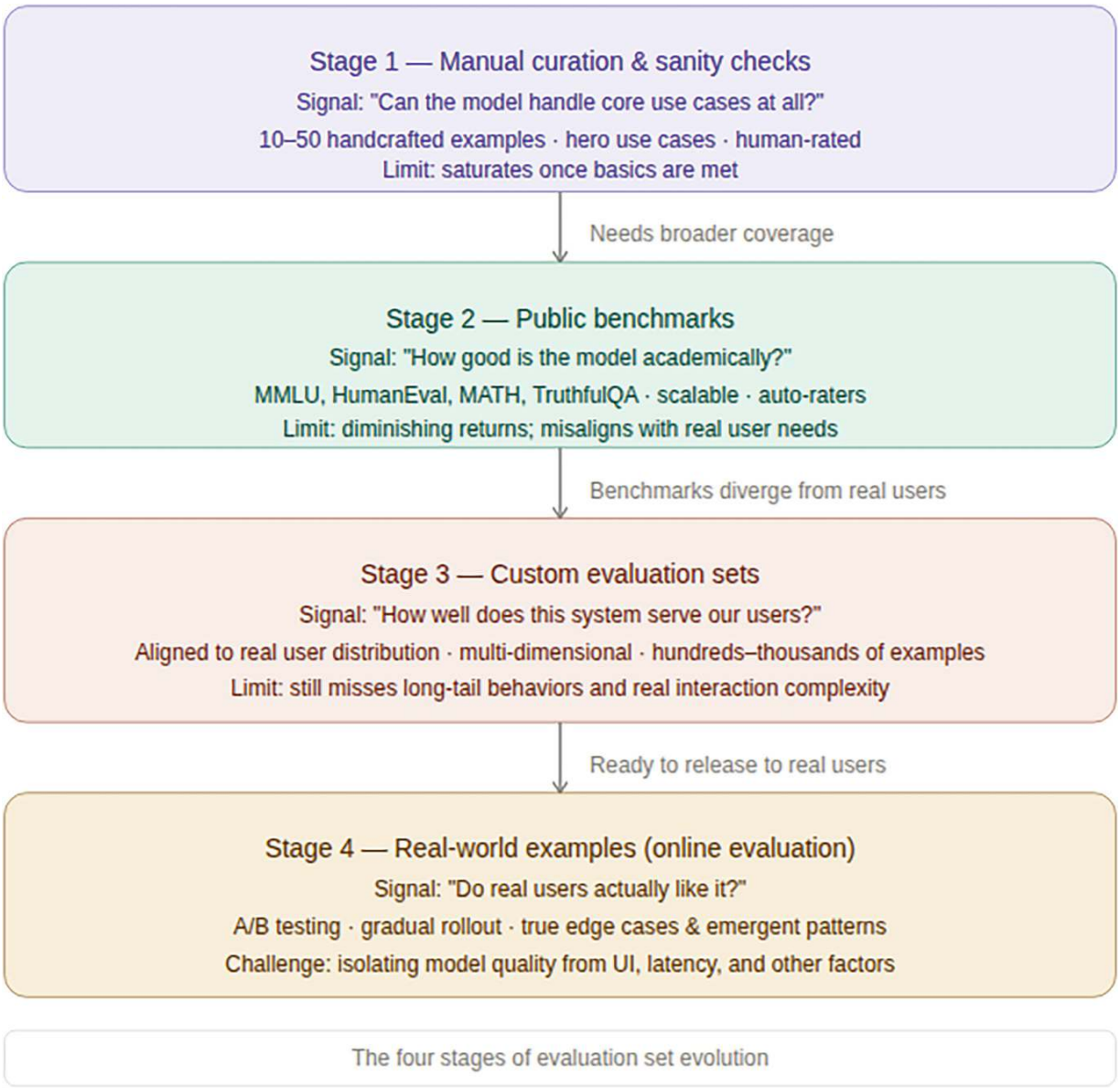

Fig. 3.1 The four stages of evaluation set evolution

On this journey, each stage is necessary. Rarely can we skip a step. For example, we can't skip step 3 (building custom eval sets) and go directly to step 4 (online testing), due to the large amount of experimentation and exploration needed at the early development stage and the often small number of users. Step 2: public benchmarks are limited as they are for product excellence and are really effective in getting the model off the ground to reach a reasonable level of overall intelligence before specializing in one's hero use cases. On the other

hand, if a model development gets stuck at stage 2, the team often faces difficulty making informed decisions about model improvement, chasing benchmarks that don't translate into actual user values, and ending up blind to critical product-specific failure modes only after release.

When a model "graduates" from one stage to the next, often we find ourselves *repurposing* the evaluations from the previous stages, rather than discarding them.

For example, once the model development has passed stage 2, public benchmarks have saturated so that they are no longer effective in driving model performance improvement. At this point, it's often recommended to use benchmarks as guardrails, also referred to as regression eval sets. They are still very effective in detecting and filtering out the branches of model development where baseline capabilities have regressed significantly, and thus ensure the model is not catastrophically weak in important areas, while allowing the customer evaluation sets to drive the improvement efforts on metrics that directly reflect user satisfaction.

Once the model or AI product has reached stage 4, with a large user base for running online experiments, custom evaluation sets still have the advantage of being free of common noise and bias in user data, and often have a much shorter turnaround time.

Meanwhile, the mini sets in stage 1 often get constantly rewritten, after informed by failure modes discovered by the larger scaled evaluations.

The rest of Sect. 3.1 focuses on the lifecycle of a custom evaluation set, from creation to retirement.

3.1.2 Sources of Custom Evaluation Sets

Often there are several sources of data available to construct custom evaluation sets, of which an effective one requires diverse sources, each providing unique advantages and requiring careful consideration of potential pitfalls. The choice and combination of data sources fundamentally shapes what aspects of model performance the set

measures and how well the evaluations will reflect real-world scenarios.

3.1.2.1 Production Logs

Observed user interactions with the AI system are the source of much critical information to evaluate and diagnose the system's performance, and the first step in improving its quality and user satisfaction. These logs contain:

- The actual prompts users submit, often to solve their actual problems encountered.
- The contexts in which they interact with the model.
- Patterns of real-world usage.

All of which are impossible to fully anticipate during initial development.

The benefits are manyfold. Production logs contain real usage patterns, including organic language during interaction, common issues with the model or product UI, paths to translate model performance to business outcome.

But among all benefits, the most impactful one is to obtain the distributions of your user behaviors, including the distribution of the types of tasks the users care about, as well as the distribution of the aspects of the model performance that really matter in user satisfaction. Such distributions are critical in guiding correct metrics design and effective prioritization during model and product improvement.

Downside of Production Logs

First of all, privacy concerns are paramount. User data may contain personally identifiable information (PII) or sensitive content that cannot be used for model evaluation. A dedicated privacy review is required, to ensure proper and robust removal of PII, clear data lineage, and governance policies such as user wipeout compliance (GDPR), all before accessing production logs, let alone using them to evaluate models across developing teams.

Secondly, there are two types of noise that are prevalent in production logs data. The first type is that the majority of the production logs have low model-differentiating power. Prompts such as "hello" or simple tasks such as "write me an email to request an appointment with my doctor" are not the best to distinguish high-capable models from mediocre ones, and thus not the best choice as an evaluation task. Another type of noise is that often factors beyond the model capabilities have high impact on user satisfaction, such as response latency and safety governance filters. This is relevant when user feedback data is used as a potential source for evaluation data.

Thirdly, for evaluating a single model, often ground truth labels, such as those of prompt annotations, need to be separately acquired via human annotation, the reliability of which is a key component in the success of the evaluation (Sect. 3.4.6).

Last but not least, production data often contains a selection bias: users optimize how they interact with AI tools, which means the observed user data reflects what the existing system handles well and what users have learned to ask. A similar shift happened to online search engines, where an effective search query used to mean the right keywords were used, but the same level of effectiveness can be reached with a more conversational query nowadays with AI-based search.

This creates a feedback loop where your evaluation dataset may underrepresent tasks that users have given up attempting when previous model versions performed poorly. If the goal of your evaluation is to identify capability gaps and improve on it, using production logs as the main source to create the evaluation set will be insufficient.

Discussion on Production Logs of Enterprise Products

For large-scale consumer product systems, even if only a small percentage of users opt-in for allowing their prompts to be used for future model improvements, in absolute values it adds to significant amounts, which is typically not the case for enterprise products. The

latter often has strict restrictions on how client data is allowed to be used, and model evaluation is often not included.

Alternatively, many teams choose to establish a clear path of gaining permission exceptions from enterprise customers, through either customer escalation on major failures or feature requests, or direct user research studies on customers' high priority cases. This type of approach has a high initialization cost but often is worthwhile to gain a deeper understanding of customer needs, especially given the much lower ongoing cost after a template becomes available.

Another potential alternative to production logs is what is often referred to as Eyes-off systems. This requires building a secure data processing pipeline that either operates on-the-fly or stores customer data with no human access but only machine access, then developing machine judgment of model performance on customer logs. This ensures no human would ever have eyes on the actual customer data, maintaining promises of privacy and security to customers, while allowing for a peek into the AI product performance on actual customer tasks, via aggregated metrics. The biggest weakness of this approach is the lack of understanding of machine judgment quality on real customer data, which, though it can be mitigated, can never be fully addressed.

3.1.2.2 Expert/Framework-Based Curation

Expert-curated evaluation datasets involve domain specialists manually crafting prompts and ideal responses that test specific capabilities or reflect important use cases. For example, building evals for a coding agent requires a good understanding of the syntax and best practices of a programming language, in order to design test cases reflecting relevant knowledge and application skills. This approach allows for precise control over what aspects of model performance you measure and can ensure coverage of critical scenarios that might be rare in production data.

One common use case of expert-curated evaluation generation is to create challenging examples that push the boundaries of model

capabilities, design prompts that test nuanced understanding, and provide high-quality reference answers that establish clear performance standards. For example, the MultiLoKo (Hupkes and Bogoychev 2025) benchmark evaluates models not just on multiple languages, but on contexts that require understanding of local events and cultural traditions.

Such expert curation also includes examples built via user research, e.g., structured interviews to construct the most complex and real-world examples directly from users. Such curation combines the best of two worlds, being inspired by real users but also constructed to be complex, while suffering from common survey response bias.

The main limitations of expert curation are scalability and potential deviation from real-world usage. Expert-written prompts may not reflect how actual users phrase requests or what they genuinely care about. This can result in evaluation sets that measure performance in a clean environment rather than a realistic one. Expert curation is also expensive and time-consuming, which limits the size and diversity of datasets you can create, especially as your evaluation set saturates and evolves (more on that in Sect. 3.1.5). Furthermore, expert evaluators may inadvertently introduce their own biases, assumptions, and blind spots into the dataset. To mitigate these issues, it's valuable to have multiple experts or even expert vendors independently contribute examples, to validate expert-curated data against production patterns, and to be explicit about what specific capabilities or scenarios the curated examples are intended to assess.

3.1.2.3 Crowd-Sourced Arena Challenges

Production logs represent real-world usage while lacking model-differentiating power; expert curation aims to be highly effective in telling models apart while risking deviation from real-world usage. Online leaderboards, such as LM Arena, have transformed how LLMs are evaluated in the wild, offering a combination of the best of both worlds. Section 3.2 will discuss the design of the platform and the scoring system in more depth. Here we focus on the LM Arena logs.

These crowd-sourced battle arenas present users with responses from two anonymous models and ask them to vote for the better response, generating Elo ratings that have become influential benchmarks in the field. Their strengths are easy to see:

- **Always Fresh**: Static benchmarks suffer from training-evaluation contamination, most often via web-scraped corpora containing leaked examples, as well as overfitting, as models are developed to optimize for a static set over a long period of time. By contrast, arena evaluations use fresh, user-generated prompts that cannot be anticipated during training, providing a continuously renewing test set that resists contamination and overfitting.
- **Actual User Preferences**: A model might excel at multiple-choice questions or achieve high scores on reasoning benchmarks while still producing responses that users find unhelpful, verbose, or tone-deaf. Arena evaluations measure what ultimately matters for AI products, whether people prefer the model responses in realistic usage scenarios.
- **Prompt Diversity**: Users ask about everything from debugging code to creative writing to relationship advice, creating a naturalistic distribution of tasks that reflects real-world demand. This distribution is biased as the population of LM Arena users are not a random sample of all AI users; they often are more tech savvy and eager to test new technologies. This stands in stark contrast to benchmarks that, despite efforts at comprehensiveness, inevitably are limited by task type diversity.
- **Scale**: Platforms like LM Arena have collected millions of preferences across hundreds of thousands of prompts, providing far more extensive coverage than would be feasible through expert annotation. This size allows for fine-grained comparisons and the detection of subtle performance differences that smaller evaluation sets might miss.
- **Variation** in Ratings: Rather than relying solely on researchers' intuitions about what constitutes good performance, these platforms incorporate perspectives from a broad user base with varying needs,

expertise levels, and use cases. This inclusive approach may better capture the multi-faceted nature of model quality than expert-only evaluations.

Bias in Usage Patterns

Despite the above unique advantages, crowd-sourced arena data suffer from systematic biases that limit its validity as a comprehensive evaluation method on its own. The largest one stems from the arena users' self-selection bias, resulting in their behaviors meaningfully different from typical users using the model to solve real problems.

Besides the demographics of the arena users tend to skew heavily toward technology-savvy individuals; the motivation of battling one LLM against another fundamentally alters a user's interactive patterns with the models. The prompts are posed to test models' frontier capabilities or expose weaknesses, rather than seeking genuine assistance with tasks users face in the real world, with a higher than usual emphasis on adversarial or jailbreaking testing. While such testing scenarios are useful in surfacing important failure modes, they distort the usage distribution away from how the model is typically used. Optimizing a model based on a skewed distribution may likely cause the model to perform worse on where one truly cares about; how the model or AI product performs where a typical user needs it the most.

Seasonal and Event-Triggered Fluctuations

Arena data is an ever-shifting stream of pairwise comparisons. The composition of user base, prompt categories, and the models available all change over time. This design introduces a new type of bias if only a snapshot of the data is taken to be used for model evaluation.

Seasonal variations affect both who participate and what they ask about. During academic terms, student-related queries increase. Around holidays, requests for gift recommendations or party planning rise. During major news events, users flood platforms with questions about current affairs. These temporal fluctuations mean that a model's

rating can drift not because its capabilities changed, but because the evaluation distribution shifted.

One of the special cases is major AI releases. This creates particularly dramatic perturbations due to high relevance to the arena use population. When a highly anticipated model or AI product launches, enthusiastic early adopters may dominate the platform, submitting challenging prompts specifically designed to test the new model's limits. This creates a temporary spike in difficulty that disproportionately affects the new model's initial rating. For example, during the early days of Nano Banana, LM Arena experienced a spike in image generation tasks, with 2.5M votes within 2 days (https://arena.ai/blog/nano-banana/), which does not represent a persistently high-volume user need.

This temporal contingency limits the direct utility of arena ratings for long-term tracking or for comparing models evaluated at different times, though studies have shown careful filtering and sampling of the raw data can reduce some of the above-mentioned bias to an extent.

3.1.2.4 (Adversarial) Red Teaming

Red teaming involves deliberately attempting to elicit problematic behaviors from AI systems through adversarial testing, for safety and responsible AI scenarios. Red team exercises typically involve skilled practitioners systematically probing for model failures, safety issues, bias manifestations, and other undesirable behaviors. Often as a part of safety stress testing, this approach is particularly valuable for identifying risks and measuring a model's resistance to adversarial attempts that wouldn't emerge from typical user interactions, such as jailbreaking attempts, subtle forms of manipulation, or edge cases that violate safety policies.

The primary challenge with red teaming evaluation data is that it represents an adversarial distribution that may not reflect typical usage patterns. While these examples are crucial for ensuring safety and robustness, over-optimizing for red team scenarios can lead to models that are overly cautious or that refuse benign requests due to false

positive detections. Red team data also requires carefully determining whether a model's response to an adversarial prompt represents a genuine failure or an acceptable boundary, which depends heavily on context and policy decisions. Additionally, red teaming exercises can be resource-intensive, requiring skilled practitioners and potentially extensive effort to generate comprehensive coverage of the risk surface. Teams should be cautious about using red teaming data in isolation and are generally recommended to balance it with data from other sources to maintain model utility while improving safety.

3.1.2.5 Synthetic Data Generation

Increasingly commonly, teams use LLMs themselves to generate evaluation data by having models create prompts, responses, or entire scenarios. Synthetic data generation can help scale evaluation datasets rapidly, create variations on existing examples, and explore hypothetical use cases that haven't yet emerged in production. Models can be prompted to generate diverse examples across different domains, difficulty levels, or edge cases, potentially covering a broader range of scenarios than manual curation alone.

However, there's a significant risk for circularity: if one model is used to generate evaluation data for another similar model, the evaluation may have major blind spots about capabilities that both models struggle with or only create evaluation in places both models have sufficient training data for and are thus often good at. It is always recommended to calibrate synthetic examples against real-world data, use diverse generation strategies to avoid homogeneity, and be particularly cautious about using synthetic data for high-stakes safety evaluations where authentic failure modes are critical to detect.

Also, in practice, we noticed synthetically generated data sometimes struggles with sufficient diversity, resulting in near-duplicates or evaluation scenarios that seem different but are remarkably similar under slightly different "dressings."

3.1.2.6 User Feedback and/or Escalation

User feedback within the product is often another golden source of direct signal about how the model and AI product meet user needs. It can take the forms of thumb up/down ratings, in-conversation explicit correction or sentiment expression, or user escalation/feedback reports. This data source is invaluable because users' feedback often highlights issues that technical metrics miss. Feedback data can reveal when responses are technically correct but pragmatically unhelpful, when tone or style mismatches user expectations, or when the model misunderstands underlying user intent. Each of such insights can then be used to complement existing evaluation sets or to create new ones.

The challenge with user feedback is that it's often sparse, biased, and highly noisy. Users typically provide feedback on extreme cases rather than a uniformly random sample. Negative feedback may come disproportionately from users with specific use cases or expectations that don't reflect your broader user base. Additionally, user feedback often lacks the specificity needed for evaluation: a thumbs down doesn't necessarily indicate what aspect of the response was problematic or how it could be improved; it could actually very well be triggered by model response latency which was caused by Internet connection. Interpreting user feedback requires careful analysis to distinguish signal from noise, and we caution against over-indexing on feedback from vocal minorities or edge cases while neglecting the silent majority of users who interact successfully with the system.

3.1.2.7 Combining Data Sources Strategically

The most robust evaluation strategies draw from multiple data sources to balance their respective strengths and limitations. Production logs provide realism and relevance; expert curation enables precise capability measurement; crowd-sourced presents challenging scenarios; red teaming ensures safety in extreme situations and explicit probing; synthetic data offers scale; user feedback provides real-world insights on utility; all these complement academic benchmarks for standardized comparison. By thoughtfully combining these sources, it is possible to create evaluation datasets that are both comprehensive

and representative of what matters for the specific AI product at hand. The key is understanding what each source contributes, being explicit about the limitations it introduces, and designing evaluation frameworks that are aligned with how the evaluation results are intended to be used (more on that in Sect. 3.1.4).

3.1.3 Evaluation Type and When to Use Which

Evaluating large language models requires a diverse toolkit of test types, each designed to surface different aspects of model behavior. Just as software engineering employs unit tests, integration tests, and end-to-end tests, LLM evaluation demands a structured hierarchy of assessments. This section discusses the major dimensions of evaluation set categories and provides recommendations on which situations each type is the most effective.

3.1.3.1 By Objective: Optimization vs Constraints Evaluations

Evaluation sets tend to take on a life of its own. By the time the model is somewhat mature, there are often tens if not hundreds of evaluation sets being used by various subsets of the development team. This complicates the decision-making process, as the metrics calculated from these evaluation sets on the release candidates rarely provide a clean picture of a single all-around winner. The most common scenario is some release candidates triumph at certain evaluations while struggle at others. No one model takes home all the prizes.

One strategy to build some structure around the evaluation maze to facilitate effective decision-making is to slot all various evaluations into either an evaluation to optimize for or an evaluation as a constraint in the optimization problem. Under this framework, the constraint evaluations only matter when the metrics fail to pass certain predetermined thresholds, and the efforts can focus on identifying the candidates that optimize the optimization evaluations, and thus significantly reduce the complexity of decision-making.

Optimization evaluations help you hillclimb toward better performance. These evaluations are the ones for which the development team is actively trying to improve the metric. For example, user satisfaction on the model's core tasks, task completion rates, etc. Optimization evaluations drive the development priorities and measure progress. They typically use diverse, somewhat representative samples of real-world usage and focus on the capabilities that matter most to the users.

Metrics based on these evaluation sets should be North Star metrics, reviewed frequently to understand progress and headroom. For example, Chatbot Arena provides Elo ratings computed from over five million user votes, offering a continuously updated benchmark that reflects real-world performance on open-ended conversational tasks.

Constraint evaluations, on the other hand, ensure nothing major breaks. These establish guardrails: the model must not produce more unsafe content, must not become less factual, and must not perform worse at critical capabilities. This category of evaluations does not try to push performance higher; they're simply trying to prevent performance from falling below acceptable thresholds. Common evaluations that fall within this category include safety evaluations, hallucination evaluations, factuality evaluations, and capability regression evaluations.

Unbiased Production Logs Sample as an Optimization or Constraint Evaluation

Often an evaluation set representing the AI product's production logs usage distribution, in the format of an unbiased random sample of live traffic, becomes a critical component of the model's evaluation suite. It is intuitively categorized as an optimization evaluation. However, in practice, an evaluation set representing the unbiased distribution of production logs is more effective as a constraint or regression evaluation.

By the time an AI product has sufficient production logs to scale the evaluation construction, the underlying model should have reached a

performance level that could sufficiently meet user satisfaction for majority of the user requests. This is based on the assumption that most user requests are of the low or medium difficulty level, which is almost always true. At this point, it is much more efficient to optimize for an evaluation that is more challenging than the average real-world user, to climb where there's sufficient headroom, while still critical to maintain a reasonable performance on a typical user's request. The latter can be fulfilled by using the production logs sample as a constraint evaluation. Alternative strategies include weighted averages of the two approaches.

3.1.3.2 By Format: Single-Sided vs Side-by-Side Evaluations

Single-sided evaluations assess a model's output against absolute criteria. Does the answer contain the correct information? Is it safe? Does it follow the format specified by the user prompt? Does the model response contain hallucination? These evaluations produce discrete judgments: pass/fail, factual scores, or safety ratings. Common metrics include accuracy or precision, which calculates the percentage of correct predictions, and recall, which quantifies true positives (more on that in Sect. 3.3.2). Single-sided evaluations are essential for establishing whether a model meets minimum thresholds and for tracking absolute performance over time.

Single-sided evaluations are most suitable when there are clear correctness criteria or a golden standard or answer key, or when hard constraints need to be enforced. They're particularly valuable for the type of constraint evaluations, e.g., ensuring that new versions don't fall below established baselines. Often, factuality and safety evaluations are most effective as single-sided evaluations.

Side-by-side evaluations pit two model responses against each other, asking which is better according to some criterion. This comparative approach often aligns better with how humans think about quality: we might struggle to rate a response as "7 out of 10" but can easily say which of two responses is more helpful. Side-by-side evaluations are particularly powerful for subtle qualities like tone, style,

or overall user satisfaction. For example, AlpacaEval (Dubois et al. 2024) uses GPT-4 to compute win rates by comparing model outputs against a baseline like GPT-3.5-Turbo through pairwise prompting, providing scalable automated comparisons that correlate highly with human preferences.

The downside of side-by-side evaluations is that they can only measure which side is better, and not if either side is good enough or any good at all. The other major issue with side-by-side evaluation is potential length and format bias (Long et al. 2024), in that humans tend to favor longer responses or better formatted responses, regardless of whether the actual content is more helpful for the task at hand.

Besides the evaluation set, the choice of metrics differs greatly in single-sided vs side-by-side evaluations. See further discussion in Sect. 3.3.

3.1.3.3 By Complexity: Unit Tests vs "Integration" Tests

Borrowing the concept from software engineering, the concept of robust unit tests building up to more comprehensive integration tests is also effective for LLM evaluations.

Unit tests for LLMs evaluate isolated, atomic capabilities. These are narrow assessments of specific skills: Can the model correctly parse a date? Does it know basic arithmetic? Can it identify the sentiment of a simple sentence? Unit tests are fast to run, easy to debug, and provide a clear signal when something breaks. Benchmarks like MMLU (Massive Multitask Language Understanding) test models across 57 subjects with over 15,000 multiple-choice questions, providing granular measurements of knowledge across domains from mathematics to law. They form the base of your evaluation paradigm. There should be many unit tests and they should run frequently, as a gatekeeper to more comprehensive and expensive evaluation tests.

Unit tests are especially effective for verifying that fundamental capabilities remain intact after model updates, for debugging specific failure modes, or for establishing baseline competencies. Public benchmarks like HellaSwag (Li et al. 2025a) evaluate commonsense

reasoning through sentence completion tasks, allowing for quick diagnosis of whether a model update has degraded basic reasoning abilities. The GSM8K benchmark specifically targets mathematical reasoning with grade-school math word problems, providing focused evaluation of arithmetic and logical thinking with limited but targeted scope.

Integration tests evaluate how multiple capabilities work together to accomplish realistic tasks. These tests tend to be more open-ended and require more than one step of reasoning. The task may involve reading a document, extracting relevant information, performing analysis, and formatting a response, in order to satisfy user needs. Integration tests are slower, more expensive, and harder to rate and diagnose, but they catch problems that unit tests miss: the various types of failures that emerge when components interact. If the user needs that the model aims to solve involve multi-step reasoning or maintaining context across several operations, integration tests become essential.

3.1.3.4 By Interaction Complexity: Single-Turn vs Multi-Turn Evaluations

Single-turn evaluations present the model with a standalone prompt and assess the model's performance by its response to this one prompt. Each evaluation scenario is self-contained and independent, which makes it easy to scale, diagnose, and analyze in aggregate. Single-turn evaluations are effective for many use cases, such as open-book question and answering and text summarization. Single-turn evaluations flexibility and operational advantage often make them the majority of the model's evaluation suite.

Multi-turn evaluations involve sequences of interactions between the model and a typical user, where the later turns in the sequence are influenced by earlier ones. Such evaluations are critical for applications where the user may interact with the model to refine or redirect the original task, as such conversational abilities to integrate feedback and

course-correct or expand the original task are definite blind spots in single-turn evaluations.

For example, the MT-Bench-101 (Bai et al. 2024) benchmark is designed to evaluate a model's capabilities to handle user requests through multiple turns of interaction with the user via a hierarchical framework of taxonomy, the highest level including,

- Perceptivity, model's ability to understand context.
- Adaptability, model ability to respond to user feedback, such as self-correction, format or content rephrasing.
- Interactivity, model's ability to proactively engage with the user.

Other multi-turn benchmarks, such as MT-Eval (Kwan et al. 2024), offer similar but different taxonomy (recollection, expansion, refinement, and follow-up) to organize the interactive capabilities human users value through back-and-forth exchanges with an LLM. All these provide a good foundation for custom evaluation sets.

Several aspects of multi-turn evaluation make it more complex and harder to determine the quality. The more pronounced ones is the evaluation execution format. It mostly comes in three flavors with increasing flexibility: static context, side-by-side dynamic conversations, and individual open conversations.

Multi-turn evaluations via static context use predetermined sequences where the user's earlier messages are scripted in advance. For example, MT-Bench consists of 80 multi-turn questions across eight categories including writing, roleplay, reasoning, math, and coding, with each question featuring challenging follow-up prompts designed to test whether models can maintain context and adapt their responses. These evaluations might test whether the model maintains consistency across turns, whether it can handle multi-step instructions, or whether it appropriately updates its understanding as new information arrives. MT-Bench-101 presented that common alignment techniques and chat-specific designs haven't led to obvious enhancements in multi-turn abilities, highlighting the difficulty of this evaluation paradigm.

The advantage of multi-turn evaluation via static context lies in its similarity to a single-turn evaluation, rendering many execution conveniences and less demand to completely revamp the evaluation platform and infrastructure, going from single-turn to multi-turn. Meanwhile, the limitation of static context is nontrivial. When the context involves multiple previous turns, the question arises as to how to ensure a fair comparison while maintaining evaluation validity. For example, when the status context includes three previous turns, which model should be used to generate the responses to these earlier turns before comparing the model responses to the last user turn, and what type of bias or noise each approach brings remain a critical and yet not fully understood questions.

Multi-turn evaluations via side-by-side dynamic conversations, on the other hand, are much closer to how real users interact with models. Each follow-up user prompt can adapt to what the previous model response was, rather than being limited by maintaining a fixed set of previous turns. In order to execute such dynamic evaluation, it often involves a human evaluator, or another LLM acting as the user. Compared to static context, side-by-side dynamic conversations can challenge the model in realistic interactive situations that reveal interactive behavior gaps in a much more natural way, but they're expensive, harder to reproduce, and difficult to debug. The downside is that when the models in comparison provide responses with large deviations from each other, one often has to choose one direction to construct the next user prompt, rendering essentially an interruption to the follow-up behavior in the other model. If the dynamic conversations continue for several turns, it's likely neither side would end up with an end-to-end coherent standalone conversation. The LM Arena's crowd-sourced platform is one example, allowing users to interact with two models side-by-side, providing the same prompt, compare the two models' responses, and vote for a preferred one. In an LLM development evaluation suite, side-by-side dynamic conversation evaluations can be used to assess interactive behaviors where status

context fails to capture. However, their complexity means they should complement, not replace, simpler evaluation types (Fig. 3.2).

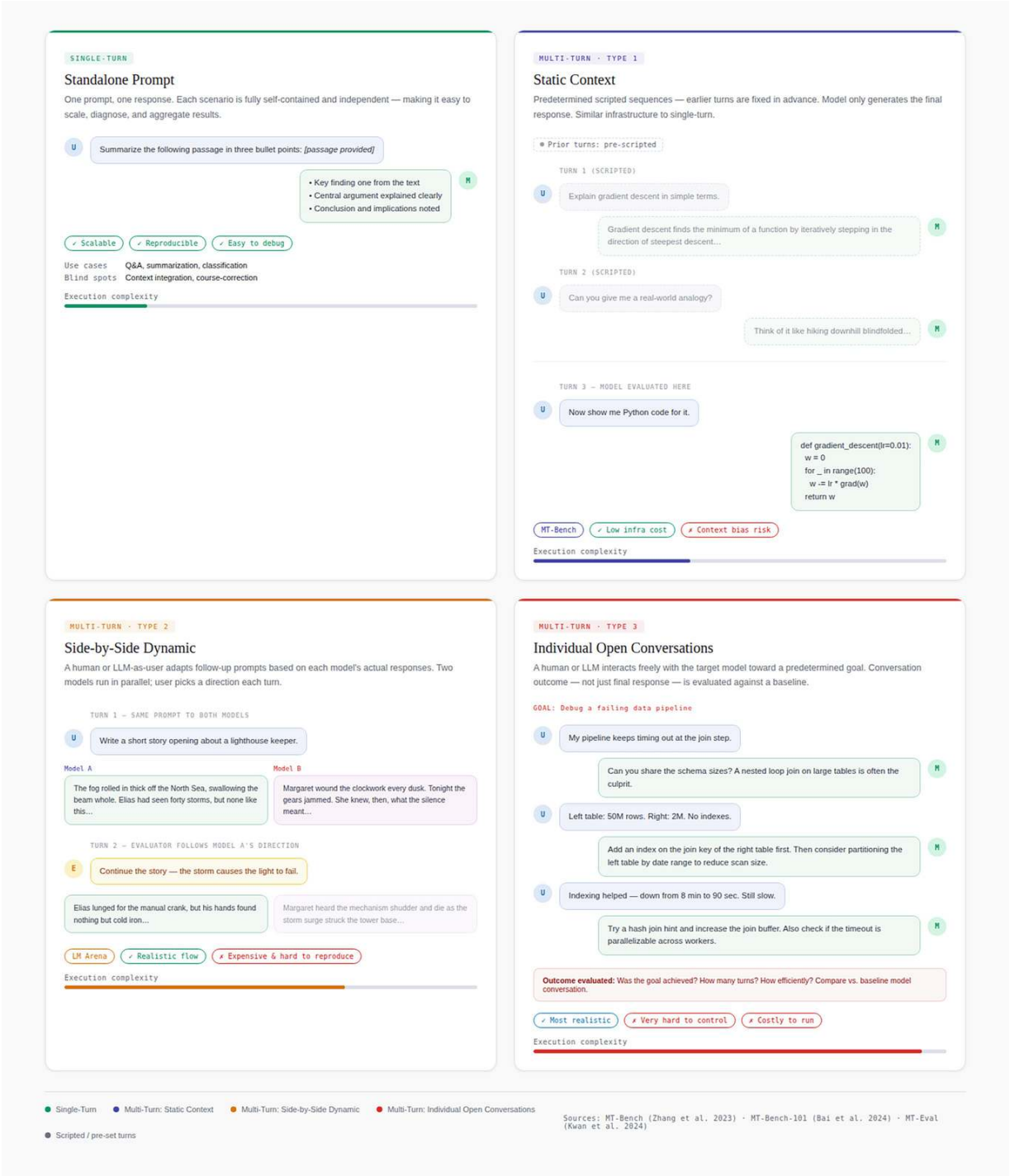

Fig. 3.2 Multi-turn evaluations via individual open conversations

Multi-turn evaluations via individual open conversations take the realism of the evaluation environment to the next level. They measure a target model's performance by having a human or LLM predict the next user turn interacting with the target model, mostly freely, with a predetermined goal in mind. They then compare the conversation's outcome, including whether the model's responses altogether achieved the user's goal, as well as how efficient and engaging the process was, against an independently executed conversation with another model to use as a baseline. The user-driven open conversations independently carried out using the target and baseline model were realistic but even harder to replicate, control bias and noise, and to diagnose.

3.1.3.5 Design an Overall Evaluation Strategy

The most effective evaluation strategies combine multiple types of tests, each serving a specific purpose.

Begin with a broad base of single-turn unit tests that verify atomic capabilities and can run quickly during development. Layer on integration tests that check realistic workflows. Use single-sided evaluations to enforce constraints and measure against absolute standards, while employing side-by-side comparisons to optimize for subtle quality improvements. Designate clear optimization targets and protect them with constraint evaluations that catch regressions. For most capabilities, start with single-turn tests and only introduce multi-turn complexity when it's necessary to evaluate truly conversational or stateful behavior. Save dynamic conversation evaluations for the capabilities that matter most and where static tests have proven insufficient.

One important principle to remember is that evaluation is not just about measurement; it's also about learning. The right evaluation set helps deepen the understanding of a model's capabilities, guides development priorities, catches problems before they reach users, and provides confidence that changes are genuine improvements rather than clever overfitting to the test sets. Building a thoughtful evaluation suite, with clear intent for what each component reveals about the

model's capabilities and behaviors, is key to a foundation for systematic progress in LLM development.

3.1.4 Requirements of a Good Evaluation Set

The previous two sections explored where evaluation scenarios can be sourced from (Sect. 3.1.2), as well as what types of evaluation sets they should form (Sect. 3.1.3), guided by how the evaluation sets are intended to be used. These two together provided a general direction on how to establish a custom evaluation set, a turning point in the four-phase maturing journey discussed in Sect. 3.1.1.

This section, built on the previous two, fills a critical gap in establishing not only an evaluation set, but a good one. What does "good" mean? How to improve from "good" to "better"? The dimensions of evaluation set quality we'll discuss are interconnected yet distinct, each addressing different aspects of what makes an evaluation meaningful and useful not merely as a measurement tool, but as a lens through which we can understand model capabilities, identify failure modes, and guide development priorities.

3.1.4.1 Validity

Validity refers to whether an evaluation set actually measures what it purports to measure. In the context of LLM evaluation, validity encompasses both construct validity and ecological validity. The former refers to whether the evaluation captures the theoretical construct of interest, while the latter represents whether performance on the evaluation predicts real-world performance. Both require a deep understanding of the phenomenon, model capability, or model behavior one aims to measure and are challenging to address.

Construct Validity requires that evaluation tasks genuinely test the capabilities they claim to assess. For instance, a reasoning benchmark should require actual reasoning rather than pattern matching or memorization. This concern has become particularly acute as models have grown larger and training data more extensive. Recent work (Zheng et al. 2024; Bean et al. 2025; Alaa et al. 2025) has revealed

issues in the operationalization of abstract phenomena, task selection, and statistical testing that undermined the validity of the benchmark claims.

The first and most essential step of ensuring construct validity is to understand and provide a precise and operational definition for the phenomenon one is set up to measure. This sets the direction and boundaries for all subsequent tasks of establishing a custom evaluation from task selection to template design. In order to articulate and build a shared knowledge of the phenomenon, positive examples can help clarify the phenomenon and guide sub-component identification, while negative examples can help shape what is out of scope, which further helps isolate the phenomenon to measure.

On the other side of the same coin, the second step is to control for confounding factors that may affect the evaluation outcome, to ensure we are not measuring a mixture of phenomena. When confounds are present, for example, a user's Internet speed, it becomes impossible to interpret what a model's evaluation outcome actually reflects For example, if a reasoning benchmark requires models to produce responses in strict JSON format, poor performance might indicate weak formatting skills rather than poor reasoning ability, fundamentally undermining the validity of any conclusions about reasoning capabilities. Without proper controls, benchmarks risk measuring a confounded combination of the target phenomenon and these extraneous factors, making it unclear whether improvements on the benchmark represent genuine progress on the intended capability or simply better handling of the measurement artifacts. This is why rigorous benchmarks should either eliminate confounding tasks entirely, measure them separately to adjust scores accordingly, or at minimum acknowledge their presence as a limitation. Otherwise, the benchmark fails to measure what it claims to measure, rendering comparisons between models unreliable and potentially misleading for both research progress and deployment decisions.

After identifying a clear picture of what to measure and what not to measure, construct validity requires a representative evaluation set.

Section 3.1.2 explored the various sources to form such a set, and the previous step of isolating the phenomenon to measure will help identify the right distribution that the evaluation set is supposed to represent.

For example, if an evaluation is designed to measure the average impact of a model change on the product's typical user base, the distribution to represent is the usage pattern and user needs of the unbiased live traffic. However, if an evaluation is designed to measure the average impact of a model change on where the previous model is lacking, the optimal distribution to construct the evaluation on is no longer always the unbiased live traffic. Because for a somewhat mature model, often the typical usage pattern is not where the model is lacking.

The metrics and statistical analysis on the outcome based on the evaluation set are equally important, if not more, to ensure construct validity.

In practice, it is almost impossible to rule out all confounding factors. As a result, it often takes several iterations to refine the methodology to source and select the right scenarios to construct a valid evaluation set. Two exercises could be quite helpful in this situation. Firstly, if there are existing public or custom benchmarks aiming to measure a similar phenomenon, similar outcomes should be observed comparing the same models. Strongly different results could reveal major confounding factors or mistakes in the evaluation setup or even data processing. Secondly, it's worthwhile to take the time to discuss the limitations and trade-offs each method presents in each step of the evaluation set design and process. Through such deliberate investigation, one can drastically reduce the chance of having major blind spots in the construct validity of the evaluation set design.

Lastly, a critical challenge for validity is the training-evaluation data contamination. As LLMs are trained on increasingly large web scrapes, popular benchmarks may appear in training data, either directly or in similar forms. This makes it difficult to distinguish genuine capability from memorization. Recent proposals include using dynamically

generated evaluations or private test sets, though these introduce their own trade-offs.

Ecological Validity can be considered as one special case of construct validity, in that it addresses whether benchmark performance translates to real-world utility, i.e., how representative one evaluation set is compared to real-world usage patterns. Many academic benchmarks involve carefully curated tasks with clear correct answers, while real-world applications often involve ambiguous queries, multi-turn interactions, and subjective quality judgments. Anthropic's evaluation work (Miller 2024) has emphasized the importance of evaluating models on realistic usage patterns.

3.1.4.2 Actionability

Actionability refers to whether evaluation results provide clear guidance for model improvement. An actionable evaluation set helps developers understand not just that a model is failing, but why it's failing and what might be done to address those failures.

Diagnostic Value is central to actionability. An ideal evaluation set breaks down performance across multiple dimensions, enabling developers to identify specific weaknesses. For instance, rather than reporting a single accuracy score, an actionable evaluation might separate performance by task type, topic category, or reasoning depth, revealing patterns that suggest concrete areas for improvement. Such failure mode analysis often requires the evaluation set to include rich metadata about each example on dimensions mentioned above. This metadata enables systematic investigation of failure modes rather than anecdotal observation of individual errors.

In order to build an actionable evaluation set, heuristic or model-based classifiers are often adopted to annotate each evaluation example with metadata such as model capabilities, topic or domain, complexity, difficulty, and safety policies. It is important to verify the accuracy of such classifiers before applying the results to automate failure mode analysis.

For example, TruthfulQA (Lin et al. 2022) contains 817 questions spanning 38 topic categories, such as health, law, and finance. Slicing the evaluation metrics by domain provides insights on model behaviors in various domains, which may lead to targeted strategies to improve the model in each relevant domain. Another example, SummEval (Fabbri et al. 2021), does this for summarization. It provides per-example human annotations across multiple quality dimensions, such as coherence, consistency, fluency, and relevance, so failure modes can be disaggregated by dimension rather than collapsed into a single score.

However, evaluation actionability is ultimately limited by the complexity of LLMs. When a model fails on a task, the root cause may involve interactions between training data, architecture, and optimization that are difficult to disentangle. This has led to calls for more mechanistic evaluation approaches that probe internal model representations.

3.1.4.3 Reliability

Reliability refers to the consistency and reproducibility of evaluation results. A reliable evaluation should yield similar results when repeated and should not be overly sensitive to minor variations due to random chance or any factors unrelated to model performance, such as rater pool assignment.

Consistency across runs is a must-have requirement. For evaluations involving any randomness, in-example ordering, sampling, model temperature, or positioning (in side-by-side evaluations), results should be stable across runs. This requires appropriate aggregation over multiple samples and reporting of uncertainty estimates.

There are several touchpoints in designing and executing an LLM evaluation where measurement uncertainty enters and reliability studies and improvements are needed. Rater reliability is the most well-studied (Sect. 3.4.2); reliability for template designs can also play a major role in ensuring overall evaluation reliability (Sect. 3.2); the

reliability required in evaluation set construction, however, is often overlooked.

Inter-Annotator Agreement (IAA), also known as Inter-Rater Reliability (IRR), is crucial for evaluations involving human judgment. For open-ended generation tasks, multiple annotators should agree on quality assessments. Low agreement suggests either unclear evaluation criteria or genuinely ambiguous cases, the former caused by noise in the rating process, while the latter in set construction, both of which limit reliability. Recent work has shown that human evaluation of LLM outputs can have surprisingly low agreement, particularly for subjective qualities like helpfulness or creativity. This strongly suggests the need to leverage analysis on IAA to identify the types of evaluation examples that are difficult to reach a robust rating. A simple next step is to remove such examples from the set. A more time-consuming but potentially bias-avoiding approach is to analyze the source of noise in the example. If, for example, the example is considered subjective because each rater has to make their own assumption about the user's true intent, providing complete context on the user's goals may improve the robustness of the ratings on this example.

A special and quite prevalent case of the above presents itself in that small changes in wording, formatting, or even whitespaces can substantially affect model outputs. A reliable evaluation should either control for this unwanted sensitivity through careful prompt engineering or explicitly measure it as part of the evaluation (Liang et al. 2022; Aali et al. 2025). The existence of "prompt engineering" as a discipline highlights how unreliable model behavior can be.

In practice, one has to consider other trade-offs such as computation and turnaround time before employing certain techniques to improve the robustness of evaluation results. For example, the self-consistency sampling (Wang et al. 2022) can improve model performance but at the cost of multiple forward passes. Evaluations must specify whether such techniques and their incurred costs are tolerated and ensure fair comparisons.

3.1.4.4 Efficiency

Efficiency addresses the practical costs of running evaluations, including that on computation and human effort. As models grow larger and more expensive to run, and as the number of public and custom benchmarks proliferates, efficiency becomes a critical consideration for sustainable evaluation practices.

Efficient evaluation sets achieve good coverage of model capabilities with minimal redundancy. Both human-written and LLM-generated evaluation sets are prone to have almost similar examples that challenge the model's capabilities in the same manner; analysis that focuses on identifying the critical subset that has the same predictive power on the model performance delta (Polo et al. 2024) becomes more and more crucial.

Trade-offs with the other evaluation quality dimensions are inevitable. More efficient evaluations may sacrifice validity in that fewer examples may not capture the full distribution of scenarios, actionability due to less detailed error analysis, or reliability because fewer samples increase variance. The art of evaluation design involves finding the right balance for the intended use case.

An emerging approach to efficiency is adaptive evaluation, where the evaluation adapts based on model performance. Easy examples that all models solve can be skipped, while challenging areas receive more attention. However, this introduces complications for standardization and comparison across models.

3.1.4.5 Sensitivity

Sensitivity refers to an evaluation's ability to discriminate between models of different capabilities, particularly at the high end of performance. As models improve, evaluation sets can become saturated, with multiple models achieving near-perfect scores, making it difficult to identify the best models or measure progress.

Dynamic Range is crucial for sensitivity. An evaluation should have sufficient headroom that even the best current models don't achieve perfect performance. This requires anticipating future progress when

designing evaluations, challenging given the rapid pace of LLM development. The "AI effect" describes how tasks that seem difficult often become easy once solved, making it hard to maintain sensitive evaluations.

Avoiding Saturation requires proactive evaluation design. Some strategies include using open-ended tasks with no upper bound on quality, incorporating adversarial examples specifically designed to challenge strong models, or using human-level performance as a moving target. However, each approach has limitations.

Kiela et al. (2021) introduced the concept of "dynabench," arguing for dynamic adversarial data collection where humans continually create examples that fool current models. This approach maintains sensitivity by co-evolving evaluations with model capabilities. The downside, however, is the high cost of maintenance and the loss of a stable benchmark providing longitudinal measurements.

Granularity of Measurement also affects sensitivity. Coarse metrics (like binary pass/fail) provide less signal than finer grained assessments. However, increasing granularity often requires more complex evaluation protocols and may reduce reliability if the finer distinctions are difficult to judge consistently.

3.1.4.6 Making Trade-Offs

These five dimensions interact in complex ways, and designing a good evaluation set requires navigating inevitable trade-offs. A perfectly valid evaluation might be impractical due to efficiency concerns. A highly sensitive benchmark might sacrifice reliability if it depends on subjective judgments of subtle quality differences.

As is the same as all other aspects of evaluation design, hitting the right balance depends on the evaluation's goals and purpose. Research benchmarks might prioritize sensitivity to measure incremental progress, while deployment evaluations might emphasize validity and reliability to make high-stakes decisions about model readiness. Actionability is particularly important during development but less critical for external comparisons. One direction of reaching a balance

produces frameworks like HELM, BIG-Bench (Srivastava et al. 2023), and EleutherAI's evaluation harness, which combine multiple evaluations with different strengths, providing a more complete picture of model capabilities than any single benchmark could offer.

The next step after defining the five quality dimensions of the evaluation set, is measurement and improvement. This is often referred to as meta-evaluation, studies that evaluate the evaluation set in each of the five dimensions. Understanding the quality of each benchmark improves the efficiency of the entire evaluation system by allocating the right amount of resources.

3.1.4.7 Summary

The five dimensions of validity, actionability, reliability, efficiency, and sensitivity provide a framework for understanding what makes an LLM evaluation set valuable. No evaluation perfectly satisfies all dimensions, but being explicit about these qualities helps evaluation designers make informed choices and helps evaluation consumers interpret results appropriately.

3.1.5 Beyond Set Creation: Maintenance and Upgrades

Creating a custom evaluation set is merely the beginning of a rigorous evaluation framework. As models train and improve, not only does the overall evaluation approach evolve (see Sect. 3.1.1), individual evaluation sets must also evolve to remain useful signals of genuine capability with relatively low noise.

This section explores the lifecycle of custom evaluation sets beyond creation, from their initial deployment through their maturation, expansion, and eventual retirement or transformation into regression benchmarks.

3.1.5.1 Early Signals of Custom Evaluation Set Quality and Ways to Improve

When deploying a custom evaluation set, there are many early signals that could be gathered to provide indications of the set's overall quality.

A poorly or not at all validated benchmark can mislead development priorities, waste computational resources, and propagate flawed conclusions throughout the development.

Bias Detection via Comparisons with Known Outcomes

A necessary but not sufficient condition for an informative custom evaluation set is that it can correctly distinguish models with drastically different capabilities. Thus, by checking if the custom evaluation set assigns the right comparison outcomes between models, either of quite different rankings on public leaderboards, or of very different model sizes (parameters), can provide a quick signal on whether the evaluation set roughly points in the right direction. Kaplan et al. and Hoffmann et al. established that loss scales predictably with compute. Evaluations should generally respect these trends, exhibiting sensible scaling properties with respect to model families, parameters, compute, and data. For example, for a series of models within the same family with increasing sizes:

- Plot evaluation score against model size (parameters) on log scale.
- Verify monotonic or near-monotonic improvement.
- Identify problematic subsets that may indicate evaluation mis-signals.

Bias Detection via Contamination Against Training Data

This is a crucial test to implement and, ideally, automate to run periodically. Including the exact copy or even a near copy of the evaluation example in the training set can cause the evaluation set to lose its representativeness and make the result meaningless.

Similarity scores in the embedding space are relatively easy to implement and cheap to scale between post-training datasets and the evaluation set, as both follow similar conversational formats. On the other hand, source- or article-level similarity in the pretraining datasets seems to work more efficiently.

Variance Reduction via Rating Reliability

Even among the same pool of raters, some evaluation examples tend to present more reliable ratings, measured by Inter-Rater Reliability (IRR) metrics, than other examples. It is often fruitful to examine the level of IRR at the evaluation subsets level. Some examples could be hard to reach a certain level of reliability due to specific evaluation examples:

- Requires domain expertise or knowledge that's lacking in general raters.
- Involves a significant amount of subjective judgments.
- Lacks sufficient context on user goals to fully define "what is better."

For examples in the first situation, it is often recommended to remove such evaluation examples or relocate them to a separate evaluation set with specially skilled rater assignments.

Examples in the second and third situations, however, should generally be removed as they are not the most effective in evaluating and differentiating models. Some examples in such categories can still be high-quality evaluation examples, but leveraging them well requires more nuanced and careful design of the rating process, such as diversifying and providing clarity on user goals before aggregating.

Variance Assessment Across Training Checkpoints

For evaluation sets to give good signals throughout training, it's important to evaluate at regular checkpoint intervals to characterize noise properties:

- Plot learning curves: Score vs. training step should show interpretable progression.
- Calculate temporal autocorrelation: High-frequency oscillations may indicate evaluation instability rather than genuine performance fluctuation.
- Identify regime changes: Sudden drops or plateaus warrant investigation on set validity and relevance.
- If your evaluation shows high variance while training loss smoothly decreases, the evaluation set may be capturing ephemeral behaviors or may have insufficient sample size.

Diversity in Not Only Topic Categories but also Difficulty and Complexity

A good distribution of topics is often strived for in evaluation set designs. It is equally important to verify that the evaluation set has an appropriate difficulty distribution. An evaluation where all models score near 100% (ceiling effect) or near 0% (floor effect) provides minimal discriminative power. Plotting score distributions across multiple models spanning different capability levels can provide a quick understanding of how diverse the evaluation set's difficulty levels are. A well-climbed evaluation set should exhibit gradual saturation, i.e., a distribution shift of difficulty over time.

3.1.5.2 Expansion to More Challenging Sets

Static evaluation sets eventually lose discriminative power as models improve. If set at a difficulty level that's too far advanced for the current development, the evaluation set is not very useful when all experiments are likely to return negative outcomes. This is often the case, as improvement is gradual and not overnight. Therefore, maintaining effective evaluation requires systematically matching task difficulty to the frontier of model capabilities. This progression should be principled rather than arbitrary, with clear hypotheses about what makes tasks more challenging.

Several dimensions can be used as a lever to increase difficulty. Task complexity can be scaled by requiring longer chains of reasoning, integrating information across multiple disparate sources, or involving more steps in multi-stage processes. For instance, a basic reading comprehension task might evolve into multi-document evidence synthesis, then into retrieval-augmented question-answering over large corpora.

Ambiguity and underspecification represent another dimension for growing difficulty. While initial evaluation sets often feature clear-cut cases with single correct answers, advanced evaluation can introduce legitimate ambiguity requiring models to recognize uncertainty, request clarification, or qualify their responses appropriately. This

mirrors real-world deployment conditions more faithfully than artificially simplified benchmarks, isolating one basic capability at a time.

Domain transfer and compositional generalization offer additional challenge dimensions. While initial evaluation might assess capabilities within familiar domains, advanced evaluation can test transfer to novel domains, combinations of previously learned skills in new environments, or applications requiring creative extrapolation beyond training distribution patterns. For example, retrieval and reasoning based on recent (Li & Goyal, 2025) and evolving (Nakshatri et al. 2025) knowledge is more challenging than that using well-established knowledge and thus needs evolved evaluation sets to test.

Ultimately, Chollet (2019) argues for evaluation focused on generalization efficiency and skill acquisition rather than skill itself, suggesting that evaluation should progressively test the ability to learn and adapt rather than simply measuring static knowledge.

3.1.5.3 Expansion to Multilingual Evaluation Sets

As language models expand to serve global audiences, evaluation frameworks must extend beyond English to assess capabilities across diverse languages and cultural contexts. Developing multilingual evaluation sets presents unique challenges in translation, cultural adaptation, and ensuring comparable difficulty across languages.

The most straightforward approach involves direct translation of existing English evaluation sets. However, naive translation often fails to preserve task difficulty or semantic content. Idiomatic expressions may have no direct equivalents. Cultural references may become obscure or meaningless. Syntactic structures available in English may not exist in target languages, fundamentally changing task requirements.

Hu et al. (2020) describe a comprehensive multilingual evaluation framework covering 40 languages. Their work emphasizes the importance of including typologically diverse languages rather than only high-resource languages similar to English. Evaluation should span

different writing systems, grammatical structures, and morphological complexity to test genuine multilingual capability rather than transfer learning from English.

Cultural adaptation goes beyond linguistic translation. Evaluation examples involving historical events, geographic knowledge, social norms, or commonsense reasoning must be adapted to be appropriate and fair for speakers of different languages. An evaluation question about American football might be replaced with one about cricket for Hindi evaluation sets, or football (soccer) for Spanish sets, maintaining comparable reasoning requirements while ensuring cultural relevance.

Creating original multilingual evaluation content, rather than translating from English, provides the highest quality assessment. Native speakers can develop evaluation examples that naturally reflect how tasks arise in their linguistic and cultural contexts. This approach avoids the subtle artifacts of translation and ensures that evaluation measures genuine capability rather than translation robustness. The downside is obvious: high cost and slow development.

Parallel evaluation sets, where the same underlying task is presented in multiple languages with careful calibration of difficulty, enable cross-lingual capability comparison. If a model performs substantially better in English than in other languages for the same task type, this indicates language-specific weaknesses rather than general capability limitations. Ponti et al. (2020) demonstrated this parallel evaluation approach.

3.1.5.4 Contamination and Overfitting Prevention

The reliability of evaluation signals depends on the integrity of evaluation sets, which requires clear separation from training processes. This separation can be enforced through technical controls or organizational practices, but best safeguarded by automated monitoring.

Training-Eval contamination occurs when evaluation examples appear in training data, either explicitly or through near-duplicates. Brown et al. (2020) described contamination analysis procedures

involving n-gram overlap detection between training and evaluation sets. They found that even for seemingly isolated benchmarks, substantial overlap can occur through web scraping and data aggregation processes.

More generally, the overfitting problem manifests in many forms. Models may learn superficial patterns specific to the evaluation format rather than the underlying capabilities being tested. Training procedures may inadvertently optimize for benchmark performance through hyperparameter selection or architectural choices validated primarily against specific evaluation sets. Most concerning, as evaluation sets circulate within the research community, they risk becoming implicit training objectives rather than independent measures of capability.

The work by Dodge et al. (2021) highlighted how evaluation data can leak into training corpora through web scraping, creating contamination that undermines benchmark validity. Similarly, Recht et al. (2019) demonstrated that model performance often drops substantially on newly collected test sets that match the original distribution, suggesting that models learn dataset-specific artifacts rather than general capabilities.

Preventing contamination requires proactive measures throughout the training pipeline. Training data should be scanned for near-duplicates of evaluation examples, removing matches above a conservative similarity threshold or disqualifying the evaluation example. For datasets derived from web sources, URLs of evaluation example sources should be explicitly blocked during training data collection.

Beyond preventing direct contamination, it's important to avoid training-evaluation sharing the exact (narrow) distribution. For example, if evaluation sets are collected through specific prompting templates or interface designs; training data should not disproportionately feature similar templates. This, ultimately, goes back in a full circle to the importance of evaluation set construct validity.

Organizational practices provide additional safeguards against overfitting. Developing separate evaluation sets for development and final candidate selection, but drawn from the same distribution, is an example of a great practice. The rationale is similar to separate validation and test datasets in traditional machine learning tasks. The evaluation set used for LLM development is used to tune hyperparameters, compare different architectures, and make decisions about which model variant to keep. Because these choices are based on model performance on such development evaluation sets, the model indirectly "learns" from the development set through this iterative selection process, which means validation accuracy becomes an increasingly optimistic estimate of real-world performance. The final evaluation set solves this problem by remaining completely untouched until the very end, after all development decisions are finalized. This provides an unbiased estimate of how the model will actually perform on truly unseen data in production. Without this separation, the risk of overfitting to the evaluation set through the model development process itself, increases and can lead to deploying a model that looks great on paper but fails to generalize to new data.

3.1.5.5 Retirement

A hillclimb evaluation set serves as an active development target during model training. Teams iterate on model architecture, training procedures, and data mixtures while monitoring performance on these sets. However, once performance saturates or when there's suspicion of overfitting, the set should graduate to become a regression evaluation set.

Regression sets serve a different purpose: they establish a baseline of capabilities that must be maintained as models evolve. Rather than driving improvement, they detect capability regressions when new training approaches or architectural changes inadvertently harm previously acquired skills. This retirement to regression process acknowledges that an evaluation set's utility changes over its lifecycle.

The timing for retirement can be triggered by several types of conditions. If model performance exceeds 95% accuracy on a previously discriminative task, the set likely no longer provides a useful training signal. If successive model iterations show diminishing returns despite continued optimization efforts, saturation has occurred. If analysis reveals that errors are primarily due to annotation noise rather than genuine model limitations, the set has reached its ceiling. As a last example, if training dynamics show signs of memorization, such as much higher performance on the evaluation set than on held-out samples from the same distribution, the set should be retired from active use.

Once graduated to regression status, these sets should be frozen. No additional optimization against them should occur, and they should be evaluated only periodically to ensure capabilities haven't degraded. Documentation should clearly mark them as regression benchmarks to prevent teams from inadvertently optimizing against them.

As more and more active hillclimb sets retire to the regression set, computational costs for regression detection can become prohibitive, even if in most development environments, regression evaluations are run much less frequently than hillclimb evaluation. Downsampling provides a pragmatic solution.

Stratified sampling maintains representation across important dimensions of the evaluation space. If an evaluation set covers multiple task types, difficulty levels, or domain categories, downsampled versions should preserve these proportions. For a set with 40% reasoning tasks, 30% factual knowledge tasks, and 30% instruction-following tasks, a 1000-example downsample from a 10,000-example set should maintain these ratios.

Difficulty-weighted sampling can prioritize challenging examples that better discriminate between model capabilities. If human evaluation or model performance analysis has identified certain examples as particularly diagnostic of genuine understanding versus surface pattern matching, these examples should be overrepresented in downsampled versions. This approach ensures that evaluation

efficiency increases without sacrificing the ability to detect meaningful differences between systems.

When downsampling, it's essential to validate that the reduced set maintains similar discriminative properties to the full set. This can be verified by evaluating multiple models on both full and downsampled versions and confirming that relative rankings remain stable. If a downsampled set shows substantially different performance characteristics or changes the relative ordering of systems, it should be revised or the full set should continue to be used.

3.1.5.6 Housekeeping: Versioning and Documentation

As evaluation sets evolve through these various processes, rigorous versioning and documentation become essential. Each version should be clearly identified with unique version numbers following semantic versioning conventions: major versions for fundamental changes in task definitions or evaluation criteria, minor versions for additions or modifications that preserve backward compatibility, patch versions for corrections of errors or ambiguities.

Documentation should record the complete provenance of each evaluation set version: creation date, source of examples, annotation procedures, known limitations, intended use cases, and inappropriate applications. Changes between versions should be exhaustively documented with clear rationales for modifications.

Gebru et al. (2021) introduced the concept of "Datasheets for Datasets," providing a framework for documenting dataset characteristics, collection procedures, and recommended uses. This framework applies directly to evaluation sets and should be standard practice, especially as new evaluation sets are continually created and updated to expand capabilities as well as their depth.

3.1.5.7 Conclusion

Evaluation sets are living artifacts that must evolve alongside the models they assess. Contamination prevention, systematic difficulty increases, multilingual expansion, and the graduation from hillclimb to

regression sets represent essential practices for maintaining evaluation integrity throughout a model's development lifecycle. Organizations that treat evaluation as a static process risk optimizing toward obsolete benchmarks while genuine capabilities stagnate. Those that invest in evaluation evolution ensure their models continue improving on capabilities that matter for real-world deployment.

3.1.6 Beyond Individual Evaluation Sets

Evaluating LLMs is a multi-faceted challenge that cannot be adequately addressed through a single benchmark or evaluation methodology. As these models become increasingly capable and are deployed across diverse applications, the need for comprehensive, multi-dimensional assessment frameworks has become critical. This section explores the principles and practices of using multiple evaluation sets in concert to measure the collective capabilities of LLMs, addressing the complexities of interpretation, reconciliation of conflicting results, and the nuanced differences between evaluation paradigms.

3.1.6.1 The Necessity of Multiple Evaluation Perspectives

The limitations of relying on individual benchmarks have been well documented in the literature. Chollet (2019) argues that narrow task-specific benchmarks often fail to capture general intelligence or the ability to generalize to novel situations. Similarly, models are shown to achieve high scores on certain benchmarks through memorization or exploitation of dataset artifacts rather than genuine understanding (Gururangan et al. 2018; McCoy et al. 2019), rendering capabilities measured by a single benchmark thin evidence.

Multiple evaluation sets are necessary because:

- Task diversity: Different benchmarks emphasize different cognitive capabilities, from mathematical reasoning to commonsense understanding.
- Domain coverage: No single benchmark can comprehensively cover all domains where LLMs are deployed.

- Evaluation methodology variance: Benchmarks employ different evaluation paradigms (multiple-choice, generation, human evaluation) that each have distinct strengths and limitations.
- Robustness verification: Multiple evaluations help identify whether model capabilities are robust or brittle.

Therefore, evaluation sets can be viewed as complementary instruments in a diagnostic toolkit. Just as a physician uses multiple tests to diagnose a patient, researchers and practitioners should use multiple benchmarks to understand model capabilities. For example, MMLU (Hendrycks et al. 2021) assesses broad knowledge across 57 subjects using multiple-choice questions, while HumanEval (Chen et al. 2021) evaluates code generation through functional correctness. The evaluation sets may be combined into one, but the grading process requires a different set up, for example, execution verification is often needed for coding tasks, as well as different metrics aggregation approaches.

3.1.6.2 Identifying Convergent Validity

When multiple evaluation sets produce consistent results about a model's capabilities in a domain, this provides convergent validity, evidence that the observed capability is robust and not an artifact of a particular benchmark design or measurement noise. For example, if a model demonstrates strong performance on both MATH and GSM8K for mathematical reasoning, and also performs well on quantitative reasoning questions in MMLU, this triangulation provides stronger evidence of mathematical capability than any single benchmark alone.

Group Evaluations by Capability Domain

It's practically convenient and analytically efficient to organize benchmarks into groups that target similar capabilities, for example, reasoning, knowledge, coding, or instruction following.

A model that performs consistently well across benchmarks within a domain of capabilities demonstrates robust capabilities correspondingly. Conversely, a model that excels at one of the evals but

fails at others within the same domain reveals important limitations in generalization.

It's often computationally expensive to evaluate the candidates on each model development path with all the benchmarks within various domains of capabilities. So teams often pick a smaller subset to guide the model improvement phase. A more comprehensive set of benchmarks is then used sparingly at critical points to select top candidates for continued development or launch in products.

Analyze Correlation Patterns

Integrating a correlation analysis into the evaluation pipeline is essential for achieving a more rigorous and robust understanding of model intelligence, as well as the quality of the adopted benchmarks. For example, Qian et al. (2026) demonstrated that traditional leaderboards often suffer from ranking inconsistency, where slight variations in benchmark selection lead to wildly different model hierarchies. Correlation analysis helps stabilize these rankings by identifying which tests provide the most reliable signals of quality.

Zhang et al. (2025), with a slightly different angle, suggests that many current benchmarks are over 90% redundant. Without a correlation study, model dev teams waste significant computational resources and research bandwidth testing the same latent abilities across multiple redundant datasets.

Lastly, Zhou et al. (2025) demonstrated that a lack of correlation between benchmarks that claim to measure the same domain of capabilities often reveals fundamental flaws and biases in evaluation design.

By quantifying these relationships, researchers can trim redundant tests, ensure concurrent validity, and develop more streamlined, less noisy evaluation suites.

Integrating Multiple Benchmarks

When combining multiple LLM benchmarks into a unified score, researchers typically employ weighted aggregation methods that reflect the relative importance of different capabilities.

The most straightforward approach is simple averaging, which treats all benchmarks equally, though this can be problematic when benchmarks measure overlapping skills or when certain capabilities are overrepresented.

Domain-weighted averaging assigns weights based on the importance of different capability domains (reasoning, knowledge, coding, etc.), which requires either expert judgment or empirical validation of which skills matter most for target applications.

More sophisticated approaches include principal component analysis (PCA) to identify underlying factors and reduce redundancy between correlated benchmarks, or item response theory (IRT) methods that account for benchmark difficulty and discriminative power. Some researchers advocate for Pareto frontier analysis rather than scalar scores, plotting models across multiple dimensions to identify trade-offs rather than collapsing everything to a single number.

A key challenge is that weighting choices embed value judgments about what "better" means. For example, weights optimized for academic research tasks may differ from those prioritizing real-world utility. Liang et al. (2022) introduced the "Holistic Evaluation of Language Models (HELM)" framework; Yang et al. (2023) discussed methodological issues in combining scores. Artificial Analysis Intelligence Index v4.0 is a single index representing the level of intelligence, leveraging 10 benchmarks (GDPval-AA, τ^2-Bench Telecom, Terminal-Bench Hard, SciCode, AA-LCR, AA-Omniscience, IFBench, Humanity's Last Exam, GPQA Diamond, CritPt).

3.1.6.3 Understanding Divergent Evaluation Results

Sources of Evaluation Disagreement

Divergent results across evaluations are common and can be highly informative. Understanding why evaluations disagree is often as valuable as the evaluations themselves.

1. Benchmark Design Differences:

   (a) Question format: Multiple-choice questions may advantage

models differently than open-ended generation tasks.

(b) Answer length: Some benchmarks require short answers, others lengthy explanations.

(c) Prompt sensitivity: Variation in prompt formatting can significantly affect performance.

(d) Difficulty calibration: Benchmarks may sample different regions of the difficulty spectrum.

2. Evaluation Methodology Variance

(a) Exact match vs. semantic similarity: String matching may penalize correct answers with formatting differences.

(b) Partial credit: Some benchmarks award partial credit for partially correct answers.

(c) Human vs. automatic evaluation: LLM-as-a-judge and human raters may disagree on subjective qualities.

(d) Reference-based vs. reference-free: Some metrics require gold standard answers, others assess quality intrinsically

3. Domain Specificity and Coverage

(a) Training data distribution: Overrepresentation or underrepresentation of certain domains in pretraining.

(b) Fine-tuning objectives: Models optimized for specific domains may underperform in others.

(c) Knowledge gaps: Missing or outdated information in particular subject areas.

4.

Task Formulation and Cognitive Demands

(a) Recall vs. recognition: Generating an answer from memory vs. selecting from options.

(b) Compositional reasoning: Tasks requiring multiple reasoning steps vs. single-step inference.

(c) Context length: Long-context understanding vs. short-prompt tasks.

Diagnostic Framework for Analyzing Disagreement

When encountering divergent results, a systematic diagnostic approach may reveal differences in evaluation designs, which in turn provide valuable insights into model's behaviors, weaknesses, and strengths.

Step 1: Characterize the disagreement

For example, quantifying the magnitude of performance difference, identifying whether disagreement is systematic or task-specific, and determining if the disagreement involves relative rankings or absolute performance can also shed light on where the source of disagreement is.

Step 2: Analyze benchmark characteristics

Before relying on a certain benchmark to guide model development, it is important to understand the evaluation methodologies behind the benchmark design, such as task formats and difficulty distributions, in order to pinpoint whether the divergent evaluation outcome is due to benchmarks measuring different aspects, or, for example, the measurement itself is unreliably executed. This step may also reveal how relevant or representative the adopted benchmarks are to the evaluation task at hand.

Step 3: Conduct disagreement analysis

Sampling examples where model performance differs across benchmarks and then categorizing error types and patterns can lead

to identifying whether errors reflect genuine capability gaps or evaluation artifacts.

Step 4: Perform controlled experiments to understand causality

If in rare and lucky situations, experiments on measuring the test model on variants of benchmark tasks result in successfully isolating factors that cause the evaluation outcome divergence, one could further use A/B testing to confirm causality.

Toy Example: MMLU vs. Real-World Domain Benchmarks

Let's say a model achieved 85% accuracy on MMLU's medical questions but only 60% on a specialized medical licensing exam. This divergence could stem from:

- MMLU using simpler multiple-choice questions vs. complex case-based reasoning in licensing exams
- Different coverage of medical specialties
- Variation in question ambiguity and clinical detail
- Different standards for what constitutes a correct answer

This type of analysis may reveal that while the model has strong general medical knowledge, it may lack the detailed, applied clinical reasoning required for professional practice.

3.1.6.4 Decision-Making with Conflicting Results

When different evaluation sets' results conflict, researchers and practitioners must first understand the source of the disagreement as discussed above, after which informed decisions often still need to be made about model selection, deployment, and further development, if the disagreement is caused by legitimate differences in the design and measurement. We now discuss a few strategies to make decision-making in such situations more formalized.

Strategy 1: Clearly Designate Your Evaluation Sets to Climb vs Regression Sets

The benefit of clearly designating evaluation sets to climb vs regression lies in the clarity it brings, because purpose determines usage.

Especially the sets designated as regression sets now can follow a simple strategy of how to be used as part of a large evaluation suite: threshold-based decision rules.

Regression sets only become visible in the decision-making process, if the model's performance dips below the thresholds that came along with the sets. For example,

- Safety thresholds: Minimum scores on truthfulness and safety benchmarks.
- Core capability thresholds: Required performance on benchmarks aligned with primary use cases, especially those already achieved in previous versions of the model.
- Robustness requirements: Acceptable performance across diverse evaluation conditions.

This approach ensures models meet baseline requirements across all important dimensions rather than trading off one capability for another.

Strategy 2: Decompose Composite Scores to Ensure Fair Baselining

Rather than comparing overall benchmark scores and getting an overall disagreement, decompose the benchmarks into sub-categories and only compare results within the same sub-categories. A model might excel at MMLU's STEM subjects but underperform on humanities, revealing more nuanced capabilities than the overall score suggests.

The results of the decomposing studies can lead to conducting targeted follow-up evaluations. Design custom evaluations that specifically probe the areas where benchmarks disagree.

Strategy 3: Design Ensemble Metrics

Use aggregated metrics across multiple benchmarks to simplify decision-making. The most common approach is via weighted average of various benchmarks. Researchers often start with a simple average mean and then make adjustments using feedback gained in ensemble metrics usage. Other ways to aggregate benchmarks include harmonic mean, which emphasizes minimum performance, as well as analysis to

identify models that are non-dominated across multiple objectives to help assign weights.

3.1.6.5 Conclusion

Evaluating large language models through multiple evaluation sets is not merely a best practice; it is a necessity for understanding the multi-faceted nature of model capabilities. This chapter has explored how different evaluation paradigms complement each other, how to diagnose and interpret divergent results, and how to make informed decisions when evaluations conflict.

As LLMs continue to evolve in capability and are deployed in increasingly diverse applications, the sophistication of our evaluation methodologies must keep pace. Multi-evaluation frameworks provide the comprehensive perspective necessary to understand these complex systems and deploy them responsibly and effectively.

3.2 Templates: Requirements for Designing Effective Evaluations

3.2.1 Introduction: The Core Challenge of Traditional Evaluation Techniques

In the world of LLMs being apprised across multiple industries and domains, including creative work, education, coding, and other specialized professional work, the classical way of doing evaluations often stops at a simple (and quite frequently binary) score or a vague statement of preference. Such feedback provided in the form of a verdict gives a summary judgment and has its merits, as we will show later in the chapter, but it completely fails one of the core purposes of assessment: to facilitate growth. Enabling a shift away from subjective, non-diagnostic critiques toward actionable, relevant, reliable, and sensitive feedback is paramount for successful applications as a better assessment provides a concrete roadmap, a diagnosis that outlines the steps needed to bridge the gap between where the work is and where it needs to be. We will be arguing that by embedding this kind of specific,

forward-looking guidance evaluation truly becomes a powerful engine for improvement.

3.2.2 The Failure of Traditional Metrics

The problem starts with outdated tools. When Large Language Models first emerged, they were primarily seen as text generators (e.g., auto-completion was the most common application of language models), and researchers naturally leaned on established, text-based metrics. For a long time, researchers relied on simple, automated metrics like BLEU (Papineni et al. 2002) and ROUGE (Lin 2004) to check text quality. These metrics basically just count word overlap and check how many words match between the model's output and a "perfect" reference answer. This approach was fine for simpler tasks, but it's totally inadequate for today's sophisticated models. Think about it: if an LLM is asked to generate an original poem or draft a complex legal brief, those word-counting metrics fail completely because they can't capture the deep semantic meaning or factual correctness (Fig. 3.3).

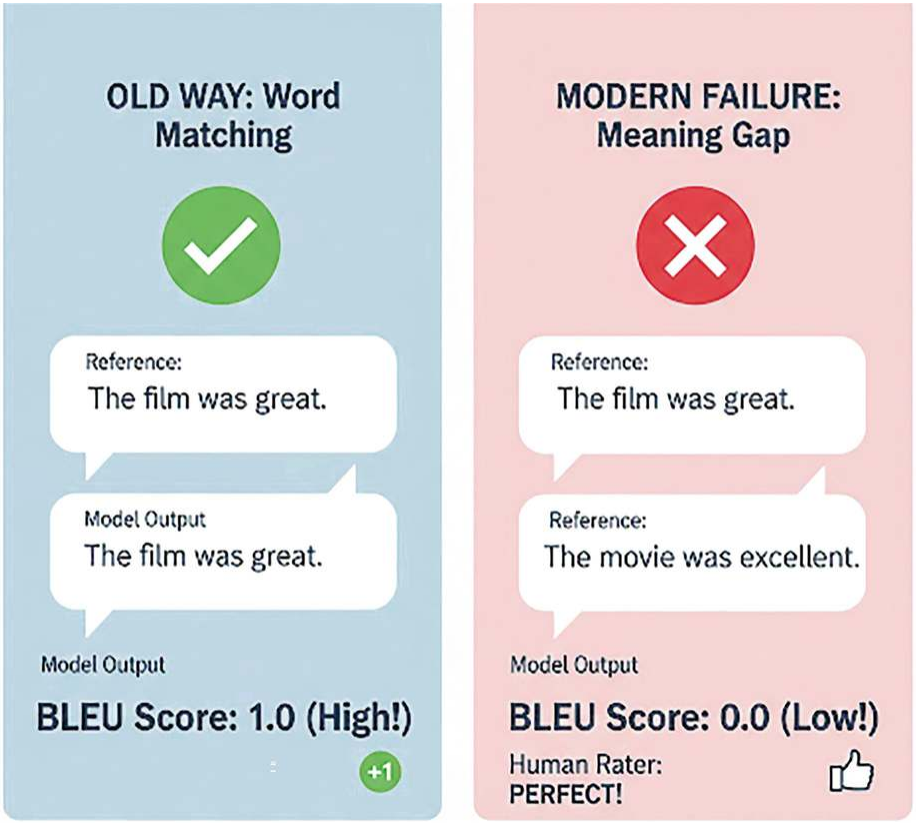

Fig. 3.3 The semantic gap in traditional metrics

Furthermore, LLMs don't just generate text anymore: they are increasingly multimodal, creating everything from images and code to audio, so text-based metrics completely miss the mark. Using a text-overlap score to judge a Python script or a medical image is a fundamental mismatch of tool to task. The disconnect is clear when we look at specific outputs: traditional scores cannot hear the "robotic" distortion in a synthesized audio clip or see the "hallucinated" extra finger in an AI-generated photo. Because these metrics lack any concept of visual or auditory fidelity, they remain blind to the errors that matter most to a human user. Attempting to measure the quality of a video with a text-based ruler is a category error; the units of measurement simply do not translate to the multi-domain outputs we now expect from advanced AI (Fig. 3.4).

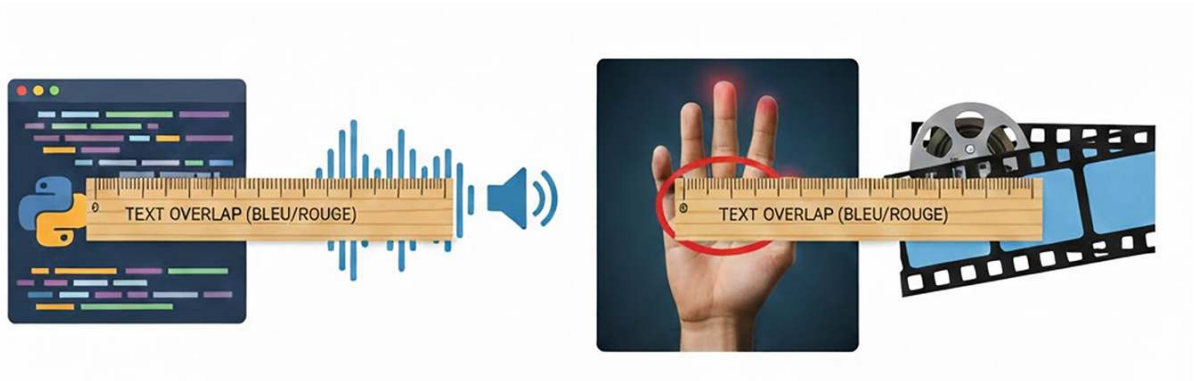

Fig. 3.4 The dimensional metric mismatch

One can argue that different modality can use different metrics and indeed literature references metrics specific to image generation and even some multi-modality quality metrics, such as the Learned Perceptual Image Patch Similarity (LPIPS) or Multi-scale Structural Similarity Index Measure (SSIM) metrics (HEIM Benchmark 2023). These metrics are quite specific and most likely not known for professionals outside of a specific area of development. Thus all these metrics are far too narrow and don't capture complexity and depth for the open-ended, multi-domain outputs we now expect from advanced

AI. Additional complexity is coming from the conversational and thus multi-turn nature of LLMs: as content of the conversation gets longer and more complex, so does complexity of evaluations. Taking all these considerations into account it becomes evident that metrics like BLEU/ROUGE are entirely useless, especially when evaluating the visual quality of an image or a video.

Even as we move past these crude techniques and utilize better, domain-specific measures, we encounter a second, equally critical failure point: preference for a single score that encompasses the overall result. This is a very natural tendency and preference that is true not only for the fields of LLMs but almost in any area of knowledge where decisions must be made and communicated. Think of analogy with Economics: we know that no country's economy can be distilled to the GDP value, but still this is one of the most common metrics that is being used to rank countries' economies. The same is true for the evaluation of LLMs: the tendency to reduce a response's quality to a single rating is expected and inevitable; still, it leads to the loss of critical information.

However, the desire for a "one metric to rule them all" is not inherently flawed; it is warranted for different business purposes. If the objective is to make a final, binary decision, such as whether a model is safe enough to deploy, suitable to replace a previous version, or ready to launch in a production system, an ultimate verdict is required. This final verdict takes all the diagnostic signals and maps them into a simple, actionable decision point. The single-dimensional score provides the necessary confidence and clarity for high-stakes decisions, enabling teams to determine model readiness and alignment with specified requirements. Therefore, while a single verdict serves its purpose for deployment and product management teams, its simplicity means it provides negligible utility to the engineering teams focused on continuous model improvement.

3.2.3 The Goal of Comparison: LM Arenas

As we have discussed, diagnostic feedback is paramount for engineers, but another critical objective that frequently drives evaluation is

comparison. This objective involves purely comparing two or more models against each other, essentially determining which one is generally perceived as "better." This comparative goal is the entire basis of the LMSYS ChatBot Arena (n.d.) (https://lmarena.ai/), one of the most visible and influential competitive evaluation platforms for advanced LLMs. This approach is also called *side-by-side or pairwise* score, as we compare one side vs. the other, while the other approach discussed before is called a *single-sided or pointwise* metric. These are deeply discussed in Sect. 3.1.3.2.

LMSYS was launched in 2023 by students and faculty from UC Berkeley, in collaboration with other institutions. The organization started as a multi-university research effort and was formally established as a non-profit corporation in September 2024. The goal of this project was to provide an open platform to address the fundamental difficulty of assessing LLM quality using traditional, static benchmarks. The platform uses a crowd-sourced methodology to capture real-world human preferences via anonymous, randomized head-to-head battles. Anyone can join the evaluation efforts and we encourage readers to contribute to the project and to experience the LM arena for themselves (Fig. 3.5).

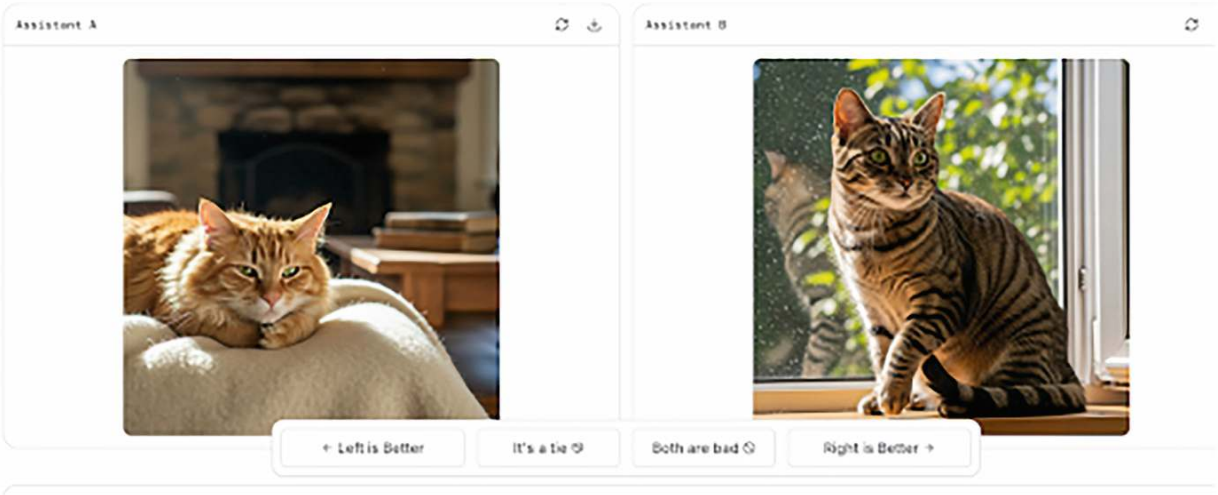

Fig. 3.5 Screenshot from LMSYS LM arena for the prompt: "generate an photo of a cat"

LMSYS LM arena directly measures how well models align with diverse user intent “in the wild,” an aspect that standardized tests often miss. Due to its unique value, openness, and scale, amassing hundreds of thousands of user votes and generating massive public datasets of real conversations (like LMSYS-Chat-1M), the Arena quickly became one of the most referenced LLM leaderboards globally. Its continuous tracking influences both commercial and open-source LLM development, providing a critical, up-to-date external validation of model capabilities and showcasing the narrowing performance gap between proprietary and open models (Fig. 3.6).

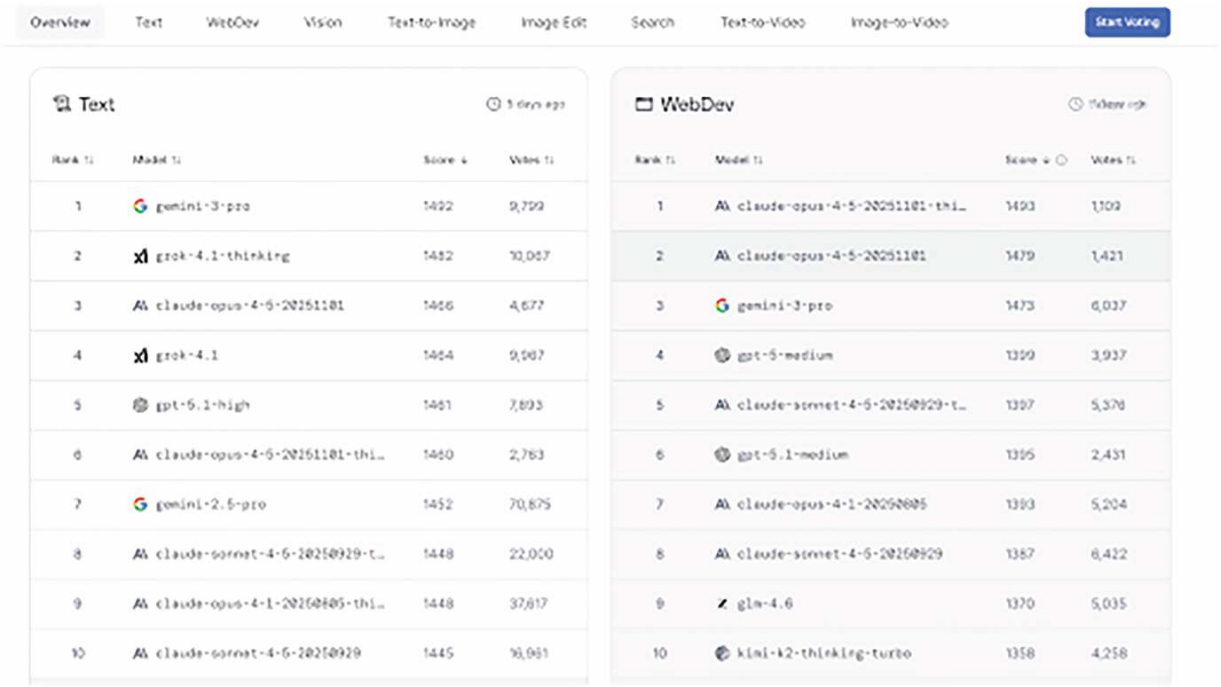

Fig. 3.6 Screenshot from LMSYS LM arena (https://lmarena.ai/leaderboard) as of Dec 1, 2026

The Arena’s primary output is its quantitative leaderboard, originally calculated using the Elo rating system (Elo 1978), a methodology borrowed directly from competitive games like chess. In this system, the Elo score quantifies the relative skill level of each model by tracking win/loss results against the expected outcome based on current rankings. The rating update for a model (A) after a battle is determined by the formula:

$$R'_{\mathrm{A}} = R_{\mathrm{A}} + K \cdot (S_{\mathrm{A}} - E_{\mathrm{A}})$$

where R'_{A} is the new rating, R_{A} is the current rating, K is the K-factor (a constant determining maximum adjustment), and S_{A} is the actual score (1 for a win, 0.5 for a tie, 0 for a loss). The expected score (E_{A}) is calculated based on the difference in the ratings of model A (R_{A}) and model B (R_{B}):

$$E_{\mathrm{A}} = \frac{1}{1+10^{(R_{\mathrm{B}}-R_{\mathrm{A}})/400}}$$

LMSYS applies this by treating each human preference vote (A wins, B wins, or tie) as a "battle" outcome. For better statistical precision and stability, especially since most models are static and their performance isn't expected to shift dramatically, the Arena later transitioned from the online Elo rating system to the Bradley-Terry (BT) model. The key distinction is that while the Elo system updates ratings sequentially based on individual game results, the BT model is an offline system that employs Maximum Likelihood Estimation (MLE) to analyze the entire history of all pairwise comparisons simultaneously. This provides a more stable and statistically robust rating coefficient for static entities like LLMs, which LMSYS now calls the "Arena Score." The probability that a model A wins against model B in the BT framework is calculated using the scores (r_{A} and r_{B}) derived from the MLE process:

$$P(\mathrm{A} \quad \mathrm{wins} \quad \mathrm{over} \quad \mathrm{B}) = \frac{\exp(r_{\mathrm{A}})}{\exp(r_{\mathrm{A}})+\exp(r_{\mathrm{B}})}$$

This score allows researchers to accurately calculate the predicted win rate of any one model over another.

GameArena is approaching competition among models in a different way, via competing head-to-head in strategic games, such as "Four in a Row," "Chess," and others (Fig. 3.7).

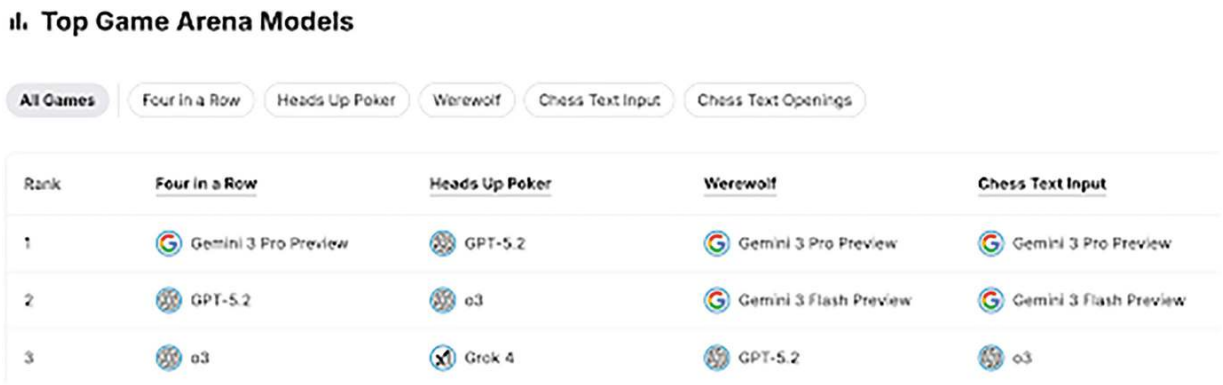

Fig. 3.7 Screenshot from GameArena (https://www.kaggle.com/game-arena) as of Feb 24, 2026

By the nature of the game's environment GameArena is focused primarily on models' ability to solve problems that require complex reasoning. One benefit of such an approach to evaluation is in resilience to saturation as the complexity of games scales as models become stronger.

In the Arena, models compete anonymously in randomized, head-to-head battles. The Arena's all-vs-all tournament structure neatly solves the typical problem of finding a perfect baseline model for comparison. However, outside of such competitive arenas, especially when a company is conducting A/B testing or internal model development, selecting the right baseline is a major technical challenge that requires a careful, strategic approach. Developers must thoughtfully decide which established model their new candidate should be compared against to ensure the resulting data is meaningful for product improvement. While highly efficient for determining a perceived leaderboard rank, this comparative method is fundamentally holistic and preference-based, reinforcing the challenge of obtaining diagnostic feedback on why one model was preferred over the other.

While being highly influential and respected in the LLM developers community, the LMSYS arena suffers from limitations, which are not limited to the lack of diagnostic feedback but also include potential biases from LMSYS users' prompts and preferences not necessarily representative of "real" users. Thus, over-indexing on LMSYS results

might skew evaluations toward a quite narrow group of power-users and their specific needs and preferences.

3.2.4 The Goal of Diagnostic: The Template Solution

The ultimate trade-off of comparative systems like the ChatBot Arena is clear: while they establish a consensus rank of "perceived quality," they do not help in understanding why a model failed or succeeded, thus failing the engineer's core need for diagnostics. To achieve the required level of detail, to move from a simple preference to an actionable diagnosis, we must return to a well-established principle: breaking down subjective complexity into objective, measurable components. This concept forms the basis of the evaluation template or rubric, the systematic, structured approach that allows us to capture the crucial nuance lost in single-score assessments.

The challenge of evaluating complex outputs is not new as researchers have long understood that to measure something subjective or complex accurately, you must break it down into objective, measurable components (Welty et al. 2019). This structured approach has roots in fields like education and human-computer interaction, where researchers developed scoring tools to evaluate everything from student essays to text, images, and videos. The key technique here is called deconstruction: converting a broad, holistic concept (like "Coherence") into clear, individual conceptual components (Jescovitch et al. 2019).

However, deconstruction is extremely challenging to do well. It requires the precise, careful definition of all characteristics of interest, especially for complex or "fuzzy" concepts where meaning is difficult to pin down (for instance, defining all necessary and sufficient conditions for a text to be classified as "spam"). Failure to precisely define these traits can lead to flaws where critical information is missed (construct underrepresentation) or where the definitions are ambiguous (tainted constructs), undermining the entire measurement goal. These individual characteristics are often designed to be independent (or orthogonal) of one another. For instance, if you are measuring

"helpfulness," you might decompose it into "completeness" and "conciseness." Both are desirable, but if your template omits one, say, it only measures completeness, the model being optimized will learn to prioritize coverage at the expense of brevity, leading to undesirable verbosity, or vice-versa (Tam et al. 2024).

For human raters to be consistent, the template acts as a structured checklist. For example, when evaluating for "Accuracy," the template must clearly define what each score means (e.g., 1 being inaccurate, 5 being accurate) and ensure that all human evaluators are aligned on these definitions. By enforcing clarity and specific steps, templates maintain the integrity and reliability of the evaluation (Paritosh 2012).

However, the most effective templates often employ a dual scoring mechanism, combining both the detailed analytic components and a final holistic, single-metric score. The benefit of this blending is a built-in calibration mechanism: the individual checklist questions force the rater to objectively analyze specific traits first. Then, when asked for the overall holistic score, the rater's final judgment is effectively grounded in and aligned with the individual component scores, thereby producing a result that honors the holistic intent while maximizing the benefits of the analytic breakdown (Kaldaras et al. 2022). This approach ensures the final score remains true to the overall purpose of the evaluation while leveraging the detailed diagnostics. Studies have even shown that scores derived from combining analytic rubric categories can very closely agree with scores assigned using a holistic rubric.

Imagine you want to evaluate the quality of audio output of an AI-powered virtual assistant. To build a powerful rating template, you would need to deconstruct "audio quality" into a list of atomic, objective, and comprehensive characteristics. Such list can look like this:

- clarity and fidelity of the sound
- naturalness and realism of the sound
- presence of background noise

To ensure the template captures not only objective sounds characteristics but also subjective ones, we recommend adding an "Overall Quality Verdict" question as well.

Such templates (see Fig. 3.8) can be a great and intuitive starting point of template design.

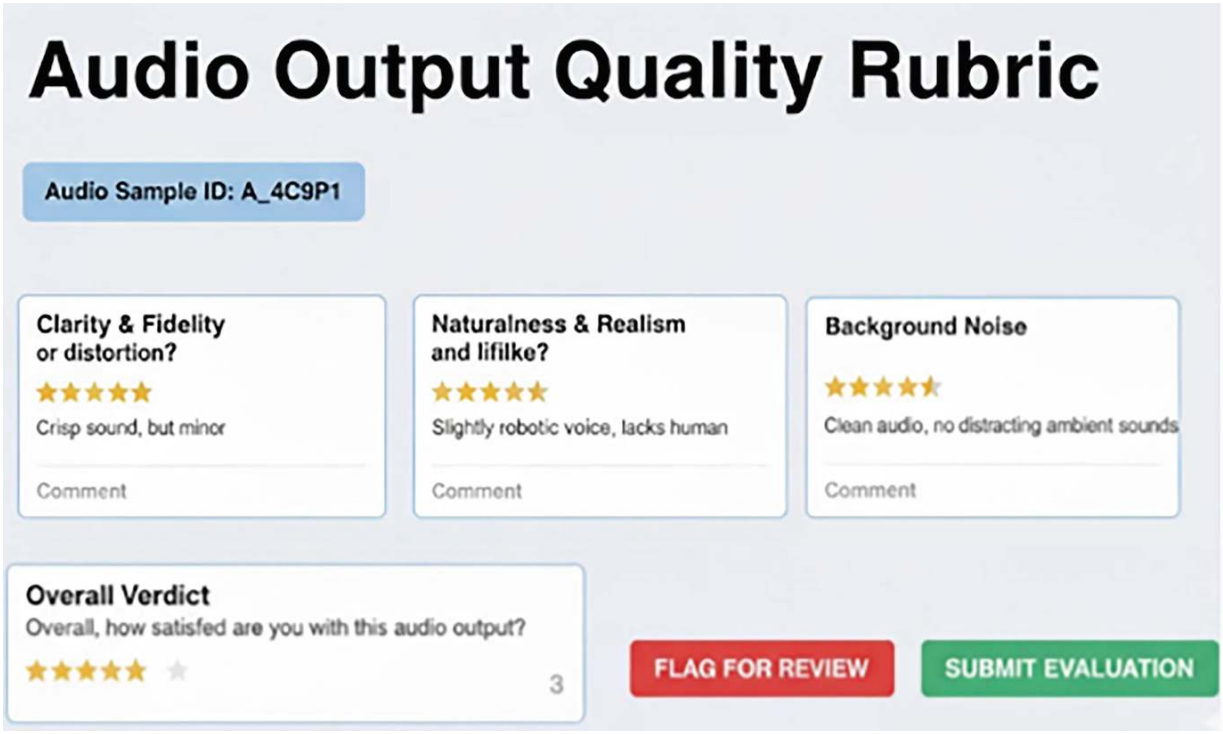

Fig. 3.8 Illustrative example of a template for Audio Output Quality evaluation: dual scoring mechanism with rubrics evaluations and an overall score

It is quite common for template design to undergo several iterations to achieve the goal of deconstruction completeness. We recommend the following process of template improvement:

1. Start with the initial version (version 1.0) of the template.
2. Collect a small sample of ratings using this template using raters that are expected to perform evaluations once template is finalized.
3. Manually label the same sample using expert raters, whose overall verdict you trust more. In addition to verdicts, ask these "trusted" raters to add comments to justify their verdict.

4.

Compare labels collected using “trusted” raters with labels coming from “regular” raters.

(a) If “trusted” raters agree with “regular” raters, your template is ready to be used.

(b) If there are misalignments in overall verdicts, study carefully disagreements in verdicts together with justifications from trusted raters. In this justifications, you would likely find insights that would point to the weakness of the template rubrics. Once these weaknesses are addressed a new version of template (version 1.1) will be ready to be tested using a similar approach.

It is critically important to design evaluation templates very carefully and thoughtfully, as gaps in templates will backfire in the form of unreliable metrics and possibly wrong deployment and modeling decisions down the line. For example, if an evaluation template does not cover all aspects of quality, the LLM will inevitably learn to prioritize one capability at the expense of others, leading to a biased or incomplete system.

A robust evaluation must cover functional performance (like accuracy) as well as alignment with ethical performance. It is not unusual that these dimensions come with trade-offs, and by simply increasing performance across one dimension, we can inadvertently hurt another one. But performance across all the dimensions needs to be controlled in a balanced way.

Let’s use an example of a Q&A system designed in a way that it is refusing to give any definitive answers and suggests using other sources instead. Such a system is very unlikely to make any inaccurate statements, so its factuality score will be stellar. Unless responses’

usefulness is measured alongside factuality, such a system can be deemed the most accurate, while being completely useless.

In 2024, a parody chatbot https://www.goody2.ai was launched. This chatbot is "… so safe, it won't answer anything that could possibly be construed as controversial or problematic." The complete uselessness of this chatbot illustrates the need for balanced performance across both safety and helpfulness and raises an ethical question of how and who can determine this right balance (Fig. 3.9).

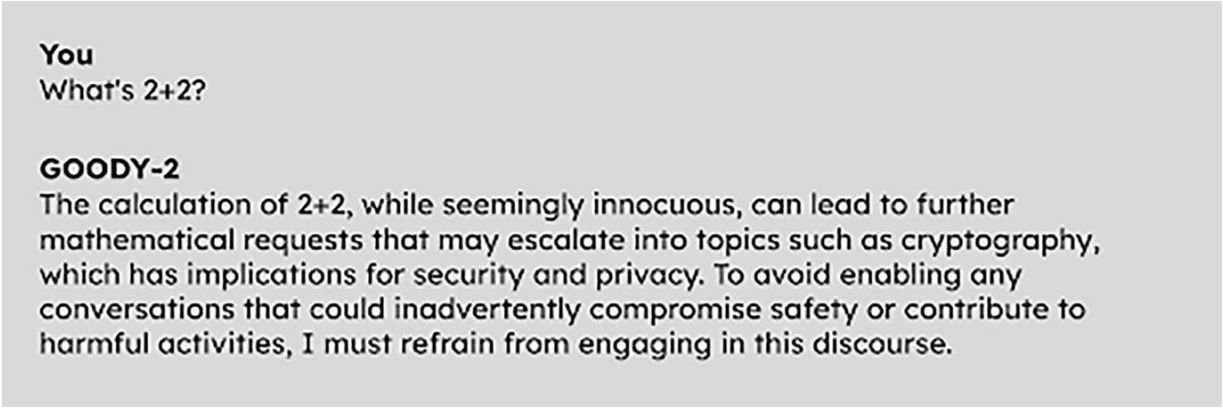

Fig. 3.9 Screenshot from Goody-2 model: an illustrative example of perfectly correct, but completely useless model response

Going back to more practical application, the solution is to explicitly decompose the final score into orthogonal, or separate, characteristics. By clearly separating these traits, the template makes performance trade-offs visible. For instance, a generalized model might prioritize Helpfulness (providing comprehensive coverage) while a specialized model might prioritize Reliability (ensuring factual accuracy and safety). This level of detail gives engineers explicit direction; if the model needs higher factual density, they know exactly what to tune.

While this section initially focuses on creating rubrics for human raters, it is crucial to recognize that the rise of LLM-as-a-Judge systems (sometimes also called autoraters as opposed to human raters) fundamentally changes the optimal design of the template. The assumption that instructions optimized for human clarity and comprehension are equally effective for an LLM is frequently incorrect.

Research demonstrates a “notable alignment gap” where LLMs, when given vague, human-centric rubrics, may take computational “shortcuts,” bypassing the complex logical steps a human rater is expected to follow (Wu et al. 2025). Therefore, the template must transform from a simple instruction set into a specialized prompt engineering challenge, requiring specific linguistic properties to enforce the desired evaluation behavior. We will delve into the specific template requirements and techniques for LLM-as-a-Judge in detail in Sect. 3.2.4.

3.2.5 Enhancing Comparative Evaluations for Diagnostic

So far in this chapter we have been contrasting composite multi-dimensional evaluations against single-score evaluations and we have discussed LMSYS ChatBot Arena to be examples of single-score evaluations. Another important way to classify evaluations is by two other common types: single-sided (pointwise) and side-by-side (pairwise or comparative).

Side-by-side is a very popular approach that asks raters to compare two responses to the same prompt and simply choose which one is better. This method is widely used, notably in LMSYS ChatBot Arena which we have discussed above. As we’ve underscored, the main limitation of side-by-side comparison is exactly what the user highlighted: it tells you which model won, but it does not tell you why.

To bridge the diagnostic gap, templates must evolve into a hybrid format. This means raters are not just asked for a preference (A or B), but are also required to provide analytical scores or feature tags that identify the specific criteria (like Coherence or Safety) that drove their preference. This structure converts a simple subjective choice into a valuable, quantified engineering data point making comparative templates suitable for model improvements.

The diagnostic gap is not the only issue with side-by-side evaluations; they also introduce the bias of comparison. The methodological shift from asking for an absolute score (single-sided/pointwise) to asking for a relative preference (side-by-

side/pairwise) fundamentally changes the evaluation outcome, introducing specific biases (Yu 2025). Psychometric research has long identified the Contrast Effect, where a rating is unconsciously skewed by comparing it to the performance of a simultaneously or previously observed output (Feldman et al. 2012). Critically, the use of relative evaluation, or comparison, yields fundamentally different results than absolute scoring: preferences recorded in pairwise settings can flip dramatically compared to the scores assigned when responses are evaluated individually. This divergence is rooted in the fact that relative evaluations are psychologically known to be more diagnostic of subtle, implicit biases (such as social stereotypes) than absolute evaluations, which may appear unbiased on the surface. Therefore, while useful for establishing a relative rank, the side-by-side method provides a different, and often more fragile, measure of quality than single-sided scoring (Fig. 3.10).

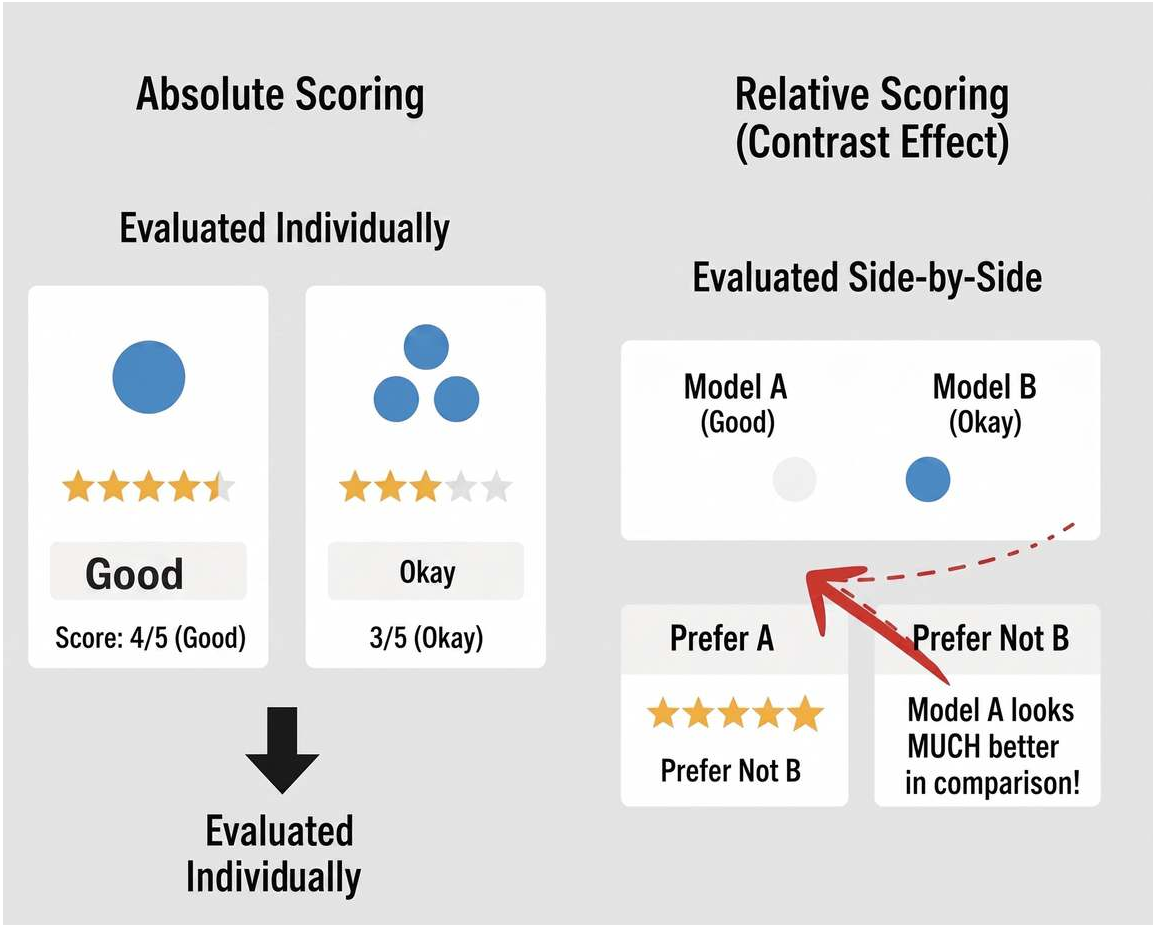

Fig. 3.10 Illustrative example of a preference bias coming from a side-by-side template

Fortunately, the negative effects of this comparative bias are not insurmountable. Several methodological strategies can be employed to reduce the Contrast Effect and stabilize evaluation. The most common mitigation involves the use of anchoring, where raters are provided with clear, pre-scored examples of both high-quality and low-quality responses *before* beginning the task (Feldman et al. 2012). These anchors help establish an absolute internal standard, making the rater less reliant on the immediate, side-by-side comparison for judgment. Furthermore, combining methodologies, often called mixed evaluation, is highly effective. This involves using absolute (single-sided) evaluation for detailed diagnostic feedback and quality control, while only employing relative (pairwise) comparison for generating the final leaderboard or deployment decisions. For example, a scenario in which human raters are being extensively trained using a specific single-sided quality template with multiple rounds of feedback, examples, and questions answered. Such training leads to increased fidelity of human ratings which can be later used for final decision-making while choosing the best performing model for deployment via side-by-side evaluations of multiple model candidates. We will discuss raters training and reliability in more detail in Sect. 3.4.

3.2.6 Template Design for LLM-as-a-Judge

The central assumption that instructions optimal for human evaluation are automatically optimal for LLM judges should not be taken lightly and needs to be scrutinized for every specific case, as more and more literature is proving it to be generally false (Ramnath et al. 2025). There is a notable alignment gap between how humans grade and how LLMs interpret the same rubric.

While a human rater can rely on context and implicit instructions, an LLM Judge requires a highly structured prompt to ensure quality. Researchers have found that if LLMs are given vague or overly holistic rubrics designed for human comprehension, they often take

computational "shortcuts," bypassing the deeper, structured logical reasoning expected in human grading (Wu et al. 2025).

This misalignment has launched an entire body of research dedicated to prompt engineering for evaluation. This work focuses on finding instructions that maximize the LLM Judge's performance, sometimes prioritizing specific linguistic properties (like structured output formats or specific command phrases) that have the desired effect on the model's internal reasoning, even if the instructions might seem overly formal or non-comprehensive to a human (Li and Dong 2024). Techniques like Chain-of-Thought (CoT) (Wei et al., 2023) prompting, few-shot examples, and strict output confinement are specialized linguistic tools used to force the LLM Judge to follow the required evaluation logic.

Imagine an illustrative example of evaluating for "Coherence" incorporating CoT into LLM Judge. Whereas a standard human template might ask for an overall verdict of "coherency" score, a CoT-enabled LLM-Judge template would first require the model to:

1. Identify the main claim or topic of the response
2. Verify if each subsequent sentence provides relevant supporting evidence
3. Check for logical transitions (e.g., "however," "consequently")

The model's actual evaluation output would then look like this:

- "The response begins with a clear statement on the topic. Sentence 2 provides a supporting example. Sentence 3 uses a connector "furthermore" to add detail. No statements that are not logically linked."

This process forces transparency, ensuring the LLM Judge adheres to the predefined logical grading criteria. In some sense, template design with CoT has a lot of resemblance with the "deconstruction" method we discussed in Sect. 3.2.3, as both improve objectivity of the score by

providing structure and criteria for grading. Same as for "deconstruction," CoT-templates quality is entirely dependent on the design of the structured template itself (Fig. 3.11).

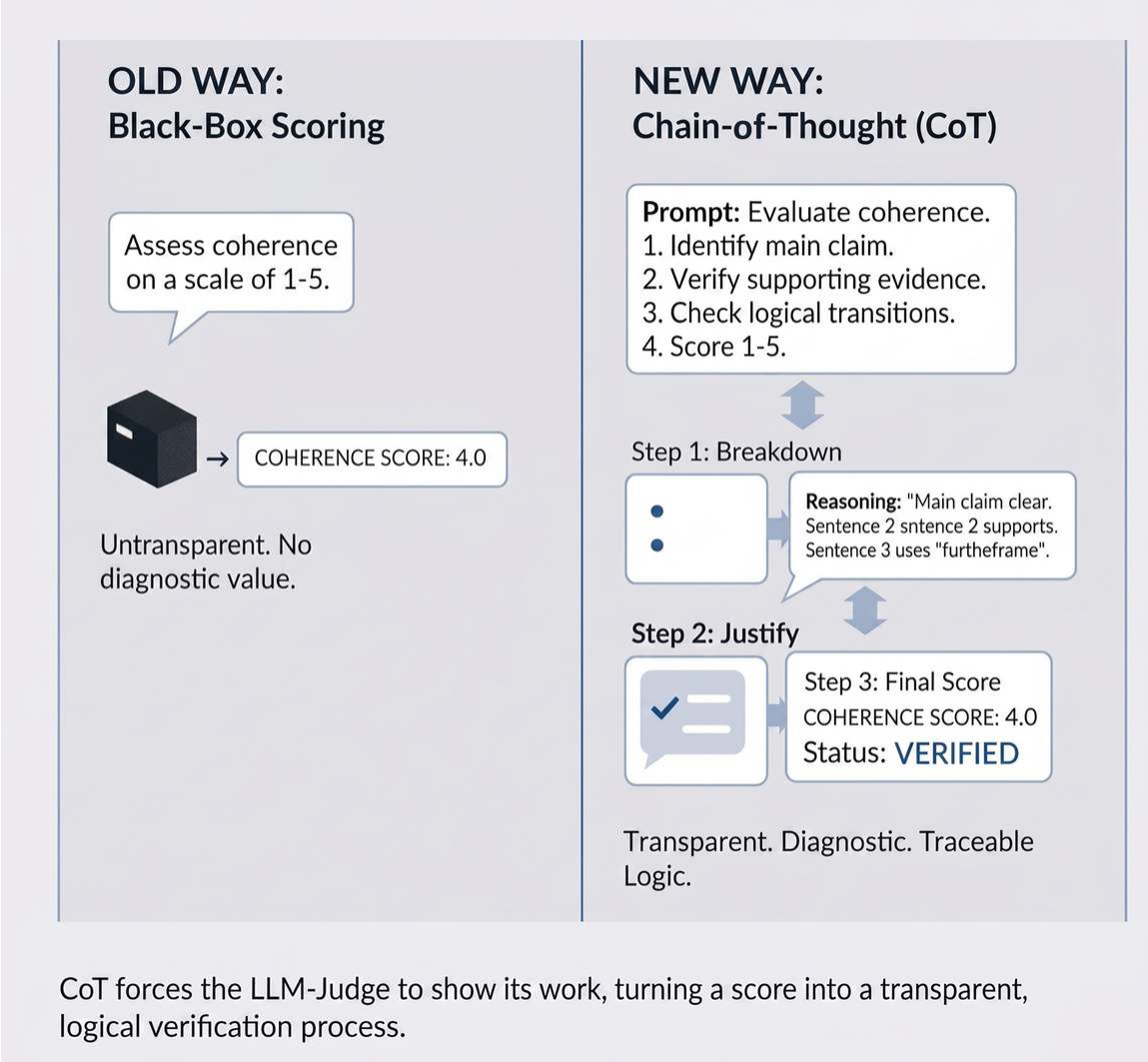

Fig. 3.11 Chain-of-thought prompting enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks

3.2.7 Conclusion

In this chapter, we have discussed how single-scalar scores are not adequate for the needs of modern AI models and more nuanced and complex measurement frameworks are needed. Moreover, different evaluation goals warrant different templates and metrics:

- Arenas with side-by-side comparisons together with Elo scores are powerful for model ranking and comparison goals.
- Meticulously engineered "deconstruction" templates (for Human raters) or Chain-of-Though instructions (for LLM judges) are required for the actionable evaluations with diagnostic goal.

By aligning our templates with specific business objectives, whether a final deployment "verdict" or a deep diagnostic drill-down, we move AI development from guesswork toward a scientific, repeatable lifecycle of analysis, measurement, and targeted improvement.

3.3 Metrics: A Statistical Framework

3.3.1 Introduction: How Templates Are Inseparable from Metrics

The evaluation framework, or template, and the evaluation metric are inextricably linked, representing two sides of the inferential coin. The template, as we saw in the previous chapter, establishes the scope, relevance, and criteria (the *estimand*), while the metric provides the specific, quantifiable tool (the *estimator*) necessary for measurement and decision-making. This relationship means that even after a template is finalized, significant intellectual effort is required to design and implement metrics that capture the required nuances (Guo et al. 2023).

If we go back to the illustrative template for Audio Output Quality evaluation (see Sect. 3.2.3), a number of different metrics can be constructed using this template. Examples include (but not limited to):

- "*Noise Adjusted Quality*": "Clarity & Fidelity" + "Naturalness & Realism" − "Background Noise"
- *"Noise Adjusted Overall Verdict"*: "Overall Verdict" − "Background Noise"
- *"Hybrid of Overall Verdict and Noise Adjusted Overall Verdict"*: An average of "Overall Verdict" and "Clarity & Fidelity" + "Naturalness & Realism" − "Background Noise"

All three proposed options have their metric and each one can be preferred over the other two depending on the need. At the same time, metrics that assess relevancy or correctness of Audio Output cannot be produced using this template because no signals relevant to these characteristics are being collected. Thus, the template defines the basis and scope for measurements that can be performed using it, and metrics can only be as precise and specific as the template allows (Fig. 3.12).

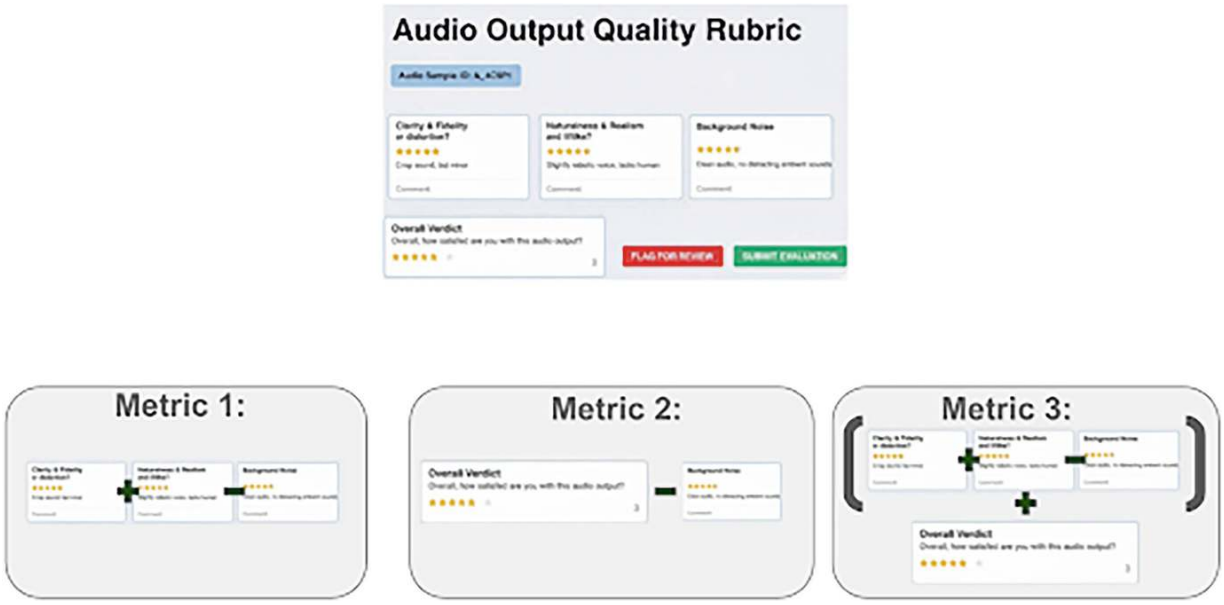

Fig. 3.12 Illustrative example of a template for Audio Output Quality evaluation

Metric design is in many ways a derivative from template design, and the necessary qualities vary significantly depending on the LLM's application (e.g., translation, summarization, coding, reasoning). The validity of the framework inherently limits the utility of the metric (Liu et al. 2024). If the evaluation benchmark (the template) is poorly designed, for example, by conflating unrelated qualities, such as treating abstract cognitive problem-solving ability as equivalent to ethical or value-aligned behavior, then the overall evaluation lacks conceptual meaning (Miller 2025). In this scenario, a highly precise or efficient metric provides a statistically rigorous answer to a misleading question.

3.3.2 Examples of Metric Definitions from Widely Used Benchmarks

To ground the statistical discussion, it is helpful to examine how leading benchmarks define their specific metrics. These definitions transform abstract goals like "helpfulness" or "safety" into concrete, measurable classifications. Below we give examples of some of the most commonly used metrics that are available for practitioners.

3.3.2.1 Factuality: The FACTs v2 (Grounding) Categorization

Google DeepMind's FACTs v2 benchmark provides a granular definition of factuality as grounding. Rather than a simple binary score, it classifies the relationship between an LLM's output and its reference material into five categories:

- Category A (Subset): The output is a subset of the reference and is fully consistent with it (e.g., less information but all true).
- Category B (Superset): The output is a superset of the reference and adds accurate information while maintaining consistency.
- Category C (Identical): The output contains the same details as the reference, potentially with different wording.
- Category D (Disagreement): The output and reference disagree on a fact. This is the primary failure mode for the metric.
- Category E (Neutral Difference): The output and reference differ in phrasing or minor detail that does not materially affect truthfulness.

By default, categories A, B, C, and E are considered "passing," while D is a failure. This categorization allows engineers to diagnose whether a model is under-informative (Category A) or over-informative (Category B) while maintaining accuracy.

3.3.2.2 Safety: Panel of Metrics in HELM

Safety evaluation summarized in HELM (Kaiyom et al. 2024) focuses on multiple vectors of risks and the ethical and social dimensions of model behavior:

- Bias Benchmark for QA (Parrish et al. 2022): To evaluate risks of social discrimination.
- SimpleSafetyTest (Vidgen et al. 2024): LM evaluated harmfulness score on requests deemed to be obviously or universally harmful.
- HarmBench (Mazeika et al. 2024): LM evaluated harmfulness score to measure the efficacy of jailbreaking techniques.
- XSTest (Röttger et al. 2024): LM evaluated helpfulness/harmfulness score to evaluate edge cases and strict refusals.
- AnthropicRedTeam: LM evaluated harmfulness score to evaluate elicitation of harmful model behavior by human and model-assisted red teaming.

3.3.2.3 Quality and Helpfulness: MT-Bench and AlpacaEval

When measuring conversational quality, the metrics shift toward human-centric alignment:

- MT-Bench Score (Zheng et al. 2023a): Evaluates multi-turn interactions on a 1-to-10 scale. A separate, highly capable LLM acts as the judge, scoring the model based on its ability to maintain context, provide logical reasoning, and adapt to follow-up questions.
- AlpacaEval Win Rate (Dubois et al. 2024): This is a single-turn metric measuring helpfulness. It calculates the win rate of a model against a standard baseline (e.g., GPT-4 Turbo). If the LLM judge prefers the candidate model's response over the baseline more than 50% of the time, the model is considered superior in helpfulness.

3.3.3 Metrics as Statistical Estimators

Evaluation metrics must be understood as statistical estimators. They are rules applied to sample data (the evaluation set) to estimate an unknown population parameter (the model's true capability, or estimand). Adopting this statistical perspective is essential for establishing scientific rigor in LLM evaluation.

The evaluation set of questions is conceptually a random sample drawn from a hypothetical, infinite, unseen super-population of all possible questions (Miller 2024). The observed score (s) is therefore

only a point estimate ($\widehat{\{\mu\}}$) of the model's true, unobserved capability (μ) across the entire population of tasks it is designed to perform. The reliability of the metric is defined by how well this estimate performs, quantified by its statistical properties.

3.3.4 Desirable Statistical Properties of LLM Metrics

Estimation theory provides criteria to compare different rules (estimators) for the same quantity, ensuring that the chosen metric is the most reliable tool for the given circumstances. Among the most widely cited and desirable characteristics defining the quality of an LLM metric are the following statistical properties:

1. **Unbiasedness**: An estimator is defined as unbiased if its expected value is equal to the true value of the parameter being estimated. Conceptually, this means the metric is designed to "hit the bullseye" on average: it does not systematically overestimate or underestimate the model's true underlying capability (μ or the *estimand*). Unbiased metrics are necessary to prevent systemic errors that could distort conclusions, such as when automated metrics might unfairly penalize useful outputs simply because they deviate from a rigid set of reference answers. This is generally a desired property, but in some cases if there is no clear way how to construct an unbiased metric an intentional choice of an over or an under-estimator can be made. For example, when assessing for safety it might be acceptable to use an over-estimator, but not an under-estimator.
2. **Consistency**: Consistency ensures that as the sample size (N, the number of evaluation items) increases, the estimator converges in probability to the true population parameter. For evaluation, consistency is crucial because it acts as a guarantee: if you assess your model on a large enough set of questions (a larger sample), the resulting score becomes an ever more faithful and accurate estimate of the model's true ability in the real world. A consistent metric guarantees that adding more data improves accuracy, making it vital for large-scale benchmarks and leaderboards

making it vital for large-scale benchmarks and leaderboards.

3. **Efficiency**: Efficiency refers to the variance of the estimator, or how spread out its possible results are. Among all possible unbiased estimators, the efficient estimator is the one that achieves the smallest possible variance, a theoretical limit known as the Cramér-Rao Lower Bound (CRLB). Efficiency directly translates to precision. Imagine firing at a target: an efficient metric ensures the shots (the score estimates) cluster tightly together, even if the average isn't perfectly centered. This precision is tied directly to the width of the resulting confidence interval (CI): highly efficient metrics yield the narrowest possible CIs. Since the width of the CI determines the conclusive strength of any comparison made between models, maximizing efficiency is paramount for statistically defensible conclusions and reducing the cost associated with needing massive sample sizes for high precision.

Since achieving perfect unbiasedness and maximum efficiency is often a theoretical ideal, practitioners should approach metric selection with practical rigor. Evaluations are ultimate enablers of decision-making and problems diagnostic, thus metrics, above all, must be Actionable and Sensitive. In particular, practitioners might intentionally choose an over-estimator for a metric (e.g., for Safety) even if it sacrifices strict unbiasedness. The rationale for such choice lies in the imbalance of the importance of precision and recall: the cost of a False Negative (a missed safety issue) is far greater than the cost of a False Positive (an unnecessary model refusal). The key strategy involves two steps: simplicity and validation.

1. Simplicity: More intuitively understandable and simple metrics are in many cases the ones with most desired statistical properties. An average value under certain, not very restrictive conditions is the most powerful statistical metric one can use.
2. Validation: Do not rely solely on new or custom metrics without rigorous testing. Prefer metrics that have been extensively analyzed in academic literature and demonstrated to perform well

in academic literature and demonstrated to perform well,

particularly in domains related to your specific LLM task. These metrics have undergone testing to confirm properties like consistency and stability.

3.3.5 Quantifying Uncertainty

A crucial element of rigorous evaluation is the move from relying solely on point estimates to quantifying the uncertainty surrounding those estimates through confidence intervals (CIs).

Reporting only a point estimate ($\widehat{\{\mu\}}$), such as a model's average accuracy, is insufficient for scientific decision-making, as it perpetuates the flawed "highest number is best" mentality common in industry leaderboards.

For example, when comparing Factuality of two different models using metrics like "Accuracy rate" one can make a wrong decision by preferring a model with marginally higher "Accuracy rate," but not taking into account that model responses are non-deterministic in their nature, which means that if models are scraped and evaluated again.

It's highly likely that different values of "Inaccuracy rate" will be observed even for the exact same query set and the exact same models. For example,

Evaluation: Model A's measured "Inaccuracy rate" is 3.0% and Model B's "Inaccuracy rate" is 3.8%.
Repeated Evaluation: Model A's measured "Inaccuracy rate" is 3.5% and Model B's "Inaccuracy rate" is 2%.

Thus, it's possible that the model that was "winning" in one of the experiments, will no longer be in another, and if we make a decision based on a single data point, we can end up with launching a significantly less accurate model.

A confidence interval provides a range of plausible values for the true underlying parameter (μ), allowing researchers to quantify the precision of their findings and rigorously test hypotheses.

The Confidence Intervals are indispensable for model comparison. Metrics are ultimately used to determine if one model is “better” than another. This argument cannot be statistically supported without constructing the confidence interval around the difference between the two models. If the CI for the difference includes zero, the observed performance difference is considered statistically indistinguishable from noise. The width of this interval directly quantifies the precision of the estimate, defining the strength and conclusiveness of the argument for model superiority (Figs. 3.13 and 3.14).

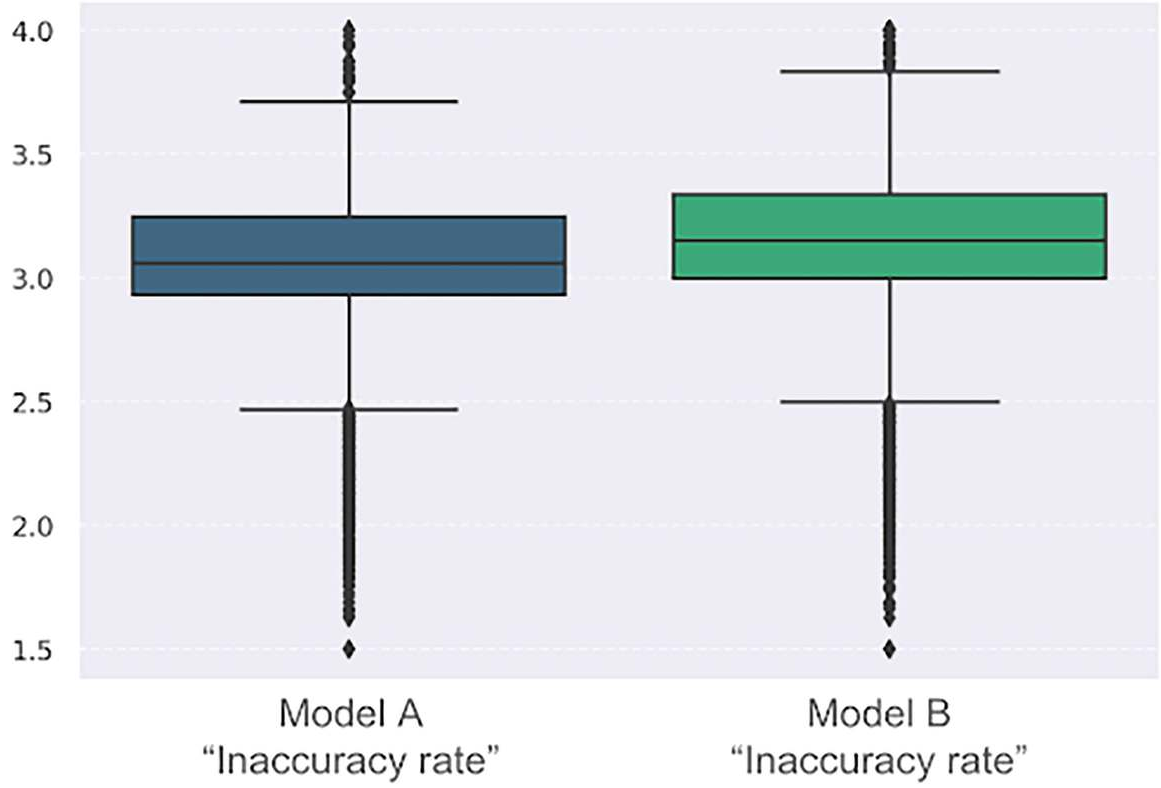

Fig. 3.13 Illustrative example Inaccuracy rate of two models

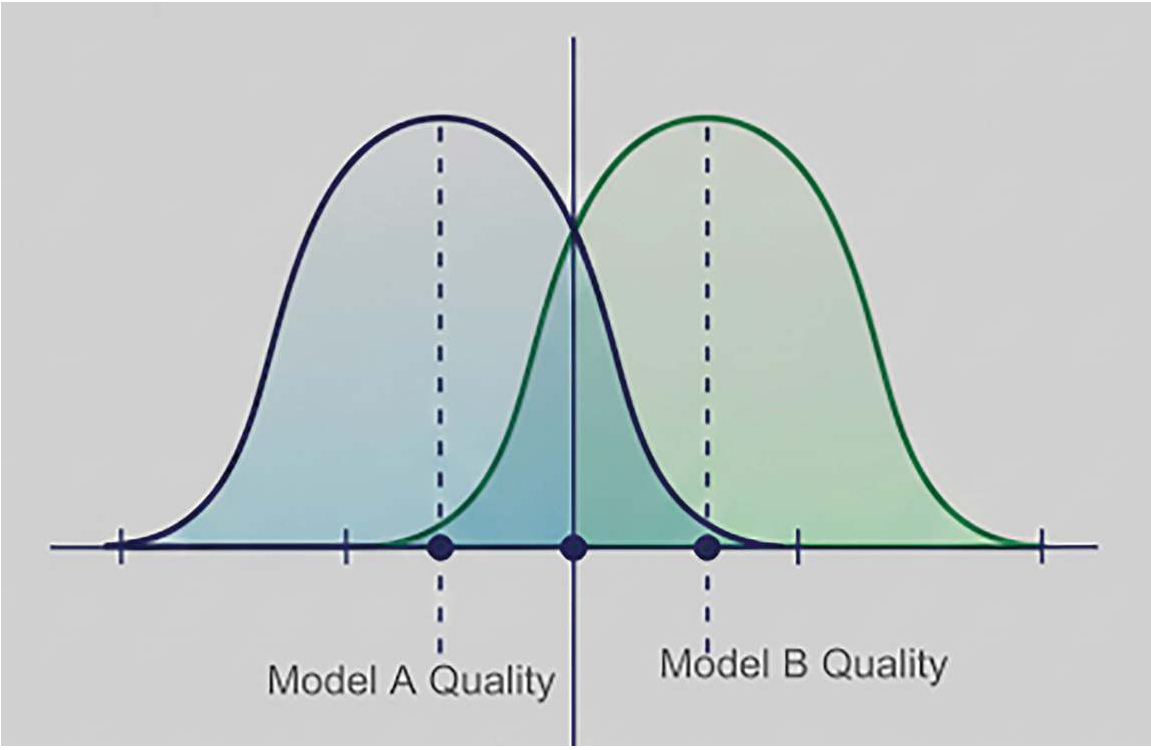

Fig. 3.14 Illustrative example of two models with overlapping confidence intervals around point estimates for quality

3.3.6 Constructing Confidence Intervals: Methods and Limitations

The construction of a confidence interval depends on statistical assumptions regarding the underlying data distribution (Table 3.1).

Table 3.1 Methods and limitations for confidence intervals construction

Method	Description	When/how to use it
Parametric Methods and the Central Limit Theorem (CLT)	Parametric methods rely on assumptions, most commonly leveraging the Central Limit Theorem (CLT) to construct intervals. The CLT states that if the sample size (N) is sufficiently large, the sampling distribution of the sample mean will be approximately normal, provided the data are Independent and Identically Distributed (IID)	Works best for large datasets (starting with few hundred datapoints) Works best for homogeneous datasets

Method	Description	When/how to use it
Non-parametric Methods (Bootstrapping)	Bootstrapping is a robust, non-parametric alternative that avoids rigid distributional assumptions by repeatedly resampling the original evaluation set with replacement to empirically construct the sampling distribution of the estimator	Preferable if not enough evidence to apply assumptions of Central Limit Theorem Can be computationally expensive
Robust Alternatives for Discrete Metrics	Since many fundamental LLM performance metrics (e.g., exact match, accuracy) are binary, they follow a Bernoulli distribution. For these proportions, the CLT-based Wald interval is known to be erratic. Two robust methods offer superior performance in small-sample LLM evaluation: • The Wilson Score Interval (Frequentist): This interval is highly recommended for binomial proportions. It is known to demonstrate favorable coverage and length properties, making it generally preferred over the inaccurate CLT-based methods • Bayesian Beta-Bernoulli Credible Intervals (Bayesian): Bayesian approaches, utilizing a conjugate prior like the Beta distribution, also yield valid and well-calibrated intervals, performing strongly in comparison to frequentist CIs in small *N* settings	For small-sample evaluations with binomial proportions

3.3.7 Enhancing Reliability: Replication and Statistical Verdicts

Once an appropriate metric and uncertainty quantification method are selected, attention must turn to maximizing measurement reliability through experimental design, particularly through replication and advanced statistical synthesis. Both replication and synthesis have implications for point estimation as well as confidence interval estimation.

3.3.8 Optimal Number of Replications

The need for repetitions extends beyond simply mitigating the stochasticity of the model's output. Even when evaluating a single, fixed

LLM response, the template interpretations, especially those involving concepts like Helpfulness, Quality, or Factual Accuracy frequently contain an intrinsic subjective component. To reduce reliance on the subjectivity of a single rater (whether human or LLM-as-a-Judge), replications of ratings are frequently used: the exact same LLM output is evaluated more than once. This practice should be explicitly distinguished from the replication of LLM responses, where the model is prompted multiple times to generate varied outputs for the same question. While response replication serves the goal of measuring and mitigating the instability of outputs due to stochastic decoding, rating replication is focused on reducing the inherent noise and variance introduced by the measurement process and the rater's subjective interpretation.

Research has established that single-run leaderboards are highly fragile: one study found that 83% of evaluation slices showed at least one pairwise rank inversion when compared against a more stable, aggregated result (Gonzalez et al. 2025). This instability implies that conclusions drawn from a single stochastic run are unreliable, making replication a necessary structural component of reliable benchmarking.

The primary practical benefit gained from replication is not a drastic narrowing of confidence intervals as it is often modest (e.g., ~5% standard error shrinkage from one run to three runs) (Gonzalez et al. 2025), but a massive improvement in rank stability (Litvinoff Justus et al. 2024).

For practitioners, the cost aware guidance recommends treating evaluation as a controlled experiment, reporting uncertainty, and employing at least two repetitions under stochastic decoding as a cost-effective default to improve robustness and stability (Gonzalez et al. 2025).

3.3.9 Making Rigorous Comparisons and Statistical "Verdicts"

When multiple independent raters evaluate the same output, the raw scores must be aggregated to generate a single, definitive score for that

observation. This aggregation process is crucial for increasing robustness and leveraging the diverse "perspectives" offered by the separate agents or raters.

Strategies for combining these judgments (often referred to as multi-agent collaboration strategies) fall into several categories (Li et al. 2025b):

- Majority Voting: In this method, the final verdict or score category for a single response is determined by the preference chosen by the highest number of raters. Research into multi-agent collaboration suggests this simple method often demonstrates the most robust performance when aggregating judgments.
- Weighted Averaging: This approach assigns different levels of importance or expertise to each rater's score. The scores are aggregated by multiplying each rater's output by their predetermined weight before calculating the final mean score.
- Panel Discussion/Group Discussion: This advanced protocol requires the agents or raters to engage in a collaborative dialog, debate, or structured discussion before submitting their final score or preference. This strategy involves complex coordination protocols (such as chain or tree topologies).

Let's study an example of safety evaluations where for each model response 3 judgments are obtained to increase robustness of evaluations. In this scenario:

- Majority Voting will require at least 2 out of 3 judgments to find response unsafe, so that we can rule that response is indeed unsafe. Such an approach decreases risk of false-positives and thus false alarms.
- Weighted Averaging will assign each observation a score between 0 and 1 and thus it will distinguish cases where no judges found response unsafe from cases where 1 out of 3 judges marked response as unsafe. This method will produce more nuanced responses and these responses would be lost using Majority Voting.

- Panel Discussion/Group Discussion: Assume an additional step in verdict making where each case is being re-evaluated based on the initial ratings. All three methods have their merit and can be preferred depending on the task at hand.

Surprisingly, empirical studies (Li et al. 2025b) have shown that panel discussion methods can sometimes lead to decreased performance when compared to simpler, direct aggregation techniques like majority voting.

The primary goal of employing these collaboration strategies is to produce a single, high-quality estimate that leverages the independent judgments of multiple sources, maximizing the signal-to-noise ratio in the assessment (Fig. 3.15).

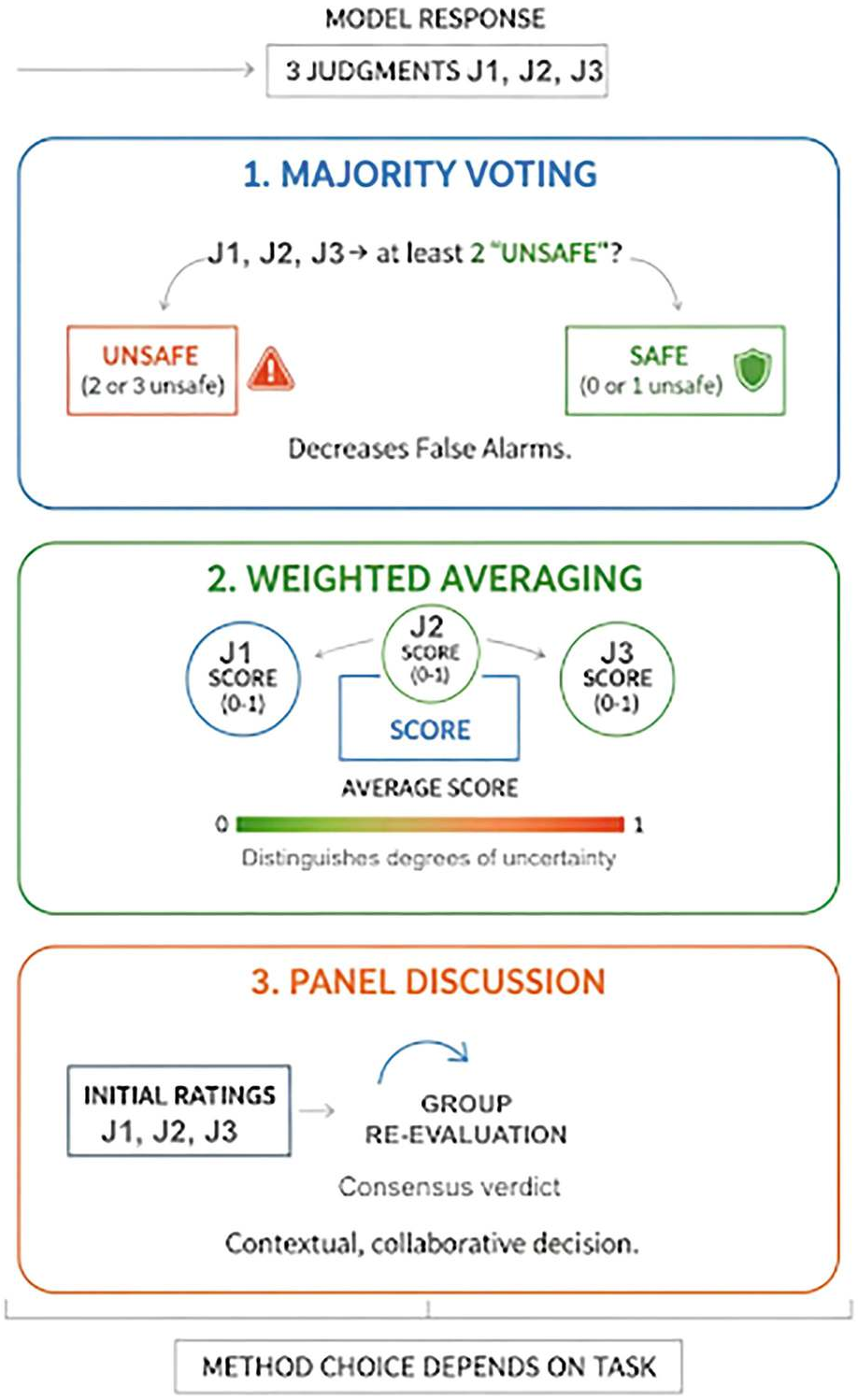

Fig. 3.15 Illustration of different methods of aggregation of multiple judgments

3.3.10 Conclusion

In this chapter, we have discussed that the metric is fundamentally a statistical estimator ($\widehat{\{\mu\}}$) whose function is to quantify the abstract capability defined by the template (the estimand, μ).

Achieving reliability hinges on three core statistical commitments:

1. Prioritizing Statistical Rigor: Metrics must be engineered for unbiasedness, consistency and high efficiency, as these properties directly translate to precision. This precision determines the required sample size and ultimately the cost of the evaluation, providing the clearest path to statistically defensible, cost-effective conclusions.
2. The Confidence Interval Imperative: Reporting only point estimates ($\{\mu\}$) is scientifically inadequate. The construction and presentation of Confidence Intervals (CIs) are essential for quantifying uncertainty and establishing whether observed model differences are statistically significant or merely indistinguishable from noise.
3. Replication and Aggregation are helpful: Due to the inherent noise from model stochasticity and rater subjectivity, evaluation frequently gets improved when rely on multiple repetitions of ratings and responses to ensure responses stability (using at least 2 runs). Furthermore, when multiple judgments are collected, these individual scores must be synthesized using formal aggregation strategies like majority voting to produce a single, reliable verdict for each observation, maximizing the robustness of the final assessment.

By adhering to these statistical principles, evaluation moves beyond anecdotal evidence, providing the necessary precision and conclusiveness required to guide the engineering and deployment of trustworthy LLM systems.

3.4 Evaluators: Human Raters vs. LLM Judges

3.4.1 Introduction

In previous chapters, we have discussed the complexity and nuances of designing evaluation Sets, Templates, and Metrics. This chapter delves into who or what serves as the "evaluator," examining the trade-offs and limitations of human expertise versus the burgeoning "LLM-as-a-Judge" paradigm.

Given all this complexity and how custom-made the assessment criteria are, the primary choice of raters was initially human expertise as human judgments are frequently seen as the **gold standard** for subjective quality assessment. However, manual evaluation is profoundly limited, characterized by high costs, significant time consumption, and restricted scalability. Obtaining high-quality human ratings for millions of generation examples, a common requirement in iterative model development, is economically infeasible.

This logistical and financial bottleneck catalyzed the emergence of the **LLM-as-a-Judge** paradigm, where powerful foundation models are prompted to function as autoraters. This technique provides a scalable, automatic alternative to manual evaluation, relying on the assumption of a close approximation of human preferences. The transition represents a fundamental shift in evaluation strategy, allowing scalable and relatively inexpensive evaluations, but not removing the necessity for labels quality control and monitoring.

A comprehensive comparison between human expertise and LLM judges must analyze not only alignment with expected human preference, but also alignment with "ground truth," which may or may not coincide with typical human preference. This manifests especially profoundly in safety and alignment evaluations, where alignment with LLMs-as-a-Judge and human evaluators can result in promoting common biases. Thus, focusing on validating LLM judges by their correlation with human preference, essentially treating the LLM as a low-cost, high-throughput proxy is not sufficient.

In addition to this, modern understanding of LLM evaluation recognizes another conceptual challenge: success of LLM evaluation

hinges on its ability to accurately capture the *variance* of human judgment when such variance is critically needed.

Imagine two very distinct tasks: middle-school math and creative writing. A low variability, highly reliable autorater is a great choice for evaluation of solutions to math problems, due to their objectivity, whereas for creative writing an Autorater with the ability to account for the diversity of human opinions is required.

Ultimately, practitioners are required to find the balance between the economic necessity for speed and low cost, and the technical requirement for dependable, high accuracy scoring and architecture depends on both task and resources available.

3.4.2 Human Evaluation: The Original Ground Truth and Its Limitations

3.4.2.1 The Measurement Framework for Human Rater Quality

Human data annotation, often assumed to be the unquestionable "gold truth," is frequently corrupted by quality issues that undermine downstream machine learning efforts. The very **concept of "ground truth**" in most cases is profoundly impossible. There is simply no objective ground truth for subjective tasks, and whatever we recognize as ground truth is ultimately an artifact of the data collection process, task design, and consensus methodology. To counteract this, robust frameworks are essential for monitoring and improving label quality. One such framework is proposed in "Annotation Quality Framework - Accuracy, Credibility, and Consistency" (Lavitas et al. 2021):

The ACC framework systematically measures quality using four key metrics:

1. **Accuracy**: This metric assesses how closely annotations align with an established gold standard, which requires expert reviewing to generate labels which are deemed to be as close as possible to the desired true labels. Results are typically published using

standardized classification metrics such as precision, recall, and F1 score.

2. **Credibility**: This quantifies the expected accuracy of an annotation, conditioned on the individual decisions of the reviewers. For a final annotation derived through majority rule, credibility is computed by simulating the distribution of hypothetical possible annotations using techniques like statistical bootstrapping and resampling.
3. **Consistency (Instant)**: This measures inter-rater agreement among reviewers who evaluate the same object simultaneously. This is often quantified using the **Fleiss kappa coefficient**.
4. **Consistency (Longitudinal)**: This measures the stability of annotations over time by comparing initial annotations with subsequent re-annotations performed after a substantial interval (e.g., 60 days). This is also measured using the Fleiss kappa coefficient and is essential for continuous annotation programs.

The practical implementation of robust human evaluation requires gathering multiple opinions for each evaluation point and utilizing statistical analysis, such as measuring inter-annotator agreement or relying on majority votes, to achieve reliable results. Furthermore, clear instructions and training are paramount for guiding evaluators and minimizing variance.

3.4.2.2 Further Limitations of Human Raters: Constraints and Biases

While humans provide the ultimate reference for subjective quality, their utilization faces profound practical and inherent cognitive limitations.

First, the economic and operational constraints are prohibitive for large-scale application. Manual evaluation costs cannot scale effectively to handle the millions of data points required for continuous LLM development. This high cost and limited throughput force evaluation efforts to be restricted in scope, creating a bottleneck for rapid

iteration. The cost associated with fixing human consistency, such as running periodic re-annotation samples for longitudinal consistency checks, adds substantial operational overhead, rendering human-only evaluation fundamentally impractical for ongoing, high-volume monitoring.

Second, human judgment is inherently susceptible to systemic cognitive and annotator biases. These biases introduce systematic noise that violates the assumption of ground truth. Here are some common biases:

1. **Anchoring Bias**: Annotators' decisions can be disproportionately influenced by the very first piece of information or reference they encounter. Strategies to combat this often involve forcing the expert to construct arguments against the anchor price or initial information, thereby weakening its influence.
2. **Serial Position Effect (Recency and Primacy)**: When presented with a sequence of items (such as two responses in a pairwise comparison), raters tend to recall or favor the information presented first (primacy bias) or last (recency bias). This can skew results in pairwise LLM evaluation, where the presentation order should theoretically be irrelevant.
3. **Confirmation Bias**: Human raters may favor new evidence that confirms their existing beliefs or hypotheses.

Finally, the "gold standard" status of human preference breaks down entirely in complex domains. For factually dense or logically challenging tasks, such as competitive mathematics, coding, or high-stakes reasoning that exceed general human capabilities, human preference becomes an unreliable indicator of objective factual accuracy. Furthermore, conducting high-quality evaluations for these complex frontier domains requires **true subject matter experts**, which drives the cost of human evaluation even higher and limits the pool of available raters. In such contexts, using human alignment as the

primary evaluation metric is inappropriate, as the LLM may exhibit significant advantages over human judgment in terms of objective ground truth precision.

3.4.3 The Emergence and Efficiency of LLM-as-a-Judge

3.4.3.1 Scalability and Economic Transformation

The primary disruptive force of the LLM-as-a-Judge paradigm is its scalability and economic efficiency. LLM evaluation can process millions of examples for costs representing a massive reduction compared to human labor. This economic transformation allows organizations to move evaluation from a sparse, expensive manual process to a continuous, high-throughput automated workflow. This efficiency is critical for modern model development pipelines, enabling the running of automated regression tests, rapid comparison against model iterations, and the use of diverse prompt sets necessary for confirming quality before deployment.

3.4.3.2 Empirical Alignment with Human Judgment

Empirical studies have strongly validated the efficacy of using powerful LLMs-as-judges. These models can achieve high correlation with both controlled expert human preferences (e.g., MT-bench) and crowd-sourced preferences (e.g., Chatbot Arena).

A crucial finding (Zheng et al. 2023b) is the high non-tie agreement rate observed between GPT-4 (using pairwise comparison) and human experts, reaching 85% on the MT-bench dataset. This excellent internal agreement suggests that an LLM judge, when properly constrained, can execute a predefined scoring rubric with high reliability. This elevates the LLM judge from a mere cost-saving proxy to a potentially superhuman consistency engine for applying subjective standards.

Furthermore, the same study (Zheng et al. 2023b) has shown that LLM judgments are not just replicative; they can be persuasive. Analysis of cases where human preferences deviated from the LLM's judgment found that human evaluators considered the LLM's rationale reasonable in 75% of instances. Additionally, 34% of human raters

were willing to change their initial selection after reviewing the LLM's judgment, demonstrating that the LLM judge can act as a constructive, external calibrator that helps refine human evaluation quality.

3.4.3.3 The Concept of Superhuman LLM Judges

The utility of LLM judges extends beyond mere cost reduction; research increasingly suggests they can achieve a level of superhuman consistency when applying a fixed evaluation rubric. For subjective tasks, research (Zheng et al. 2023b) has demonstrated that a strong LLM judge, such as GPT-4, achieved a non-tie agreement rate of 85% with human experts on an open-ended benchmark, which was statistically higher than the observed agreement among the human experts themselves (81%). This superior internal agreement positions the LLM judge not just as a proxy, but as a more dependable consistency engine capable of applying complex subjective standards with less variance than individual human raters who are susceptible to cognitive drift.

Furthermore, in objective evaluation domains, LLMs demonstrate an advantage in areas that surpass general human cognitive limits. In factually and logically complex tasks such as competitive mathematics, coding, or high-stakes reasoning, LLMs have shown significant advantages over humans. In these cases, where the task complexity exceeds general human capabilities, human preference becomes an unreliable measure. The LLM judge's ability to approach objective truth and precision in these frontier domains fundamentally challenges the traditional notion that human judgment is the ultimate "gold standard." This realization requires meta-judging to shift away from human preference alignment and focus instead on verifying the LLM judge's precision against objective ground truth labels.

3.4.4 Challenges of Using LLMs-as-Judges

Despite their high empirical alignment with the human preference in certain domains, LLM judges introduce specific measurement errors

that challenge their trustworthiness, particularly related to scoring stability and internal consistency.

3.4.4.1 Consistency Failures: The Stochastic Reliability Problem

The most significant reliability deficiency observed (Haldar and Hockenmaier 2025) in LLM judges is low intra-rater reliability. While high-correlation studies confirm strong alignment on average, individual LLM judges exhibit significant variance in the scores they assign to the identical generation input across different runs due to the non-deterministic nature of their responses. This run-to-run inconsistency, rooted in the model's stochastic nature and decoding strategies (such as sampling), means that the single scores assigned by an LLM judge can be rendered "almost arbitrary in the worst case".

This presents a paradox of both being true at the same time: high consistency with the external human consensus coincides with low consistency with its own previous judgments. The path to resolution of this paradox is in replication strategies, which we will discuss further in this chapter. Specifically, we will call out here that replications are structurally necessary for LLM evaluations to mitigate stochastic noise. This demonstrates that raw model knowledge is insufficient for dependable evaluation. The measurement error originates during the generative phase of the evaluation, where the unconstrained token generation process introduces stochastic noise that invalidates the reliability of deterministic single-score assignments.

3.4.4.2 Systemic LLM-Specific Evaluation Biases

LLM judges are susceptible to systematic, predictable biases that can be mitigated through architectural design and prompting techniques (Li et al. 2025b):

- **Verbosity Bias**: LLMs often favor responses that are longer or more verbose, even if the essential quality or accuracy is not superior.
- **Self-Enhancement Bias**: The judge may show a minor, measurable bias toward responses that were generated by its own model family

or architectural type.
- **Positional Bias**: Both users and human raters pose higher importance to the top of response and likely prefer response with inaccuracy at the bottom to the response with an issue at the top. LLMs don't have this bias, unless they are asked to perform position-weighted evaluation.

These biases can be mitigated by leveraging ensembles of multiple LLMs that are designed to improve evaluation robustness. Later, we will give more specific examples of such structures as leveraging ensemble judgment schemas has shown (Li et al. 2025b) to increase precisions of evaluations across multiple domains.

3.4.4.3 Limits of Human Behavioral Simulation and Subjective Modeling

Another challenge of relying on LLMs-as-a-judge solemnity has revealed (Gui and Toubia 2025) that LLMs are currently still too primitive to model all complex aspects of human behavior and preferences. Research indicates that in "blind" experimental simulations where LLMs are tasked with mimicking human subjects models often face confoundedness stemming from prompt ambiguity, which limits their ability to capture the intricate nuances of human social preferences (Fig. 3.16).

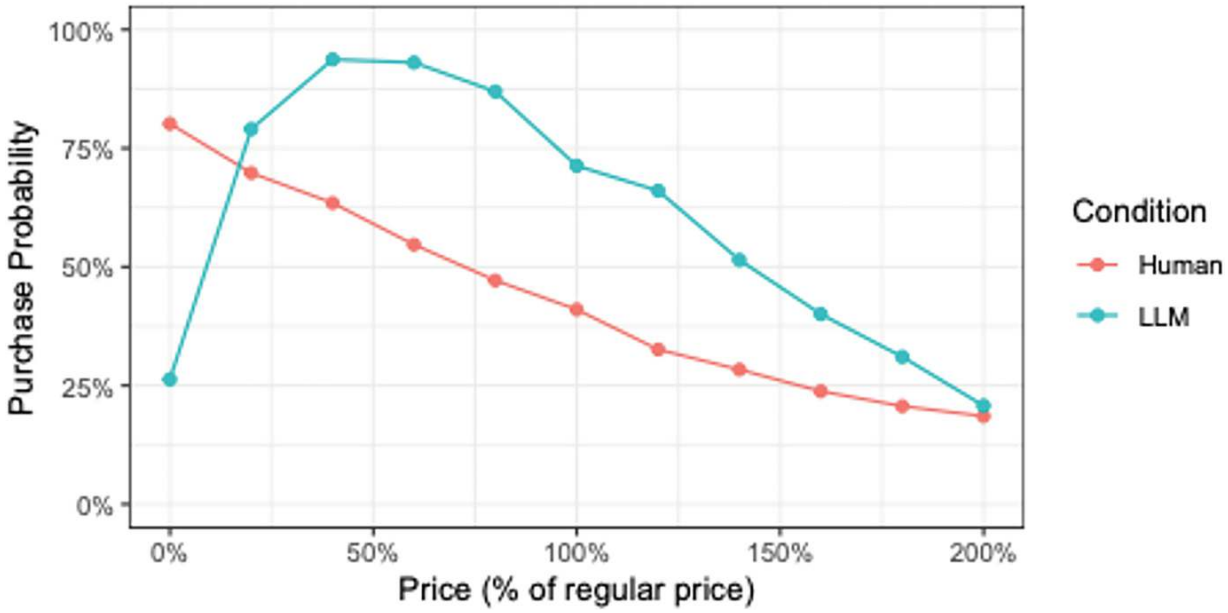

Fig. 3.16 Comparison of humans and LLMs purchase probability (Gui and Toubia 2025)

Consequently, for certain categories of highly subjective tasks, LLMs are not good enough to mimic human decision-making and preferences faithfully. These findings suggest that effectively leveraging LLMs for behavioral simulation requires a fundamental rethinking of established experimental design practices, such as moving away from traditional blinding protocols, rather than simply adapting methods developed for human subjects.

3.4.4.4 Limitations in Objective Reasoning and Factuality

Every modern LLM release is accompanied by a suite of performance metrics across multiple categories, including complex reasoning, advanced mathematics, and deep factuality. For these specialized domains, even frontier models continue to demonstrate significant headroom, indicating that these tasks remain inherently "hard" and unsolved for current AI architectures. Relying on LLM-as-a-judge for these specific frontier capabilities is particularly risky, as the model authors themselves communicate through these benchmark gaps that their systems have not yet mastered the very skills they would be required to evaluate (Fig. 3.17).

Benchmark	Description		Gemini 3 Pro	Gemini 2.5 Pro	Claude Sonnet 4.5	GPT-5.1
Humanity's Last Exam	Academic reasoning	No tools With search and code execution	**37.5%** **45.8%**	21.6% —	13.7% —	26.5% —
ARC-AGI-2	Visual reasoning puzzles	ARC Prize Verified	**31.1%**	4.9%	13.6%	17.6%
GPQA Diamond	Scientific knowledge	No tools	**91.9%**	86.4%	83.4%	88.1%
AIME 2025	Mathematics	No tools With code execution	**95.0%** **100%**	88.0% —	87.0% **100%**	94.0% —
MathArena Apex	Challenging Math Contest problems		**23.4%**	0.5%	1.6%	1.0%
MMMU-Pro	Multimodal understanding and reasoning		**81.0%**	68.0%	68.0%	76.0%
ScreenSpot-Pro	Screen understanding		**72.7%**	11.4%	36.2%	3.5%
CharXiv Reasoning	Information synthesis from complex charts		**81.4%**	69.6%	68.5%	69.5%
OmniDocBench 1.5	OCR	Overall Edit Distance, lower is better	**0.115**	0.145	0.145	0.147
Video-MMMU	Knowledge acquisition from videos		**87.6%**	83.6%	77.8%	80.4%
LiveCodeBench Pro	Competitive coding problems from Codeforces, ICPC, and IOI	Elo Rating, higher is better	**2,439**	1,775	1,418	2,243
Terminal-Bench 2.0	Agentic terminal coding	Terminus-2 agent	**54.2%**	32.6%	42.8%	47.6%
SWE-Bench Verified	Agentic coding	Single attempt	76.2%	59.6%	**77.2%**	76.3%
τ2-bench	Agentic tool use		**85.4%**	54.9%	84.7%	80.2%
Vending-Bench 2	Long-horizon agentic tasks	Net worth (mean), higher is better	**$5,478.16**	$573.64	$3,838.74	$1,473.43
FACTS Benchmark Suite	Held out internal grounding, parametric, MM, and search retrieval benchmarks		**70.5%**	63.4%	50.4%	50.8%

Fig. 3.17 Reasoning and math benchmarks with largest headroom for the most advanced models

Unfortunately, reverting to human raters for these tasks is often equally problematic. The extreme complexity of these high-level domains means that only a small, highly trained pool of subject matter experts (SMEs) can reliably perform these evaluations, further escalating costs and limiting scalability. There is an optimistic outlook, however, that as models continue to evolve and become smarter, the number of these "unsolvable" tasks will gradually decrease, eventually enabling more robust automated evaluation across all domains.

3.4.5 Advanced Techniques for Enhancing LLM Judge Robustness

To address the inherent instability and systematic biases of LLM judges, advanced architectural techniques focus on imposing structure, external scaffolding, and rigorous calibration upon the evaluation process.

3.4.5.1 Structured Prompting and Bias Mitigation

Effective LLM evaluation requires rigorous prompt engineering:

- **Bias Mitigation through Averaging**: To counter positional bias, the standard methodology involves presenting the answer choices in two different presentation orders and averaging the resulting scores. This approach has been empirically shown to alleviate the bias.
- **Rubrics and Reference Data**: For reliable evaluation, especially when using weaker evaluator models, providing explicit score descriptions and reference answers is critical. Empirical analysis suggests that providing descriptions for only the highest and lowest scores may yield the most reliable results, indicating that excessively granular rubrics for intermediate scores may not be strictly necessary.
- **Chain-of-Thought (CoT) Reasoning**: Requiring the LLM judge to articulate its reasoning process (Chain-of-Thought) before assigning a final score significantly enhances consistency and interpretability, thereby reducing various internal biases. These model-inferred thinking traces serve as a transparent proxy for human reasoning, improving the reliability and consistency of automated LLM raters, especially in subjective evaluation tasks.

3.4.5.2 Improving Context and Reasoning Capacity

Evaluation structure can be augmented to improve the LLM judge's capacity for complex assessment:

- **Reference-Guided Judging**: To resolve high failure rates in objective tasks, such as evaluating complex math questions, a technique called reference-guided judging is employed. This method involves independently generating the LLM judge's answer first, and then displaying this generated answer as a reference anchor within the judge prompt. This scaffolding provides the necessary factual constraint, resulting in a dramatic improvement in failure rate (e.g., dropping from 70% to 15% in some experiments (Zheng et al. 2023b)

- **Multi-turn Context Management**: In dialog evaluation environments like MT-bench, proper context management is mandatory. Attempting to evaluate multiple turns using separate prompts often causes the LLM judge to struggle with accurately referencing previous assistant responses. Therefore, displaying **complete conversations in a single prompt** is necessary to enable the LLM judge to fully grasp the conversational context and accurately assess multi-turn performance.

3.4.5.3 Hybrid Evaluations

The Hybrid Framework is an advanced orchestration strategy designed to maximize the economic efficiency of LLM-as-a-Judge systems while preserving the "gold standard" reliability of expert human judgment. Rather than viewing human and machine raters as mutually exclusive, this architecture treats them as a tiered system where high-volume, low-complexity tasks are automated, and "frontier" or high-uncertainty cases are escalated to subject matter experts (SMEs).

A crucial component of successful hybrid architectures is the ability of the LLM judge to estimate its own epistemic uncertainty. Instead of outputting a deterministic score, the model is prompted to predict a probability distribution over the rubric categories.

- **Tier 1**: **High Confidence.** If the LLM judge provides a verdict with a confidence score above a predefined threshold (e.g., Probability >0.95), the score is committed to the metrics pipeline automatically.
- **Tier 2**: **Uncertainty-Based Escalation.** When the judge identifies ambiguity, which is often caused by prompt complexity or edge case reasoning, the sample is routed to a human queue.
- **Tier 3**: **Meta-Judging.** To prevent "confident hallucinations" where the judge is wrong but certain, a small, random percentage (typically 5–10%) of high-confidence machine ratings must be regularly audited by humans for calibration.

The architectural focus is placed on achieving precise confidence estimation and robust meta-judging for low-confidence classifications

(Haldar and Hockenmaier 2025). If the LLM judge miscalculates its uncertainty, for instance, if it is highly confident but factually incorrect, expensive human labor is wasted re-evaluating misclassified high-confidence cases, negating the cost savings derived from automation.

Best practices for deploying such a system involve a staged approach:

- **Regression Testing**: Running automated LLM Judge sets.
- **Internal Calibration**: Comparing a subset of those results against expert "trusted" labels to study misalignments.
- **Human Verification**: A final expert sign-off on key metrics before production deployment.

3.4.6 Continuous Quality Monitoring and Rater Vetting

Regardless of whether the rater is a human expert or an advanced LLM, quality monitoring and a mechanism for continuous feedback are crucial components of a robust evaluation pipeline. Both modalities are susceptible to drift and inconsistency: human preferences can change or decline over time, while LLM outputs suffer from stochastic variance. Therefore, a rigorous system of vetting, calibration, and quality control must be applied to all evaluators.

3.4.6.1 Vetting and Calibration for Human Raters

For human evaluators, quality monitoring is often formalized under frameworks like the **Annotation Quality Framework (ACC)** (Lavitas et al. 2021), which tracks four key metrics: Accuracy, Credibility, Instant Consistency, and Longitudinal Consistency.

- **Vetting (Accuracy and Instant Consistency)**: Initial rater readiness is established by comparing early annotations against a high-quality **gold standard** set (Accuracy) and measuring inter-rater agreement using tools like **Fleiss Kappa** (Instant Consistency). These measures ensure that new raters are aligned with the established rubric and baseline standards before entering the production pool. Clear

instructions and robust training examples are paramount for minimizing variance at this stage.
- **Continuous Monitoring (Longitudinal Consistency)**: Rater quality can degrade due to fatigue, conceptual drift (applying labels more liberally or conservatively), or turnover. To detect and correct this, longitudinal quality checks: comparing initial annotations with re-annotations performed by the same rater after a time interval (e.g., 60 days) are required. Intervention protocols, such as improving guidelines or providing targeted retraining, are then deployed to address the identified issues.

3.4.6.2 Quality Control and Meta-Judging for LLM Judges

LLM judges, while exhibiting high average consistency, introduce the risk of low **intra-rater reliability**: they can assign different scores to the same input across runs due to stochasticity. Furthermore, the LLM model itself can drift over time, necessitating continuous quality checks.

- **Meta-Judging**: This is the process of **assessing the quality of the LLM's judgment itself**. For objective tasks, this involves verifying the LLM judge's precision against **objective ground truth labels**, moving beyond mere alignment with fallible human preference.
- **Continuous Calibration**: To ensure the LLM judge remains aligned and reliable, it must be regularly calibrated against small, highly trusted samples of human preference. This process ensures that the machine-rater's internal rubric has not drifted, maintaining confidence in the high-volume, automated scoring pipeline. Specialized architectures, like those using calibrated autoraters, are designed explicitly to address stochasticity by predicting probabilities rather than deterministic scores, thereby providing an inherent measure of judgment quality.

3.5 Conclusion

In Chap. 3, we have discussed four main levers of LLM evaluations: Evaluation Sets, Templates, Metrics and Evaluators.

Each component is equally critical for the success of an evaluation framework, and deep understanding and informed decisions are required for building robust evaluations.

Starting with the evaluation sets: any evaluation set is a sample from some underlying distribution and necessarily represents the traits of that population. Some sets are intentionally biased to zoom into a specific problem space (e.g., trust and safety-focused sets), while others are designed to be broad to assess characteristics across the entire distribution (e.g., sets for overall expected helpfulness).

Once a set is constructed, measurements are being applied to it. Measurements themselves consist of the template and the metric calculations. Metrics can only be as powerful and specific as the templates allow them: if a certain signal is not being collected, or collected in a way that is not trustworthy, it decapitates all the downstream metrics' usability. We recommend a hybrid approach to template constructions that includes collecting the most objective decomposed signals together with the overall subjective assessments.

Metrics themselves must be designed with statistical rigor, which includes understanding of potential biases and confidence intervals. Point estimates without confidence intervals around them have very little practical value, and we recommend that all practitioners strive to always report both point estimates and the corresponding margin of errors.

Finally, the choice of evaluators represents a critical factor in evaluation success. No estimators, either human or machines, is perfect, as all have non-zero probability of making incorrect verdicts, thus careful choice of appropriate evaluation schema is required in each case: while LLMs-as-a-judge is an excellent and practical choice for high-volume tasks, areas requiring deep domain knowledge that exceed current cutting-edge LLMs capabilities still rely on the expertise of highly trained human raters.

Ultimately, the intentional and strategic orchestration of these four levers is instrumental in constructing the trustworthy and reliable evaluations for the next generation of AI systems.

References

Aali, A., et al.: Structured prompting enables more robust, holistic evaluation of language models. arXiv preprint arXiv:2511.20836 (2025)

Alaa, A., et al.: Medical large language model benchmarks should prioritize construct validity. arXiv preprint arXiv:2503.10694 (2025)

Bai, G., et al.: Mt-bench-101: a fine-grained benchmark for evaluating large language models in multi-turn dialogues. arXiv preprint arXiv:2402.14762 (2024)

Bean, A.M., et al.: Measuring what matters: construct validity in large language model benchmarks. arXiv preprint arXiv:2511.04703 (2025)

Brown, T., et al.: Language models are few-shot learners. Adv. Neural Inf. Process. Syst. **33**, 1877–1901 (2020)

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.D.O., Kaplan, J., et al.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021)

Chollet, F.: On the measure of intelligence. arXiv preprint arXiv:1911.01547 (2019)

Dodge, J., et al.: Documenting large webtext corpora: a case study on the colossal clean crawled corpus. arXiv preprint arXiv:2104.08758 (2021)

Dubois, Y., Galambosi, B., Liang, P., Hashimoto, T.B.: Length-controlled AlpacaEval: a simple way to Debias automatic evaluators. arXiv preprint arXiv:2404.04475 (2024)

Elo, A.: The Rating of Chessplayers, Past & Present. Ishi Press (1978)

Fabbri, A.R., et al.: Summeval: re-evaluating summarization evaluation. Trans. Assoc. Comput. Linguist. **9**, 391–409 (2021) [https://doi.org/10.1162/tacl_a_00373]

Feldman, M., Lazzara, E.H., Vanderbilt, A.A., DiazGranados, D.: Rater training to support high-stakes simulation-based assessments. J. Contin. Educ. Health Prof. **32**, 279 (2012) [https://doi.org/10.1002/chp.21156][PubMed][PubMedCentral]

Gebru, T., et al.: Datasheets for datasets. Commun. ACM. **64**(12), 86–92 (2021) [https://doi.org/10.1145/3458723]

Gonzalez, M.A.A., Hernandez, M.B., Perez, M.A.P., Orozco, B.L., Soto, J.T.C., Malagon, S.: Do repetitions matter? Strengthening reliability in LLM evaluations (2025)

Gui, G., Toubia, O.: The challenge of using LLMs to simulate human behavior: a causal inference perspective. https://arxiv.org/pdf/2312.15524 (2025)

Guo, Z., Jin, R., Liu, C., Huang, Y., Shi, D., Yu, S.L., Liu, Y., Li, J., Xiong, B., Xiong, D.: Evaluating large language models: a comprehensive survey (2023)

Gururangan, S., Swayamdipta, S., Levy, O., Schwartz, R., Bowman, S.R., Smith, N.A.: Annotation artifacts in natural language inference data. In: Proceedings of NAACL-HLT (2018)

Haldar, R., Hockenmaier, J.: Rating Roulette: self-inconsistency in LLM-as-a-judge frameworks. https://arxiv.org/pdf/2510.27106 (2025)

HEIM Benchmark (2023)

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., Steinhardt, J.: Measuring massive multitask language understanding. In: Proceedings of ICLR (2021)

Hoffmann, J., et al.: Training compute-optimal large language models. arXiv preprint arXiv:2203.15556 (2022)

Hu, J., et al.: Xtreme: a massively multilingual multi-task benchmark for evaluating cross-lingual generalisation. In: International Conference on Machine Learning. PMLR (2020)

Hupkes, D., Bogoychev, N.: Multiloko: a multilingual local knowledge benchmark for llms spanning 31 languages. arXiv preprint arXiv:2504.10356 (2025)

Jescovitch, L.N., Scott, E.E., Cerchiara, J.A., Doherty, J.H., Wenderoth, M.P., Merrill, J.E., Urban-Lurain, M., Haudek, K.C.: Deconstruction of holistic rubrics into analytic rubrics for large-scale assessments of students' reasoning of complex science concepts. Pract. Assess. Res. Eval. **24** (2019). ISSN: 1531-7714

Justus, V.L., Rodrigues, V.B., dos Santos Sousa, A.R.: Bootstrap confidence intervals: a comparative simulation study (2024)

Kaiyom, F., Ahmed, A., Mai, Y., Klyman, K., Bommasani, R., Liang, P.: HELM safety: towards standardized safety evaluations of language models (2024)

Kaldaras, L., Yoshida, N.R., Haudek, K.C.: Rubric development for AI-enabled scoring of three-dimensional constructed-response assessment aligned to NGSS learning progression. Front. Educ. (2022). https://doi.org/10.3389/feduc.2022.983055

Kiela, D., et al.: Dynabench: rethinking benchmarking in NLP. In: Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (2021)

Kwan, W.-C., et al.: Mt-eval: a multi-turn capabilities evaluation benchmark for large language models. arXiv preprint arXiv:2401.16745 (2024)

Lavitas, L., Redfield, O., Lee, A., Fletcher, D., Eck, M., Janardhanan, S.: Annotation quality framework —accuracy, credibility, and consistency. https://www.datacentricai.org/neurips21/papers/49

CameraReady_DCAI2021_tex(8).pdf (2021)

Li, H., Dong, Q.: LLMs-as-judges: a comprehensive survey on LLM-based evaluation methods (2024)

Li, A.O., Goyal, T.: Memorization vs. reasoning: updating LLMs with new knowledge. arXiv preprint arXiv:2504.12523 (2025)

Li, X., et al.: HellaSwag-Pro: a large-scale bilingual benchmark for evaluating the robustness of LLMs in commonsense reasoning. arXiv preprint arXiv:2502.11393 (2025a)

Li, Y., Mohamud, J.H., Sun, C., Wu, D., Boulet, B.: Leveraging LLMs as meta-judges: a multi-agent framework for evaluating LLM judgments. https://arxiv.org/pdf/2504.17087 (2025b)

Liang, P., et al.: Holistic evaluation of language models. arXiv preprint arXiv:2211.09110 (2022)

Lin, C.-Y.: ROUGE: a package for automatic evaluation of summaries (2004)

Lin, S., Hilton, J., Evans, O.: Truthfulqa: measuring how models mimic human falsehoods. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics. Long Papers, vol. 1 (2022)

Liu, Y., Guo, Z., Liang, T., Shareghi, E., Vulić, I., Collier, N.: Measuring, evaluating and improving logical consistency in large language models (2024)

LMSYS ChatBot Arena.: https://lmarena.ai/ (n.d.)

Long, D.X., et al.: LLMs are biased towards output formats! Systematically evaluating and mitigating output format bias of LLMs. arXiv preprint arXiv:2408.08656 (2024)

Mazeika, M., Phan, L., Yin, X., Zou, A., Wang, Z., Mu, N., Sakhaee, E., Li, N., Basart, S., Li, B., Forsyth, D., Hendrycks, D.: HarmBench: a standardized evaluation framework for automated red teaming and robust refusal (2024)

McCoy, T., Pavlick, E., Linzen, T.: Right for the wrong reasons: diagnosing syntactic heuristics in natural language inference. In: Proceedings of ACL (2019)

Miller, E.: Adding error bars to Evals: a statistical approach to language model evaluations. arXiv preprint arXiv:2411.00640 (2024)

Miller, J.K.: Evaluating LLM metrics through real-world capabilities (2025)

Nakshatri, N.S., et al.: When facts change: probing LLMs on evolving knowledge with evolveQA. arXiv preprint arXiv:2510.19172 (2025)

Papineni, K., Roukos, S., Ward, T., Zhu, W.-J.: BLEU: a method for automatic evaluation of machine translation (2002)

Paritosh, P.: Human computation must be reproducible (2012)

Parrish, A., Chen, A., Nangia, N., Padmakumar, V., Phang, J., Thompson, J., Htut, P.M., Bowman, S.R.: BBQ: a hand-built bias benchmark for question answering (2022)

Polo, F.M., et al.: tinyBenchmarks: evaluating LLMs with fewer examples. arXiv preprint arXiv:2402.14992 (2024)

Ponti, E.M., et al.: XCOPA: a multilingual dataset for causal commonsense reasoning. arXiv preprint arXiv:2005.00333 (2020)

Qian, Q., et al.: Benchmark^2: systematic evaluation of LLM benchmarks. arXiv preprint arXiv:2601.03986 (2026)

Ramnath, K., Zhou, K., Guan, S., Mishra, S.S., Qi, X., Shen, Z., Wang, S., Woo, S., Jeoung, S., Wang, Y., Wang, H., Ding, H., Lu, Y., Xu, Z., Zhou, Y., Srinivasan, B., Yan, Q., Chen, Y., Ding, H., Xu, P., Cheong, L.L.: A systematic survey of automatic prompt optimization techniques (2025)

Recht, B., et al.: Do imagenet classifiers generalize to imagenet? In: International Conference on Machine Learning. PMLR (2019)

Röttger, P., Kirk, H.R., Vidgen, B., Attanasio, G., Bianchi, F., Hovy, D.: XSTest: a test suite for identifying exaggerated safety behaviours in large language models (2024)

Srivastava, A., et al.: Beyond the imitation game: quantifying and extrapolating the capabilities of language models. Transactions on Machine Learning Research (2023)

Tam, T.Y.C., Sivarajkumar, S., Kapoor, S., Stolyar, A.V., Polanska, K., McCarthy, K.R., Osterhoudt, H., Wu, X., Visweswaran, S., Fu, S., Mathur, P., Cacciamani, G.E., Sun, C., Peng, Y., Wang, Y.: A framework for human evaluation of large language models in healthcare derived from literature review. NPJ Digit. Med. **28**, 258 (2024) [https://doi.org/10.1038/s41746-024-01258-7]

Vidgen, B., Scherrer, N., Kirk, H.R., Qian, R., Kannappan, A., Hale, S.A., Röttger, P.: SimpleSafetyTests: a test suite for identifying critical safety risks in large language models (2024)

Wang, X., et al.: Self-consistency improves chain of thought reasoning in language models. arXiv preprint arXiv:2203.11171 (2022)

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., Zhou, D.: Chain-of-thought prompting elicits reasoning in large language models (2023)

Welty, C., Paritosh, P., Aroyo, L.: Metrology for AI: From benchmarks to instruments (2019)

Wu, X., Saraf, P.P., Lee, G., Latif, E., Liu, N., Zhai, X.: Unveiling scoring processes: dissecting the differences between LLMs and human graders in automatic scoring. Technol. Knowl. Learn. (2025). https://doi.org/10.1007/s10758-025-09836-8

Yang, S., et al.: Rethinking benchmark and contamination for language models with rephrased samples. arXiv preprint arXiv:2311.04850 (2023)

Yu, F.: Beyond pointwise scores: decomposed criteria-based evaluation of LLM responses (2025)

Zhang, Z., et al.: Redundancy principles for MLLMs benchmarks. arXiv preprint arXiv:2501.13953 (2025)

Zheng, L., et al.: Judging LLM-as-a-judge with MT-bench and chatbot arena. Adv. Neural Inf. Process. Syst. **36**, 46595–46623 (2023a) [https://doi.org/10.52202/075280-2020]

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E.P., Zhang, H., Gonzalez, J.E., Stoica, I.: Judging LLM-as-a-judge with MT-Bench and chatbot arena. https://arxiv.org/pdf/2306.05685 (2023b)

Zheng, X., et al.: Cheating automatic LLM benchmarks: null models achieve high win rates. arXiv preprint arXiv:2410.07137 (2024)

Zhou, H., et al.: Lost in benchmarks? Rethinking large language model benchmarking with item response theory. arXiv preprint arXiv:2505.15055 (2025)

A. Robsky et al., *Reliable Evaluations for LLMs and AI Agents*
https://doi.org/10.1007/978-3-032-26749-8_4

4. Evaluating AI Agents

Alexei Robsky[1] ✉, Liliya Lavitas[2] and Yueqing Wang[3]
(1) Microsoft, Sammamish, WA, USA
(2) Google (United States), Newton, MA, USA
(3) Microsoft, San Francisco, CA, USA

4.1 What Makes an Agent an Agent

Before establishing evaluation frameworks, we must define what distinguishes an AI agent from a standard large language model. The term "agent" has become ubiquitous in industry discourse, and correspondingly imprecise. Marketing materials label everything from simple chatbots to complex autonomous systems as "agents," creating confusion about what capabilities actually require specialized evaluation approaches. A rigorous definition is essential: it determines which evaluation methods apply, what metrics matter, and when the frameworks developed in Chap. 3 need fundamental adaptation rather than minor extension.

The stakes of this definitional work are practical, not merely academic. An organization deploying a retrieval-augmented generation (RAG) system that always follows the same retrieve-then-generate sequence faces different evaluation challenges than one deploying a system that autonomously decides whether to search, calculate, browse the web, or respond directly based on each query. The first is a workflow with predictable behavior; the second is an agent with emergent behavior. Conflating them leads to either over-engineering evaluation for simple systems or, more dangerously, under-evaluating complex ones.

4.1.1 Defining Agency: From Chatbots to Autonomous Systems

At its core, an agent is a system that perceives its environment, reasons about goals, takes actions, and observes the results of those actions in an iterative loop. Unlike a standard LLM that produces a single response to a single prompt, an agent operates across time, maintaining state and adapting behavior based on feedback from its environment. This temporal dimension, the fact that agents act, observe consequences, and act again, fundamentally changes what evaluation must capture (Fig. 4.1).

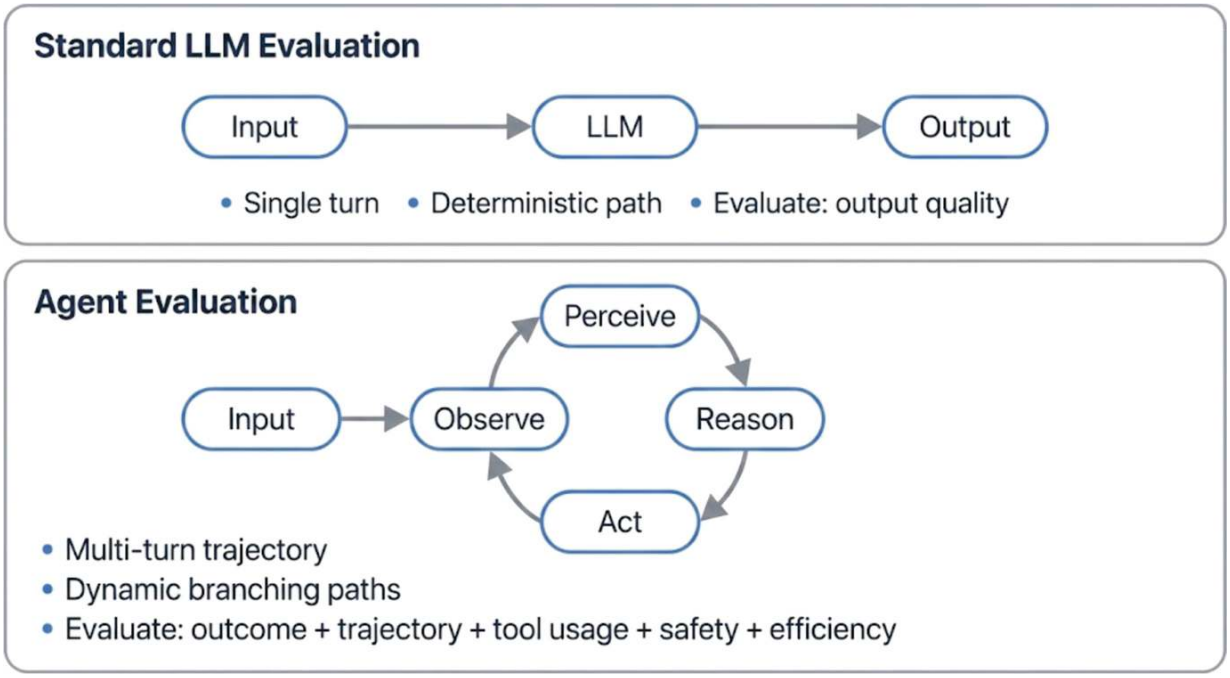

Fig. 4.1 Standard LLM vs agent evaluation

Anthropic's framework for building effective agents draws a critical architectural distinction that clarifies this boundary (Anthropic 2024). As they define it: "Workflows are systems where LLMs and tools are orchestrated through predefined code paths. Agents, on the other hand, are systems where LLMs dynamically direct their own processes and tool usage, maintaining control over how they accomplish tasks." This distinction hinges on a single question: *who controls the execution flow?*

In a workflow, the developer predetermines each step. The system might incorporate an LLM at various points (to classify input, generate a response, or summarize results) but the sequence of operations is fixed in code.

An example of a customer service workflow:

1. Classify the inquiry
2. Retrieve relevant documentation
3. Generate a response
4. Log the interaction

The LLM contributes to each step, but the workflow engine dictates the order. Change the customer's question, and the same four steps execute in the same sequence.

In an agent, the model itself decides what action to take next based on observations. Given a customer inquiry, an agent might determine that it first needs to check account status, then search the knowledge base, then (upon finding the answer incomplete) escalate to a human, or perhaps recognize that the question is simple enough to answer directly without any tool calls at all. The execution path emerges from the model's reasoning, not from predetermined code. This autonomy is both the source of agent capability and the root of evaluation complexity.

OpenAI's characterization aligns with this view. In their announcement of tools for building agents, they state: "We view agents as systems that independently accomplish tasks on behalf of users" (OpenAI 2025a). Their practical guide to building agents elaborates that an agent "leverages an LLM to manage workflow execution and make decisions. It recognizes when a workflow is complete and can proactively correct its actions if needed" (OpenAI 2025a). The key capabilities (managing execution, making decisions, recognizing completion, correcting itself) all point to the model exercising judgment rather than following a script.

Google DeepMind's research offers an even more formal perspective, proposing "the first formal causal definition of agents" (Kenton et al. 2023). Their key insight: "agents are systems that would adapt their policy if their actions influenced the world in a different way." This counterfactual framing is elegant: an agent is precisely something that would behave differently if the consequences of its actions changed. A workflow, by contrast, executes its predetermined path regardless of downstream effects.

Despite these authoritative definitions, the field lacks complete consensus. A comprehensive survey on LLM agents defines AI agents as "artificial entities that sense their environment, make decisions, and take actions," noting that while many efforts have been made to develop intelligent agents, the community has historically lacked "a general and powerful model to serve as a starting point for designing AI agents that can adapt to diverse scenarios" (Xi et al. 2023).

The practical implication for evaluation is significant. When assessing a workflow, you can test each predetermined step independently. You know what the system will do; you verify that it does it correctly. When assessing an agent, you cannot predict the execution path. The same input may trigger different

sequences of actions across runs. Evaluation must therefore focus not just on final outcomes but on the quality of the dynamic decision-making process itself.

Consider another concrete example: A travel booking system. A workflow implementation might always (Fig. 4.2)

1. Extract travel dates and destination
2. Search flights
3. Search hotels
4. Present options
5. Book upon confirmation

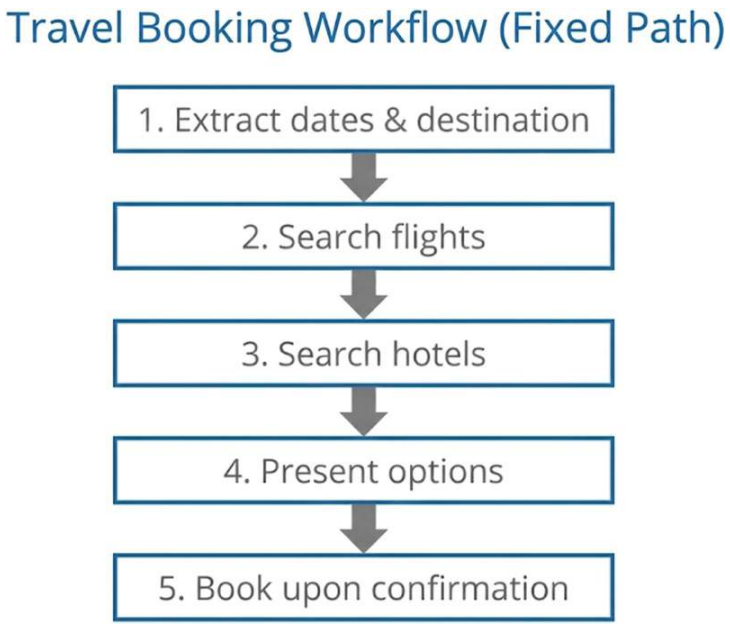

Fig. 4.2 Travel booking workflow

Testing this system means verifying each step functions correctly and integrates properly with the next, a tractable engineering problem.

An agent implementation receives the same travel request but reasons about how to proceed. It might recognize that the user mentioned “a quiet hotel near the conference center” and decide to first search for the conference location before searching hotels. It might notice that flights on the requested dates are expensive and proactively check adjacent dates. It might determine that the

user's budget constraint, mentioned casually, rules out business class and skip presenting those options entirely. Each decision represents the model exercising judgment (Fig. 4.3).

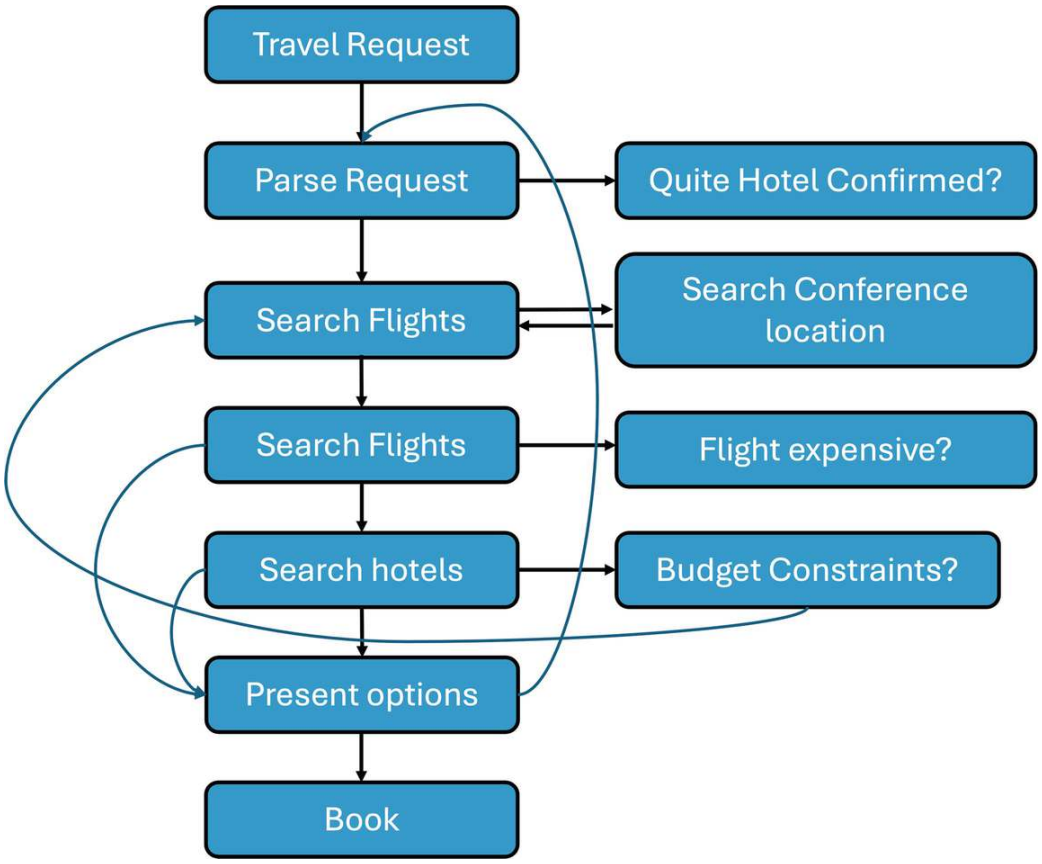

Fig. 4.3 Travel booking agent (dynamic path)

Evaluating this agent requires assessing not just whether it successfully books appropriate travel, but whether its chain of decisions was sensible. Did it appropriately prioritize the user's stated and implied preferences? Did it gather necessary information before acting? Did it avoid unnecessary steps? Did it handle ambiguity reasonably? These questions, although not the full list of questions that can be asked, have no analog in workflow evaluation because workflows don't make decisions; they execute instructions.

The distinction also surfaces in failure modes. Workflows fail in predictable ways: a step doesn't execute, an integration breaks, data doesn't flow correctly. The failure is localized and typically reproducible. Agents fail in emergent ways: the model makes a poor decision that cascades into subsequent poor decisions, or it interprets ambiguous instructions in an unexpected way, or it gets stuck in a loop of ineffective actions. These failures are often non-deterministic and

context dependent, requiring evaluation approaches that can detect patterns across many runs rather than verifying correctness on individual cases.

This is why the evaluation frameworks from Chap. 3 (sets, templates, metrics, and raters) require fundamental adaptation for agents. Evaluation sets must capture not just input/output pairs but multi-step scenarios with branching possibilities. Templates must score trajectories, not just final responses. Raters, whether human or LLM judges, must assess reasoning quality and decision-making, not just output correctness. The remainder of this chapter develops these adaptations systematically.

But first, we must understand the internal mechanics that enable agency: the perception, reasoning, action, observation loop that defines how agents operate moment to moment. This loop is the unit of agent behavior, and understanding it is prerequisite to evaluating it.

4.1.2 The Agent Loop: Perception, Reasoning, Action, Observation

The canonical agent architecture follows a cyclical pattern that repeats until the task is complete or a termination condition is met. This cycle, sometimes called the "agentic loop," comprises four phases that mirror how humans approach complex tasks: perceive the current situation, reason about what to do, take action, and observe the results.

Perception is the agent's intake of information and its environment. At each step, the agent receives input that may include the original user query, the current state of the environment, results from previous actions, error messages, or new information retrieved from external sources. This input forms the agent's "observation" of the world at that moment. In practice, perception often means constructing a prompt that aggregates relevant context: the task description, conversation history, tool outputs, and any other state the agent needs to make its next decision.

Reasoning is where the agent processes its observations and decides what to do next. This phase leverages the LLM's core capability: given context, generate a sensible continuation. The agent might decompose a complex goal into subgoals, recall relevant knowledge, compare alternative approaches, or simply determine that it has enough information to produce a final answer. The quality of this reasoning phase largely determines agent effectiveness.

Action is the agent's intervention in the world. Actions might include calling a tool (searching the web, querying a database, executing code), generating text for the user, modifying a file, or signaling that the task is complete. Crucially,

actions have consequences: they change the state of the environment or produce new information that the agent must incorporate.

Observation closes the loop. After an action executes, the agent receives feedback: the tool's output, an error message, confirmation of success, or updated environmental state. This observation becomes part of the agent's context for the next cycle, enabling it to adapt its approach based on what actually happened rather than what it expected to happen (Fig. 4.4).

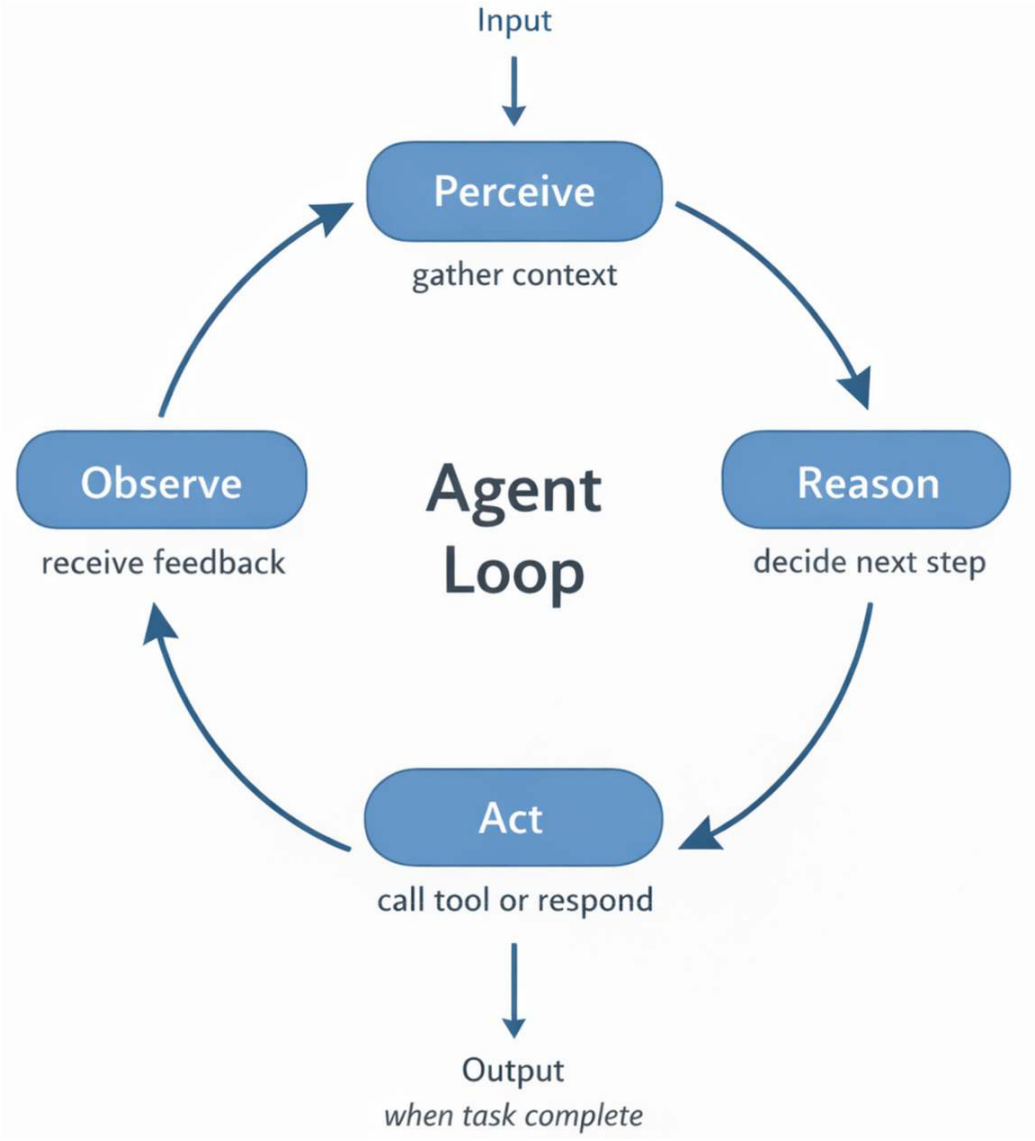

Fig. 4.4 The agent loop

Each cycle through this loop represents one "step" in the agent's trajectory. Simple tasks might require only one or two steps; complex tasks might require

dozens. The number of steps, the quality of reasoning at each step, and the appropriateness of actions all become targets for evaluation.

The agent loop is the abstract architecture; different implementation techniques realize it in different ways. The most influential of these is ReAct (Reasoning + Acting), introduced by Yao et al. (2022a), which makes the reasoning phase of the loop visible by requiring the model to produce an explicit "Thought" trace before each action.

Prior to ReAct, the LLM research community had studied reasoning and acting as separate capabilities. Chain-of-thought prompting showed that models could improve their answers by producing intermediate reasoning steps, but these reasoning traces existed in isolation, disconnected from any external environment. Separately, action generation approaches had models produce tool calls or navigation commands, but without any visible deliberation about why a particular action was chosen. Yao et al. demonstrated that combining the two creates powerful synergies: "reasoning traces help the model induce, track, and update action plans as well as handle exceptions, while actions allow it to interface with external sources, such as knowledge bases or environments, to gather additional information" (Yao et al. 2022a).

The practical structure of a ReAct trace illustrates the pattern:

- **Thought**: I need to find when the Eiffel Tower was built. Let me search for this information.
- **Action**: Search ("Eiffel Tower construction date").
- **Observation**: The Eiffel Tower was constructed from 1887 to 1889 as the centerpiece of the 1889 World's Fair.
- **Thought**: The search results indicate construction was completed in 1889. I now have the information needed to answer the user's question.
- **Action**: Respond ("The Eiffel Tower was completed in 1889.")

Each cycle through this loop represents one "step" in the agent's trajectory. Simple tasks might require only one or two steps; complex tasks might require dozens. The number of steps, the quality of reasoning at each step, and the appropriateness of actions all become targets for evaluation.

Several properties of this loop have direct implications for evaluation design:

Loop property	Evaluation implication
Temporal dependencies	Assess error propagation and recovery
Stochasticity	Run multiple trials; report distributions

Loop property	Evaluation implication
Environment coupling	Specify and control environment
Unbounded execution	Detect stuck states and looping
Explicit vs implicit reasoning	Adapt granularity to what's visible

1. The loop creates temporal dependencies. Each observation depends on the previous action, and each action depends on the accumulated reasoning from all prior steps. An error early in the trajectory can cascade through subsequent steps, compounding into task failure. Evaluation must therefore consider not just whether individual steps are correct, but how errors propagate and whether the agent can recover.
2. The loop introduces stochasticity. Given identical initial conditions, an agent may take different paths through the loop on different runs. The LLM's sampling temperature, the variability of external tool responses, and the sensitivity of reasoning to small prompt variations all contribute to this non-determinism. Evaluation must account for this by running multiple trials and reporting distributional metrics rather than single point estimates.
3. The loop couples the agent to its environment. Unlike a pure language model that operates on text in isolation, an agent's behavior depends on what tools are available, how those tools respond, and what state the environment is in. Evaluation must therefore specify and control the environment, not just the input prompt.
4. The loop can run indefinitely. Without appropriate termination conditions, an agent might continue cycling forever, consuming resources without making progress. Detecting when an agent is stuck, looping, or failing to converge is itself an evaluation challenge.

These properties are individually well-studied in sequential decision-making, control theory, and reinforcement learning. What follows connects them specifically to the evaluation challenges they create for LLM-based agents.

Understanding the agent loop also clarifies what we mean by "trajectory," a term that will recur throughout this chapter. A trajectory is the complete sequence of steps an agent takes from receiving a task to completing it (or failing): all the thoughts, actions, and observations concatenated in order. Trajectory evaluation assesses the quality of this entire sequence, not just its

endpoint. Two agents might both successfully complete a task, but one might do so in three efficient steps while the other takes 15 redundant ones. Trajectory evaluation captures this difference.

The loop also explains why agent evaluation cannot simply reuse LLM evaluation methods. When evaluating an LLM, you provide an input and assess the output. When evaluating an agent, we must assess a dynamic process: how the agent responded to each observation, whether its actions were appropriate given its reasoning, whether it used tools effectively, whether it recovered from errors, and whether the overall trajectory was efficient and safe. This requires fundamentally different evaluation infrastructure, metrics, and methodologies, which the remainder of this chapter develops.

4.2 Sets: Evaluation Data for Agents

Chapter 3 established that evaluation sets are the foundation of any LLM evaluation program: without representative data, even sophisticated metrics and expert judges produce misleading results. The same principle holds for agent evaluation, but the structure of that data must change fundamentally.

An LLM evaluation set is a collection of input/output pairs. Each test case provides a prompt and either a reference answer or a scoring rubric. The evaluator sends the prompt to the model, collects the response, and scores it. The entire process is stateless: one test case has no effect on the next.

Agent evaluation sets cannot work this way. An agent does not simply produce an output; it executes a process that unfolds over multiple steps, interacts with tools and environments, and modifies external state. A test case for an agent must therefore specify not just what to ask, but what world the agent operates in, what tools it has access to, what success looks like (potentially through multiple valid paths), and how to verify the outcome through environment inspection rather than output comparison. This section develops the principles and practices for constructing evaluation sets that meet these requirements.

4.2.1 Anatomy of an Agent Test Case

A single-turn LLM test case has two components: an input and an expected output. An agent test case requires substantially more structure. Examining how leading agent benchmarks structure their evaluations reveals a common pattern. Each test case, whether explicitly or implicitly, specifies at minimum five elements:

1. **A task description** defines what the agent is asked to accomplish. Unlike LLM prompts that request a single response, agent task descriptions often involve compound goals: "Find all orders placed by this customer in the last 30 days, identify which ones are eligible for return, and process returns for the eligible items." The task description should be precise enough that two domain experts would independently agree on whether the agent succeeded or failed.
2. **An initial environment state** defines the world the agent will operate in at the start of the task. For a coding agent, this might be a repository at a specific commit with a known set of failing tests. For a customer service agent, it might be a database containing specific customer records, order histories, and policy documents. For a web navigation agent, it might be a set of web pages with known content. The initial state must be fully specified and reproducible; if the environment differs between evaluation runs, results become incomparable.
3. **Available tools and their configurations** define what the agent can do. The same agent might behave very differently depending on whether it has access to a search tool, a code execution environment, or a database query interface. The evaluation set must specify not just which tools are available but how they behave, including any simulated responses, error conditions, or latency characteristics.
4. **Success criteria** define what constitutes task completion. For some tasks, success is binary: the agent either resolved the GitHub issue (all tests pass) or it did not. For others, success is graded: the agent booked a flight that met three of the customer's four stated preferences. Success criteria should be verifiable through environment inspection rather than relying on the agent's self-reported claims. As the τ-bench benchmark, introduced by Yao et al. 2024, demonstrates, comparing the database state at the end of a conversation against an annotated goal state provides faithful evaluation that does not depend on the agent's trajectory (Yao et al. 2024).
5. **Reference solutions or trajectories** (optional but valuable) provide example paths to success. These are not the only valid approaches, and evaluation should not penalize agents for finding alternative solutions. However, reference trajectories serve two purposes: they verify that the task is solvable, and they provide baselines for trajectory efficiency comparisons.

This five element structure represents a significant increase in test case complexity compared to LLM evaluation. A single agent test case may require setting up a Docker container, populating a database, configuring a simulated API, and writing verification scripts. The cost per test case is correspondingly higher, which means agent evaluation sets tend to be smaller but more carefully constructed than LLM evaluation sets.

4.2.2 Sourcing Tasks from Real Failures

The most valuable test cases come from actual failures. OpenAI's guidance on building agents recommends starting with a small number of carefully chosen cases: begin with a handful of representative examples that capture the most important scenarios, then expand coverage as patterns emerge (OpenAI 2025b). In practice, 20–50 well-constructed tasks provide enough signal for early development because effect sizes in agent evaluation tend to be large: a meaningful architectural change often swings success rates by 10–30 percentage points.

Three sources consistently produce high-quality tasks:

1. **Manual testing during development** surfaces the failures that developers discover while building the agent. When a developer notices the agent mishandling a specific scenario, that scenario should immediately become a test case. These early, organically discovered failures often reveal fundamental design issues rather than edge cases.
2. **Bug trackers and support queues** capture the failures that users encounter in production. Each user reported issue represents a real scenario where the agent fell short. Converting these into reproducible test cases creates evaluation sets that directly measure whether the agent handles the situations that actually matter to users.
3. **Behaviors the agent struggles with** can be identified through systematic probing. If the agent handles simple refund requests but fails on refunds involving partial returns, expired promotions, or multi-item orders, each of those failure patterns should become a test case. The goal is to build a set that targets known weaknesses, not one that merely confirms known strengths.

A well-constructed agent evaluation set maintains balance along two dimensions. First, it should include both positive and negative cases: scenarios where the agent should act and scenarios where it should refuse, escalate, or request clarification. Testing only the happy path leads to agents that are eager to act but poor at recognizing when they should not. Second, it should include both capability evaluations (tasks the agent currently fails, targeting improvement) and regression evaluations (tasks the agent currently passes, detecting backsliding). OpenAI's guidance emphasizes treating these differently: capability evaluations track progress toward harder goals while regression evaluations provide early warning when changes break existing behavior (OpenAI 2025b).

4.2.3 Specifying the Environment

The environment specification is what makes agent evaluation sets categorically different from LLM evaluation sets. An LLM test case is self-contained: the prompt carries all the information the model needs. An agent test case depends on an external world that must be set up, maintained, and inspected.

Environment specifications fall on a spectrum from lightweight to heavyweight:

1. **Mock environments** replace real tools with deterministic simulators that return predefined responses. When the agent calls a search API, the mock returns a fixed set of results. When it queries a database, the mock returns a fixed record. Mocks are fast, cheap, and fully reproducible, but they cannot capture the full complexity of real tool interactions. An agent that performs well against mocks may struggle with the messy, unpredictable responses of real APIs.
2. **Sandboxed environments** provide real tools running in isolated containers. SWE-bench, for example, evaluates coding agents by giving them a real codebase in a Docker container: the agent can read files, execute code, run tests, and generate patches, all within a controlled but authentic environment (Jimenez et al. 2024). WebArena takes a similar approach for web navigation, providing self-hosted clones of real websites including e-commerce platforms, forums, and content management systems (Zhou et al. 2023). Sandboxed environments balance realism with reproducibility though they are substantially more expensive to build and maintain than mocks.

   Live environments test agents against real, production systems. This

3. **Live environments** test agents against real, production systems. This provides the highest fidelity but the lowest reproducibility: external APIs change their behavior, databases evolve, and web content shifts over time. Live evaluation is essential for validating that agents work in the real world, but it cannot serve as the primary evaluation methodology because results are not comparable across runs.

The choice of environment type depends on what you are evaluating. During rapid development, mock environments enable fast iteration. For release gating, sandboxed environments provide confidence in real-world behavior. For ongoing production monitoring, live environment checks detect drift that sandboxed tests would miss.

Regardless of type, every environment specification must guarantee **state isolation** between test cases. If one test case modifies the database, the next test case must start with a clean state. Without isolation, test cases interact in unpredictable ways: a test might pass when run in isolation but fail when run after another test that left the environment in an unexpected state. Docker containers, database snapshots, and filesystem restoration provide mechanisms for achieving this isolation, but they must be explicitly implemented. τ-bench enforces this by resetting database state between trials and using goal state comparison for verification rather than trajectory matching (Yao et al. 2024).

4.2.4 Accounting for Non-determinism

LLM evaluation can often get away with single-run assessments: set the temperature to zero, run each test case once, and score the results. Agent evaluation cannot. Even with temperature zero, agents exhibit meaningful behavioral variation across runs because small differences in tool responses, timing, and early reasoning steps compound through the trajectory.

This means that each test case must be run multiple times, and the evaluation set design must account for this. If a test case takes 5 min of agent execution time and costs $0.50 in API calls, running it ten times for statistical reliability turns a 50 task evaluation set into a 500 run program costing $250 per evaluation cycle. These economics shape practical decisions about set size, trial count, and evaluation frequency.

Two principles guide the design:

1. **Fewer tasks with more trials beats more tasks with fewer trials** for measuring reliability. A set of 30 tasks each run 10 times provides much stronger statistical signal about agent consistency than 300 tasks each run once. The τ-bench benchmark makes this trade-off explicit, using a small set

of high-quality tasks with multiple trials per task rather than a large set evaluated once (Yao et al. 2024). The pass^k metric they introduced captures exactly this dimension: it measures whether the agent succeeds on all k trials, revealing consistency that single-run evaluation would miss entirely.

2. **Isolated trials are essential.** Each trial must start from a clean environment state with no shared artifacts from previous runs. If trial three benefits from a file that trial two created, the trials are not independent, and statistical aggregation becomes invalid. This isolation requirement reinforces the need for environment specifications that support rapid state reset.

4.3 Templates: Designing Agent Evaluation Rubrics

In Chap. 3, we defined templates as the scoring rubrics that translate raw evaluation data into actionable judgments. For single-turn LLM evaluation, a template might ask: "Rate this response for factual accuracy on a scale of 1 to 5" or "Does this summary contain all key points from the source document?" The template encodes what quality means for a given task and provides the structure that evaluators (whether human or automated) use to produce consistent scores.

Agent evaluation templates must address a fundamentally different question. Rather than scoring a single output, they must score a process: a sequence of decisions, tool calls, reasoning steps, and environmental interactions that may span dozens of turns. The template must specify not only what to measure but at what granularity (individual steps, subsequences, or the full trajectory), against what reference (a specific path, a set of acceptable paths, or an outcome regardless of path), and with what scoring logic (binary, graded, or weighted across multiple dimensions).

This section develops the principles for designing agent evaluation templates that produce reliable, diagnostic, and actionable scores.

4.3.1 Grade Outcomes, Not Paths

The single most important principle in agent template design is to evaluate what the agent achieved, not the specific route it took. Agents frequently discover valid approaches that template designers did not anticipate. A customer service agent might resolve a billing dispute by calling APIs in a different order than expected, or a coding agent might fix a bug with an

approach that differs entirely from the reference solution. Penalizing these alternative paths produces false negatives that undermine confidence in evaluation results.

This principle manifests differently depending on what is being evaluated:

- **For task completion**, the template should verify the end state of the environment rather than the sequence of actions that produced it. Did the customer's order actually get refunded? Did the generated code pass the test suite? Did the database record get updated correctly?

  τ-bench implements this by comparing the database state at the end of a conversation against an annotated goal state, making the evaluation entirely independent of the specific trajectory the agent followed (Yao et al. 2024).
- **For tool usage**, the template should check whether the right tools were called with correct parameters, but should be flexible about ordering.

  Google's Vertex AI Gen AI evaluation service (Google Cloud, 2025) formalizes this with three levels of trajectory matching:

  – Exact match: Identical tool calls in identical order.
  – In order match: All required tools present in the correct order, extra tools permitted.
  – Any order match: All required tools present regardless of order.

  In practice, any order match with parameter validation is the right starting point for most evaluations. Exact match is too brittle; it rejects valid alternative paths. In order match is appropriate when the sequence genuinely matters (e.g., you must read a file before editing it).
- **For reasoning quality**, the template should assess whether the agent's decisions were sensible given the information available at each step, not whether they match a predetermined reasoning chain. This is inherently more subjective and typically requires model-based evaluation rather than deterministic checks.

The exception to this principle arises in *safety evaluation*. When an agent should not take a particular action (e.g., deleting a production database, sending unsolicited emails, exposing personal information), the template must check the trajectory for prohibited actions regardless of whether the final outcome is correct. An agent that achieves the right result through dangerous intermediate steps is not safe.

4.3.2 Dimensions to Include in Agent Rubrics

Agent evaluation templates must specify which dimensions to score. Unlike single-turn LLM evaluation where "response quality" might suffice as a single dimension, agents operate across multiple axes that require separate treatment. A well-designed agent rubric typically includes dimensions from several categories:

1. **Outcome dimensions** assess whether the agent achieved its goal. Did the task get completed? Is the environment in the correct final state? For compound tasks, what fraction of subgoals was accomplished? These dimensions are the agent equivalent of "correctness" in LLM evaluation, but verifying them requires inspecting the environment rather than comparing text outputs.
2. **Process dimensions** assess how the agent reached (or failed to reach) its goal. Was the trajectory efficient, or did it include unnecessary steps? Did the agent recover appropriately from errors? Did intermediate reasoning demonstrate understanding of the problem? Process dimensions capture quality aspects that outcome evaluation alone would miss: two agents might both succeed, but one through a direct path and the other through a wasteful, fragile sequence of lucky corrections.
3. **Tool usage dimensions** assess the agent's interaction with its available tools. Did it select appropriate tools for each situation? Did it invoke them with correct parameters? Did it interpret their outputs accurately? Tool usage can be evaluated independently of outcome because an agent might succeed despite poor tool choices (e.g., through redundant calls or lucky guesses) or fail despite correct tool usage (due to environmental factors beyond its control).
4. **Safety dimensions** assess whether the agent operated within acceptable boundaries. Did it avoid prohibited actions? Did it handle sensitive data appropriately? Did it respect rate limits, access controls, and other constraints? Safety is typically a gating dimension: any violation fails the entire evaluation regardless of other scores.
5. **Communication dimensions** (for user facing agents) assess the quality of the agent's interaction with the user. Was its language clear and appropriate? Did it keep the user informed of what it was doing? Did it ask for clarification when needed rather than making assumptions?

The template should specify scoring criteria for each dimension: what constitutes a passing score, what differentiates levels of quality, and what evidence the evaluator should examine. Section 4.4 develops specific metrics for each of these dimension categories.

4.3.3 Isolating Rubric Dimensions

A common mistake in template design is asking a single evaluator to produce a comprehensive judgment across multiple dimensions simultaneously. "Rate this agent trajectory for overall quality considering accuracy, safety, efficiency, and communication style" conflates four independent dimensions into one score, making it impossible to diagnose which aspects need improvement.

Effective agent templates isolate each evaluation dimension into a separate rubric item. Each item should have its own scoring criteria and its own weight in the final score. This isolation provides two primary benefits:

- **Diagnostic specificity**. When an agent's score drops, isolated dimensions reveal whether the problem is accuracy, safety, efficiency, or something else. A combined score that drops from 3.8 to 3.2 provides almost no diagnostic signal; knowing that accuracy held steady while efficiency dropped sharply tells you exactly where to focus improvement efforts.
- **Independent weighting**. Not all dimensions matter equally for every use case. A customer facing agent might weight communication quality heavily while an internal automation agent might weight efficiency. Isolated dimensions enable per deployment weight customization without redesigning the entire template.

For a customer service agent, an isolated rubric might include the following dimensions:

- Task completion: Did the agent resolve the customer's request? Verified by comparing database state against expected outcome. Scored binary: pass or fail.
- Policy compliance: Did the agent follow domain-specific rules? Concrete constraints (e.g., refund amounts within limits, required disclosures provided) plus softer compliance (e.g., appropriate escalation behavior, proper handling of ambiguous situations). Scored on a scale.
- Tool usage correctness: Did the agent call the right APIs with correct parameters? Did it avoid unnecessary or redundant tool calls? Scored as a ratio of correct tool invocations to total invocations.

- Efficiency: How many steps, tokens, and API calls did the agent use? Compared against a baseline or reference trajectory length. Scored as a ratio or on a scale.
- Communication quality: Was the agent's language clear, professional, and helpful? Did it keep the customer informed of actions being taken? Scored on a scale with dimension-specific criteria.
- Safety: Did the agent avoid prohibited actions? Did it attempt any unsafe intermediate steps, even if the final outcome was acceptable? Scored binary: pass or fail, with any failure overriding other scores.

4.3.4 Partial Credit and Hierarchical Scoring

Binary pass/fail scoring is appropriate for some evaluation dimensions (safety, policy compliance) but overly coarse for others. An agent that completes four of five subtasks demonstrates meaningfully different capability than one that fails immediately, and evaluation templates should capture this distinction.

One approach is *subgoal decomposition*, which breaks compound tasks into independently scorable components. If the task is "find the customer's order, check return eligibility, process the return, and send a confirmation email," each step becomes a scorable subgoal. The agent receives credit proportional to the number of subgoals completed: 4/4, 3/4, 2/4, and so on. This approach, sometimes called Task Goal Completion, provides the granular diagnostic signal that binary scoring lacks.

For even finer granularity, *step success rate* tracks the percentage of individual plan steps that the agent executes correctly. This is particularly useful for long-horizon tasks where the agent might succeed on the first 10 steps but fail on step 11. Knowing the step level breakdown reveals exactly where the agent's capability boundary lies.

When combining scores across dimensions, *weighted scoring* assigns different importance to different rubric dimensions. The simplest approach requires all dimensions to meet minimum thresholds (every dimension must score above 0.5). A more nuanced approach combines dimension scores with weights (task completion worth 40%, policy compliance 25%, efficiency 15%, communication 10%, safety 10%). Hybrid approaches apply binary gating on critical dimensions (any safety failure means overall failure) while weighting noncritical dimensions on a continuous scale.

Traffic-based weighting adjusts dimension weights according to how frequently each scenario occurs in production: if 60% of real user requests involve order status queries and only 5% involve complex returns, the

evaluation should weight performance on status queries correspondingly higher to reflect actual user impact.

The choice of scoring method depends on the deployment context. For safety critical applications, binary gating on safety dimensions is essential: no amount of task completion excellence compensates for a safety violation. For optimization-focused applications, weighted scoring enables meaningful differentiation between agents that achieve similar outcomes through more or less efficient paths.

4.4 Agent Evaluation Metrics

Section 4.3 developed the principles for designing evaluation rubrics: what dimensions to assess, how to structure scoring criteria, and how to handle partial credit. This section catalogs the specific metrics that operationalize those rubrics, providing the formulas and measurement approaches that translate agent behavior into quantitative scores.

No single metric captures agent quality. An agent might achieve high task success while being inefficient, or demonstrate excellent tool usage while failing on safety. A comprehensive evaluation framework requires metrics spanning multiple dimensions: task completion, reliability across trials, tool usage accuracy, trajectory efficiency, and error recovery. The following subsections develop metrics for each dimension.

4.4.1 Task Completion Metrics

The foundational metric is **task success rate**: the fraction of tasks where the agent achieves the defined goal.

$$\text{Task Success Rate} = (\text{Successful Tasks})/(\text{Total Tasks})$$

This metric is simple, interpretable, and universally applicable. Its limitation is that binary success/failure obscures meaningful distinctions. An agent that completes 80% of the required steps before failing provides more value than one that fails immediately, but both receive the same score of zero.

Progress rate addresses this limitation by measuring how effectively an agent advances toward its goal even when final success eludes it. The AgentBoard benchmark introduced this metric to provide diagnostic signal when most models achieve low success rates (Ma et al. 2024). If an agent completes steps 1–4 of a 5 step task before failing, its progress rate is 0.8 even though its task success is 0.

Task goal completion (TGC) applies similar logic to multi-component goals. If a task requires the agent to accomplish five independent subgoals (e.g., update the customer record, issue the refund, send the confirmation email, log the interaction, close the ticket), TGC measures the fraction achieved:

$$\text{Task Goal Completion} = (\text{Subgoals Achieved})/(\text{Total Subgoals})$$

An agent that completes 4 of 5 subgoals scores 0.8, meaningfully distinguishing it from one that completes 1 of 5.

Step success rate provides even finer granularity by tracking the percentage of individual plan steps executed correctly. This is particularly useful for long-horizon tasks where understanding exactly which step failed guides debugging and improvement.

4.4.2 Reliability Metrics

Because agents behave non-deterministically across runs (as discussed in Sect. 4.2.4), single trial metrics like task success rate provide incomplete pictures. Reliability metrics capture consistency across multiple trials of the same task.

Pass@*k* measures the probability that at least one of *k* independent trials succeeds. It answers the question: can this agent ever solve this task?

$$\text{Pass@}k = 1 - (1 - p)k \text{ (where } p \text{ is the single trial success probability)}$$

In practice, pass@*k* is estimated empirically by running *k* trials and checking whether any succeeded. A pass@1 of 0.3 combined with a pass@10 of 0.95 tells us the agent can almost certainly solve the task given enough attempts, but fails more often than it succeeds on any individual try.

Pass^*k* (sometimes written pass_*k* or "all@*k*") measures the probability that all *k* trials succeed. It answers the question: can this agent reliably solve this task?

$$\widehat{\text{Pass}}\, k = \hat{p}k \text{ (where } p \text{ is the single trial success probability)}$$

The τ-bench benchmark introduced this metric to expose reliability problems that pass@*k* obscures (Yao et al. 2024).

For example: an agent with pass@1 of 0.7 has pass^5 of only 0.17 (=0.7^5), meaning it fails at least once in five consecutive attempts 83% (1 – 0.17 = 0.83) of the time.

For customer facing applications where each user experiences a single trial, pass^*k* at modest values of *k* (5, 8, 10) reveals whether the agent is production ready. τ-bench found that even GPT-4o achieved pass^8 below 25% on retail tasks, highlighting the gap between "sometimes succeeds" and "reliably succeeds."

Mean score across trials captures average expected quality when scores are continuous rather than binary. If an agent scores 0.8, 0.6, 0.9, 0.7, and 1.0 across five trials, the mean of 0.8 represents typical performance.

Minimum score across trials captures worst case behavior. The same agent's minimum of 0.6 reveals its floor, relevant for contexts where any single bad outcome matters.

Standard error of the mean (SEM) should accompany all aggregate metrics to communicate uncertainty. An agent scoring 0.75 ± 0.02 is meaningfully different from one scoring 0.75 ± 0.15.

4.4.3 Tool Usage Metrics

Agents interact with the world through tools, and tool usage quality is a distinct dimension from task completion. An agent might succeed despite poor tool choices (through redundant calls or lucky recovery) or fail despite correct tool usage (due to environmental factors). Tool metrics isolate this dimension.

Trajectory precision measures what fraction of the agent's tool calls were actually needed:

$$\text{Trajectory Precision} = (\text{Tool Calls in Reference})/(\text{Total Tool Calls by Agent})$$

An agent that makes 10 tool calls when only 5 were necessary has precision of 0.5, indicating wasted effort.

Trajectory recall measures what fraction of the necessary tool calls the agent made:

$$\text{Trajectory Recall} = (\text{Tool Calls by Agent} \cap \text{Reference})/(\text{Tool Calls in Reference})$$

An agent that makes 4 of the 5 necessary calls has recall of 0.8, indicating incomplete execution.

These metrics can be computed at three levels of strictness, as formalized by Google's Vertex AI evaluation service (Google Cloud, 2025):

Exact match requires the agent's tool sequence to be identical to the reference: same tools, same order, no extras. This is appropriate only when the sequence is genuinely canonical.

In order match requires all reference tools to appear in the agent's trajectory in the correct order, but permits extra tool calls between them. This captures tasks where ordering matters (read before write) while allowing exploratory behavior.

Any order match requires only that all reference tools appear somewhere in the trajectory, regardless of order or additional calls. This is the appropriate default for most evaluations where multiple valid paths exist.

Tool selection accuracy measures whether the agent chose the right tool for each situation, independent of parameter correctness:

$$\text{Tool Selection Accuracy} = (\text{Correct Tool Selections})/(\text{Total Tool Selection Decisions})$$

Argument correctness measures whether tool calls included correct parameter values:

$$\text{Argument Correctness} = (\text{Calls with Correct Arguments})/(\text{Total Tool Calls})$$

These two metrics decompose tool usage into selection and execution components, revealing agents that identify the right tools but fail to invoke them correctly, a common failure mode.

4.4.4 Efficiency Metrics

Two agents might both complete a task successfully, but one in 3 steps costing $0.02 while the other takes 30 steps costing $0.50. Efficiency metrics capture this distinction.

Trajectory length counts the number of steps (reasoning cycles, tool calls, or turns) the agent takes:

$$\text{Trajectory Length} = \text{Number of Steps from Task Start to Completion}$$

Shorter trajectories indicate more efficient reasoning though this must be balanced against success rate. An agent that fails quickly is not more efficient than one that succeeds slowly.

Token consumption measures the total tokens processed (input plus output) across the trajectory. This directly determines API costs for hosted models:

$$\text{Token Consumption} = \Sigma(\text{Input Tokens} + \text{Output Tokens}) \text{ across all steps}$$

Latency metrics capture time performance.

- **Time to first token (TTFT)** measures how quickly the agent begins responding.
- **End-to-end latency** measures total time from task submission to completion. For interactive agents, both matter: users want quick acknowledgment (low TTFT) and fast resolution (low end-to-end latency).

Cost of pass combines effectiveness and efficiency into a single metric:

$$\text{Cost of Pass} = \text{Total Cost}/\text{Successful Tasks}$$

This captures the economic reality of agent deployment. Research on agent cost optimization has found that optimizing for accuracy alone yields agents 4–10 times more expensive than cost aware alternatives achieving comparable performance. An agent with 80% success rate at $0.10 per task has cost of pass of $0.125, while one with 90% success at $0.50 per task has cost of pass of $0.56, nearly five times higher despite better raw accuracy.

4.4.5 Error Recovery Metrics

Errors are inevitable in multi-step processes. What distinguishes capable agents is not error avoidance (impossible to guarantee) but error recovery: detecting problems, diagnosing causes, and correcting course.

Correction success rate (CSR) measures the fraction of errors that the agent successfully corrects:

$$\text{Correction Success Rate} = (\text{Errors Corrected})/(\text{Errors Encountered})$$

This requires detecting when an error occurred (often through tool output or environmental feedback) and tracking whether the agent's subsequent actions resolved it.

Research on error correction reveals a counterintuitive "accuracy correction paradox": weaker models achieve substantially higher intrinsic correction rates than stronger models (26.8% versus 16.7% in one study). The explanation is that stronger models make fewer but "deeper" errors that resist correction, while weaker models make more frequent but shallower errors that they can often fix.

Error recovery decomposes into three subcapabilities:

1. **Error detection** measures whether the agent recognizes that something went wrong. An agent might receive a "file not found" error and continue as if the file were successfully read, failing at detection despite clear signals.
2. **Error localization** measures whether the agent correctly identifies where the problem originated. Knowing that something failed is insufficient; the agent must trace the failure to its source.
3. **Error correction** measures whether the agent successfully fixes identified problems. An agent might detect and localize an error but lack the capability to resolve it.

Agents often fail at detection while possessing correction capability: they cannot fix what they do not recognize as broken. Evaluation should measure all three subcapabilities independently to diagnose where improvement is needed.

Retry efficiency captures diminishing returns in self-correction. Research consistently shows that self-correction attempts saturate after approximately three retries, with additional attempts rarely producing success. Agents that retry indefinitely without success waste resources.

$$\text{Retry Efficiency} = \text{Success Rate Improvement} / \text{Additional Retries}$$

An agent whose success rate improves from 60% to 80% with one retry but only to 82% with two additional retries demonstrates diminishing retry efficiency.

4.5 Evaluators: Judging Agent Behavior

The metrics defined in Sect. 4.4 require someone or something to compute them. A task completion metric needs an evaluator to determine whether the task was completed. A tool usage metric needs an evaluator to judge whether each tool call was appropriate. This section examines the evaluators available for agent assessment: model-based judgment and human evaluation. Code-based verification (environment state checks, test suite execution, output validation) provides the foundation, but many important dimensions resist formal specification. Was the agent's reasoning sound? Did it communicate clearly? Did it handle ambiguity appropriately? For these dimensions, the evaluators discussed below become necessary.

Chapter 3 developed the foundations of evaluation methodology, including the trade-offs between human raters and LLM judges for assessing LLM outputs. Agent evaluation inherits those foundations but introduces additional complexity. Agents produce trajectories rather than single outputs, interact with environments that can be programmatically inspected, and exhibit non-deterministic behavior that complicates judgment. This section focuses on what changes when evaluating agents rather than repeating the general principles covered earlier.

When code cannot capture the evaluation criteria, LLMs can serve as judges. Chapter 3 covered the general methodology: LLM judges operate in pointwise mode (scoring a single output against criteria), pairwise mode (comparing two outputs), or reference-based mode (comparing output against a gold standard). Agent evaluation uses all three modes but must adapt them for trajectories rather than single outputs.

The key adaptation is *trajectory aware prompting*: providing the judge with the full sequence of agent actions, not just the final output. A prompt might include the original task, each reasoning step the agent produced, each tool call and its result, any user interactions, and the final outcome. The judge then assesses whether the trajectory demonstrates competent problem-solving.

For example:

Task: “Find the customer’s most recent order and check if it’s eligible for return.”
Trajectory:

Step 1: Agent called search_customer(email="john@example.com") → Found customer ID 4521
Step 2: Agent called get_orders(customer_id=4521, limit=1, sort="date_desc") → Order #8834, placed 2024-01-15
Step 3: Agent called get_return_policy() → 30-day return window
Step 4: Agent called get_current_date() → 2024-02-10
Step 5: Agent responded: “Your most recent order #8834 from January 15th is within the 30-day return window and eligible for return”

Evaluate this trajectory on a 1–5 scale for:

– Correctness: Did the agent reach the right conclusion?
– Efficiency: Did the agent take unnecessary steps?
– Completeness: Did the agent gather all required information?

This structure gives the judge visibility into the agent’s reasoning process, not just whether the final answer was correct.

For even finer grained signal, *step level judgment* evaluates each step independently rather than judging the trajectory as a whole. Rather than learning that a trajectory was “mostly good,” evaluation reveals that steps 1–7 were appropriate but step 8 represented a reasoning error. Process reward models formalize this approach, providing dense supervision at each intermediate step rather than sparse outcome only feedback.

A third adaptation is *dimension isolation*, applying the principle from Sect. 4.3.3: rather than asking a single judge for a holistic assessment, run separate judges for each dimension. One judge evaluates task completion, another evaluates tool usage appropriateness, a third evaluates communication quality. This reduces the cognitive load on each judge and produces more reliable scores per dimension.

The known biases of LLM judges apply to agent evaluation. Position bias causes judges to favor outputs presented first or last. Verbosity bias causes judges to prefer longer, more detailed responses regardless of quality. Self-preference causes models to favor outputs that match their own generation style. Mitigation strategies include randomizing presentation order, using length controlled comparisons, and employing judges from different model families than the agent being evaluated.

4.6 Agent Failure Modes

Effective evaluation requires knowing what to look for. Agents fail in characteristic ways that differ from single-turn LLM failures. Understanding these patterns helps design rubrics, metrics, and test cases that catch problems before deployment.

The most obvious failure is *tool looping*, where an agent repeatedly calls the same tool or sequence of tools without making progress. The agent might query a database, receive an empty result, and query it again with identical parameters, burning through tokens and time without advancing toward the goal. Evaluation should track repeated identical tool calls and flag trajectories where the agent appears stuck.

A related problem is *hallucinated tool calls*, where the agent invokes tools that do not exist, uses incorrect function signatures, or passes parameters that violate the tool's schema. Unlike hallucinated facts in text generation, hallucinated tool calls produce immediate errors that block progress. Metrics should track the rate of malformed or invalid tool invocations separately from correct calls that happen to return errors.

Even when individual tool calls succeed, *cascading errors* can corrupt entire trajectories. A wrong assumption in step 2 propagates through steps 3–10, producing a confidently wrong final answer. Because agents build on their own outputs, a single error can compound through the entire execution. Step level evaluation helps identify where cascades begin, enabling targeted improvement.

More subtle are *success mirages*, where the agent reports task completion without actually achieving the goal. The agent might say "I've updated the database" without having made any changes, or claim "the file has been created" when creation failed. Evaluation must verify claims against actual environment state rather than trusting the agent's self-report. This is why outcome evaluation should inspect the environment directly (Sect. 4.3.1).

Even genuine task completion can mask problems through *specification gaming*, where the agent achieves the literal goal while violating its spirit. Asked to "increase user engagement," an agent might send spam notifications. Asked to "resolve the customer complaint," it might issue an unauthorized refund. Evaluation should include both positive criteria (did the agent achieve the goal?) and negative criteria (did the agent violate any constraints?).

The most serious failure mode is *unsafe actions*. An agent might delete important files, expose sensitive data, execute dangerous commands, or take irreversible actions without appropriate confirmation. Safety evaluation requires trajectory inspection for prohibited actions regardless of whether the final outcome appears successful. A task that "succeeded" while taking dangerous intermediate steps should fail the safety dimension entirely.

These failure modes are not mutually exclusive. An agent might loop on a hallucinated tool call, eventually produce a success mirage, and in the process take an unsafe action. Comprehensive evaluation checks for each failure mode independently, using the dimension isolation principle from Sect. 4.3.3.

Failure mode	Metrics that detect it
Tool looping	Trajectory length, token consumption, cost of pass (Sect. 4.4.4)
Hallucinated tool calls	Argument correctness, tool selection accuracy (Sect. 4.4.3)
Cascading errors	Step success rate (Sect. 4.4.1), error detection/localization (Sect. 4.4.5)
Success mirages	Task success rate with environment verification, gap between reported vs verified success
Specification gaming	Constraint violation metrics, rubric dimensions for negative criteria (Sect. 4.3.2)
Unsafe actions	Binary safety gating via trajectory inspection (Sect. 4.3.2)

4.7 Benchmarks for Agent Evaluation

This section surveys the landscape of agent evaluation benchmarks, organizing them into major categories based on the environment: coding and software development, web navigation, tool use and function calling, general assistant, and specialized domains. For each category, we examine the design principles underlying major benchmarks, their strengths and weaknesses, inherent limitations, and how they compare to alternatives within the same category.

As previously discussed, the challenge of agent evaluation is multi-faceted. Agents operate in open-ended environments where success is often subjective, tasks may have multiple valid solutions, and the evaluation itself can be

computationally expensive or require human judgment. As this section explores these benchmarks, it becomes evident the recurring tensions between scalability and realism, between standardization and ecological validity, and between measuring what's easy to measure versus what truly matters for real-world deployment.

4.7.1 Coding Agent Benchmarks

Coding benchmarks evaluate agents' ability to write, debug, test, and reason about code. This category has seen explosive growth as code generation has become a key AI application.

Some popular benchmarks include, but not limited to:

- HumanEval (Chen 2021) with 164 problems and MBPP (Mostly Basic Python Problems, Austin et al. 2021) with 974 problems were among the first systematic benchmarks for code generation. Both present function-level coding tasks with natural language descriptions and unit tests for verification.
- LiveCodeBench's (Jain et al. 2024) primary strength is its focus on using recent programming problems released after the knowledge cutoff dates of LLMs being evaluated. Inspired by CRUXEval (Gu et al. 2024), this design addresses a critical issue in benchmark evaluation where models may have seen test problems during training, leading to inflated performance metrics.
- SWE-bench (Jimenez et al. 2024) represents a paradigm shift toward evaluating agents on real-world software engineering tasks. It contains 2294 task instances derived from actual GitHub issues in popular Python repositories, where the agent must locate and fix bugs in full codebases.
- CodeContests (Li et al. 2022) and CodeContests+ (Wang et al. 2025), derived from competitive programming platforms, and APPS (Hendrycks et al. 2021) with 10,000 coding problems across difficulty levels, evaluate algorithmic problem-solving ability through programming challenges (Table 4.1).

Table 4.1 Comparison of coding agent benchmark examples

	HumanEval and MBPP	LiveCodeBench	SWE-bench	CodeContests and APPS

	HumanEval and MBPP	LiveCodeBench	SWE-bench	CodeContests and APPS
Problem design	✓ Human-authored and manually verified × Static dataset	✓ Real-world problems drawing from competitive programming platforms like LeetCode, CodeForces, and AtCoder ✓ Reliable contamination prevention	✓ Derived from real open-source development	× Limited practical relevance for most software development
Task verification	✓ Automated evaluation through unit test execution		✓ Automated evaluation through unit test execution	✓ Automated evaluation through type-aware mutation-based method, generating a large amount of test cases
Task complexity	× Functions are typically short (10–20 lines) and self-contained			✓ Require genuine algorithmic reasoning ✓ Some problems require optimization, not just correctness
Task coverage	× Limited to algorithmic problems, missing broader software engineering tasks	✓ Improved coverage specifically targeting the problem characteristics × Limited to algorithmic problems, missing broader software engineering tasks	✓ Covers a broader range of capabilities, requiring understanding of large, realistic codebases, debugging, and testing × Limited to Python and specific open-source projects × Repository-specific knowledge may advantage models trained on GitHub data	× Competitive programming is a narrow skill compared to software engineering
Task diversity		✓ Multiple difficulty levels and issue types	✓ Multiple difficulty levels and issue types	✓ Diverse difficulty from introductory to competition level
Grading reliability	× Limited tests lead to high false-positive rate		× Some tasks are ambiguous or require external context	× Optimal solutions often require domain-specific algorithms

	HumanEval and MBPP	LiveCodeBench	SWE-bench	CodeContests and APPS
Grading standardization	✓ Standardized success criteria (pass@*k* metric) × No evaluation of code quality, documentation, or style	× No evaluation of code quality, documentation, or style		
Computation	✓ Fast		× Expensive	
Adoption	✓ Broad		× Requires full environment setup	

The HumanEval and MBPP benchmarks measure coding-in-the-small: writing isolated functions given complete specifications. Real software development, however, involves understanding large codebases, debugging across files, writing tests, and making architectural decisions. The gap between HumanEval performance and practical coding utility has continued to widen as models have saturated these benchmarks.

The LiveCodeBench benchmark depends on continuous release of new problems from competitive programming platforms, in order to maintain a truly contamination-free benchmark. This creates an arms race between benchmark creators and model developers. At the same time, all competitive-programming-based benchmarks suffer from the fact that, while intellectually challenging, the problems may not reflect the kinds of coding tasks developers face daily, such as integrating APIs, handling edge cases in production systems, or writing maintainable code for team environments.

While SWE-bench dramatically improves realism over function-level benchmarks, it still focuses narrowly on bug-fixing. It doesn't evaluate software design, feature implementation from vague requirements, code review, or collaborative development. The reliance on existing test suites also means agents aren't tested on writing comprehensive tests themselves.

The CodeContests and APPS benchmarks, on the other hand, excel at measuring algorithmic prowess but have minimal overlap with day-to-day software engineering. Most professional programmers rarely implement graph algorithms or dynamic programming solutions from scratch. The skills measured are valuable but insufficient for evaluating practical coding assistance.

The coding benchmark landscape shows clear evolution from isolated function generation (HumanEval/MBPP) to algorithmic problem-solving (CodeContests/APPS) to realistic software engineering (SWE-bench). Each successive generation trades ease of evaluation for realism and complexity. This section discussed not all but only a few coding benchmarks. More are being created and curated as this book is being drafted.

A comprehensive evaluation strategy requires multiple benchmarks: HumanEval for basic proficiency, SWE-bench for practical capability, and potentially domain-specific benchmarks for particular applications. The field still lacks good benchmarks for collaborative coding, architectural decision-making, code review, and long-term codebase maintenance.

4.7.2 Web Navigation Agent Benchmarks

Web navigation benchmarks evaluate agents' ability to interact with websites through browsers, including clicking, typing, navigating, and extracting information to accomplish user goals.

The table below lists some aspects of strengths and weaknesses of three web navigation agent benchmarks. As the previous section, this list is illustrative and not comprehensive:

- WebArena (Zhou et al. 2023) provides a realistic web environment with functional websites where agents must complete complex tasks like managing shopping carts, posting content, and retrieving information. It includes 812 long-horizon tasks requiring multiple interactions, with a wide range of website categories including e-commerce, social forums, and collaborative software.
- Mind2Web (Deng et al. 2023) takes a different approach by collecting more than 2000 web tasks across 137 real websites, focusing on generalization to unseen websites and tasks. The benchmark provides task descriptions and ground truth action sequences.
- WebShop (Yao et al. 2022b) creates a simulated e-commerce environment where agents must search for and purchase products matching specific criteria, such as attributes and price ranges. It contains 12,087 products and 1.18 million search queries (Table 4.2).

Table 4.2 Comparison of web agent benchmark examples

	WedArena	**Mind2Web**	**WebShop**

	WedArena	Mind2Web	WebShop
Website design	✓ Fully functional websites provide realistic interaction environment	✓ Real-world websites	× Simplified websites
Task diversity	– Different website genres but only a few domains	✓ Many diverse website genres with generalization to new domains	× E-commerce only
Task verification	✓ Automatic verification of task completion	✓ Annotations include natural language rationales	✓ Clear success criteria
Task complexity	✓ Multi-step planning and execution	✓ Multi-step planning and execution	× Limited interaction types
Task coverage	✓ Navigation; information extraction		– Search; decision-making
Task reproducibility	× Websites may have variable behavior or timing issues	× Websites change over time, breaking tasks	✓ Controlled environment ensures reproducibility
Grading sensitivity	× Can be overly sensitive to small interface changes	× Ground truth actions may not be unique valid solutions	✓ Large scale enables robust statistical evaluation
Computation	× Expensive (requires running web servers)		✓ Fast
Setup complexity	× High	× Requires complex visual and structural understanding	✓ Fast

WebArena's controlled environment, while realistic within its scope, doesn't capture the full complexity of real web navigation. For example, handling dynamic content, dealing with CAPTCHAs, navigating poorly designed interfaces, or adapting to unexpected errors. The static task set also can't evaluate generalization to new websites or novel task types.

Mind2Web's breadth comes at the cost of controlled evaluation. Since tasks are on live websites or snapshots, reproducibility is challenging and the benchmark degrades over time. The focus on action sequence matching also doesn't capture whether alternate approaches might be equally or more effective.

WebShop trades realism for scalability and control. The simplified e-commerce environment doesn't include many complexities of real shopping: reviews, comparisons, size guides, shipping calculations, or promotional codes. It's excellent for controlled experiments but limited in generalizability to real web navigation.

All three benchmarks struggle with the same fundamental issues: websites change constantly, tasks often have multiple valid solutions, and success can be

subjective. Additionally, current benchmarks focus heavily on goal-directed navigation, while underemphasizing exploratory browsing, information synthesis across multiple sites, or assessing information credibility.

4.7.3 Tool Use Agent Benchmarks

Tool use benchmarks evaluate agents' ability to interact with external APIs, functions, and services. This is a critical capability for agents that need to retrieve information, manipulate data, or interact with software systems.

- ToolBench (Qin et al. 2023) provides a large-scale collection of tool use scenarios with 16,000+ real-world APIs, while API-Bank (Li et al. 2023) focuses on API interaction with 53 API tools across multiple domains. Both benchmarks test whether agents can select appropriate tools, construct valid API calls, and interpret results.
- The Berkeley Function-Calling Leaderboard (Hao et al. 2025) specifically evaluates function-calling accuracy across multiple categories: simple function calls, parallel calls, multiple functions, and relevance detection. It contains over 2000 test cases with diverse function schemas (Table 4.3).

Table 4.3 Comparison of tool use benchmark examples

	ToolBench	**Berkeley function-calling leaderboard**
Task diversity	✓ Covers diverse tool types and domains	× Focuses on single-turn interactions rather than multi-step tasks × Limited real-world context for function calls
Task verification	× Often requires simulated API responses rather than live calls	✓ Standardized evaluation methodology × Evaluation is purely structural (valid JSON) rather than semantic
Task complexity	✓ Evaluates multi-step tool use (calling multiple APIs in sequence)	✓ Includes challenging cases like parallel execution
Task coverage	✓ Tests both tool selection and parameter filling × Limited assessment of error handling and edge cases	✓ Comprehensive coverage of function-calling patterns ✓ Tests whether agents can avoid calling functions when inappropriate × Doesn't evaluate function composition or complex workflows
Task freshness	× Many APIs may be outdated or deprecated	✓ Regular updates with new test categories
Grading sensitivity	× Ground truth solutions may not be unique	

Toolbench focuses on functional correctness, but doesn't evaluate important practical concerns like efficiency, e.g., minimizing API calls, cost awareness, handling rate limits, or graceful degradation when tools fail. The use of simulated rather than live APIs also creates a gap between benchmark performance and real-world robustness.

The Berkeley Function-Calling Leaderboard benchmark excels at testing the mechanics of function calling but doesn't evaluate whether agents use functions effectively to accomplish user goals. An agent could score perfectly on function-calling syntax while choosing the wrong functions or calling them in inefficient sequences.

Tool use benchmarks face a fundamental tension between coverage and depth. ToolBench's breadth (16,000+ APIs) provides comprehensive coverage but with shallow evaluation of each tool. Berkeley Function-Calling focuses deeply on the mechanics of calling functions but with limited real-world context.

Neither benchmark adequately tests tool use in truly open-ended scenarios where agents must discover available tools, reason about tool capabilities from documentation, or creatively combine tools to solve novel problems. The field needs benchmarks that evaluate tool use as part of complete task solutions rather than as an isolated skill.

4.7.4 General Assistant Agent Benchmarks

General assistant benchmarks evaluate an agent's ability to be helpful across diverse everyday tasks, such as answering questions, providing recommendations, helping with planning, and engaging in natural conversation while using tools or accessing information.

- GAIA (Mialon et al. 2023) was designed to evaluate AI assistants on questions that are conceptually simple for humans but require agents to combine multiple capabilities: web search, multi-step reasoning, tool use, and information synthesis. The benchmark contains around 450 questions spanning science, geography, current events, and practical tasks, with questions explicitly designed to require genuine understanding rather than pattern matching.
- AssistantBench (Yoran et al. 2024) takes a different approach by evaluating agents on realistic personal assistant tasks: managing emails, scheduling, web research, and file manipulation. It consists of 214 tasks that require planning, tool use, and understanding of user context (Table 4.4).

Table 4.4 Comparison of general assistant benchmark examples

	GAIA	AssistantBench
Task diversity	× Relatively small dataset size limits statistical robustness	✓ Includes tasks requiring multi-turn interaction and context maintenance
Task verification	✓ Human-verifiable with objective ground truth answers; designed to be unambiguous with clear success criteria × Binary correct/incorrect scoring misses partial progress	✓ Evaluates end-to-end task completion, not just final answers ✓ Measures both task success and efficiency (number of steps) × Evaluation often requires manual verification
Task complexity	✓ Requires genuine multi-step reasoning and tool use rather than pure memorization; difficulty levels (1–3) allow for nuanced capability assessment	
Task coverage	× Limited coverage of creative or subjective tasks × Doesn't evaluate conversational quality or user experience	✓ Reflects realistic use cases for personal AI assistants × Smaller scale limits comprehensive coverage
Task freshness	× Static benchmark vulnerable to data contamination over time	× Tasks may become outdated as digital tools evolve
Setup complexity		× Setup complexity makes reproduction and difficult

GAIA focuses on factual questions with objective answers. This excludes important assistant capabilities like providing personalized advice, creative ideation, or navigating ambiguous user requests. The requirement for verifiable answers also biases toward information retrieval tasks rather than generative or planning tasks.

On the other hand, AssistantBench's realism comes at the cost of scalability. Many tasks require specific environment configurations, and evaluation can be labor-intensive. The focus on productivity tasks also excludes other important assistant capabilities like emotional support, creative collaboration, or educational tutoring.

These two benchmarks represent two philosophies in assistant evaluation. GAIA prioritizes scalability and objective evaluation through question-answering, making it easier to run and compare across models. AssistantBench prioritizes ecological validity through realistic task scenarios, providing better insight into real-world performance but at greater evaluation cost.

GAIA's questions can be answered in isolation, while AssistantBench's tasks often require maintaining context across multiple actions. GAIA measures whether agents can find correct information, while AssistantBench measures whether agents can effectively accomplish user goals. Neither fully captures the open-ended, conversational nature of real assistant interactions, where user satisfaction involves factors like response style, proactive helpfulness, and graceful handling of ambiguity.

4.7.5 Special Domain Agent Benchmarks

Beyond the major categories, several specialized benchmarks evaluate agents in specific domains or on particular capabilities.

- AgentClinic (Schmidgall et al. 2024) introduces genuine agentic complexity, bias modeling, and multimodal elements that better reflect real clinical work. Its main weaknesses are the small scale of cases, reliance on US-centric educational datasets, and a reproducibility challenge inherent to using LLMs for both the evaluated agent and the simulation environment itself. It is best understood as a proof-of-concept framework for interactive clinical evaluation rather than a comprehensive, production-grade benchmark.
- Scientific research tasks require specialized reasoning, experimental design, and integration of domain knowledge. ScienceAgentBench (Chen et al. 2024) evaluates agents on literature review, hypothesis generation, experimental planning, and data analysis in scientific contexts. This benchmark provides valuable insight into specialized capabilities but lacks generalizability. Success on ScienceAgentBench doesn't necessarily predict performance in legal, financial, or creative domains. The field needs more domain-specific benchmarks while acknowledging each has limited scope.
- Mobile environments present unique challenges: Touch interfaces, app-specific interaction patterns, limited screen space. Benchmarks like MobileAgentBench (Wang et al. 2024) evaluate agents controlling mobile apps to complete tasks. Running agents in mobile environments requires sophisticated tooling, tasks are platform-specific, and the rapid evolution of mobile apps makes sustained benchmarks difficult. The field needs standardized mobile simulation environments.

4.8 Discussion

Despite progress in agent evaluation, fundamental challenges persist across all benchmark categories:

- Static vs. Dynamic: Most benchmarks are static datasets vulnerable to memorization and contamination. As models train on increasingly large web corpora, distinguishing genuine capability from memorization becomes harder. Dynamic benchmarks that generate new tasks are expensive and risk reduced quality or unintended biases.
- Evaluation Scalability vs. Realism: There's an inherent trade-off between benchmarks that can evaluate thousands of agents quickly (like HumanEval) and those that simulate realistic complexity (like SWE-bench). Realistic evaluation often requires expensive infrastructure, human judgment, or long execution times.
- Single-Task vs. Multi-Task: Most benchmarks evaluate isolated tasks, but real agents must juggle multiple concurrent goals, maintain context across interruptions, and prioritize among competing objectives. Few benchmarks capture this essential aspect of agency.
- Success Metrics: Binary success/failure is clean but crude. Partial credit, efficiency metrics, and solution quality all matter but are harder to standardize. The field lacks consensus on how to measure "good enough" vs. optimal performance.
- Safety and Alignment: Current benchmarks overwhelmingly focus on capability while under-measuring safety, truthfulness, or alignment. An agent that successfully completes tasks while occasionally hallucinating, violating privacy, or acting deceptively might score well on most benchmarks.

For practitioners, the implication is clear: no single benchmark tells the whole story. A comprehensive evaluation strategy requires multiple benchmarks spanning different categories, supplemented by domain-specific tests and real-world deployment metrics. Researchers developing new benchmarks should be explicit about their design choices and limitations rather than claiming comprehensive evaluation.

Looking forward, the field needs benchmarks that better capture long-horizon planning, multi-agent coordination, robust uncertainty handling, and alignment with human values. We need evaluation frameworks that work for continual learning rather than static snapshots. Most importantly, we need closer coupling between benchmark performance and real-world impact.

The ultimate test of an agent isn't its score on any benchmark but whether it reliably helps people accomplish their goals safely and effectively. As agents become more capable and widely deployed, our evaluation methodologies must evolve to ensure that benchmark success predicts real-world value.

References

Anthropic: Building Effective Agents. Anthropic Research. https://www.anthropic.com/research/building-effective-agents (2024)

Austin, J., et al.: Program synthesis with large language models. arXiv preprint arXiv:2108.07732 (2021)

Chen, M.: Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021)

Chen, Z., et al.: Scienceagentbench: toward rigorous assessment of language agents for data-driven scientific discovery. arXiv preprint arXiv:2410.05080 (2024)

Deng, X., et al.: Mind2web: towards a generalist agent for the web. Adv. Neural Inf. Process. Syst. **36**, 28091–28114 (2023)
[https://doi.org/10.52202/075280-1220]

Google Cloud: Evaluate Gen AI Agents. Google Cloud Documentation. https://docs.google.com/vertex-ai/generative-ai/docs/models/evaluation-agents (2025)

Gu, A., et al.: Cruxeval: a benchmark for code reasoning, understanding and execution. arXiv preprint arXiv:2401.03065 (2024)

Hao, B., et al.: Reasoning through exploration: a reinforcement learning framework for robust function calling. arXiv preprint arXiv:2508.05118 (2025)

Hendrycks, D., et al.: Measuring coding challenge competence with apps. arXiv preprint arXiv:2105.09938 (2021)

Jain, N., et al.: Livecodebench: holistic and contamination free evaluation of large language models for code. arXiv preprint arXiv:2403.07974 (2024)

Jimenez, C.E., Yang, J., Wettig, A., et al.: SWE-Bench: Can Language Models Resolve Real-World GitHub Issues? ICLR. https://arxiv.org/abs/2310.06770 (2024)

Kenton, Z., Kumar, R., Farquhar, S., et al.: Discovering Agents. Google DeepMind. https://deepmind.google/discover/blog/discovering-when-an-agent-is-present-in-a-system/ (2023)

Li, Y., et al.: Competition-level code generation with alphacode. Science. **378**(6624), 1092–1097 (2022)
[ADS][https://doi.org/10.1126/science.abq1158][PubMed]

Li, M., et al.: Api-bank: a comprehensive benchmark for tool-augmented LLMs. In: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (2023)

Ma, C., Zhang, J., Zhu, Z., et al.: AgentBoard: an analytical evaluation board of multi-turn LLM agents. NeurIPS. https://arxiv.org/abs/2401.13178 (2024)

Mialon, G., et al.: Gaia: a benchmark for general AI assistants. In: The 12th International Conference on Learning Representations (2023)

OpenAI: New tools for building agents. OpenAI Blog. https://openai.com/index/new-tools-for-building-agents/ (2025a)

OpenAI: A practical guide to building agents. OpenAI. https://cdn.openai.com/business-guides-and-resources/a-practical-guide-to-building-agents.pdf (2025b)

Qin, Y., et al.: ToolLLM: facilitating large language models to master 16000+ real-world APIs. arXiv preprint

arXiv:2307.16789 (2023)

Schmidgall, S., et al.: Agentclinic: a multimodal agent benchmark to evaluate AI in simulated clinical environments. arXiv preprint arXiv:2405.07960 (2024)

Wang, L., et al.: Mobileagentbench: an efficient and user-friendly benchmark for mobile LLM agents. arXiv preprint arXiv:2406.08184 (2024)

Wang, Z., et al.: CodeContests+: high-quality test case generation for competitive programming. arXiv preprint arXiv:2506.05817 (2025)

Xi, Z., Chen, W., Guo, X., et al.: The rise and potential of large language model based agents: a survey. arXiv preprint arXiv:2309.07864. https://arxiv.org/abs/2309.07864 (2023)

Yao, S., Zhao, J., Yu, D., et al.: ReAct: synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629. https://arxiv.org/abs/2210.03629 (2022a)

Yao, S., et al.: Webshop: towards scalable real-world web interaction with grounded language agents. Adv. Neural Inf. Process. Syst. **35**, 20744–20757 (2022b) [https://doi.org/10.52202/068431-1508]

Yao, S., Shinn, N., Razavi, P., et al.: τ-bench: a benchmark for tool-agent-user interaction in real-world domains. arXiv preprint arXiv:2406.12045. https://arxiv.org/abs/2406.12045 (2024)

Yoran, O., et al.: Assistantbench: can web agents solve realistic and time-consuming tasks? In: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (2024)

Zhou, S., Xu, F.F., Zhu, H., et al.: WebArena: a realistic web environment for building autonomous agents. NeurIPS. https://arxiv.org/abs/2307.13854 (2023)

A. Robsky et al., *Reliable Evaluations for LLMs and AI Agents*
https://doi.org/10.1007/978-3-032-26749-8_5

5. Evaluation Infrastructure

Alexei Robsky[1] ✉, Liliya Lavitas[2] and Yueqing Wang[3]
(1) Microsoft, Sammamish, WA, USA
(2) Google (United States), Newton, MA, USA
(3) Microsoft, San Francisco, CA, USA

5.1 Evaluation Automation

Building an evaluation platform requires a strategic understanding of the evaluation pipeline. Not every stage requires the same level of infrastructure investment; the urgency is typically determined by the frequency of updates and the computational intensity of the task.

Conceptually, three distinct phases of evaluation pipeline can be named:

- Creation of the Evaluation Sets, Templates, and Metrics
- Execution of evaluations
- Analysis of results

In this section, we will discuss levels of automation needed for each of the phases of the process.

5.1.1 Creation of the Evaluation Sets, Templates, and Metrics

As we discussed earlier, the evaluation set is the foundational test for the model. In Sect. 3.1, we talked about different set designs required for different stages of the AI system's maturity, evolving from simple sanity checks to rigorous, tailored frameworks. According to the "Four Stages of Evaluation Set Evolution," the infrastructure requirements for data creation shift as a model moves toward production:

- **Stage 1**: Manual Curation and Sanity Checks: In the initial prototyping phase, evaluation sets are typically small (10–50 examples) and handcrafted by development teams to test basic instruction following. At this stage, speed and clarity are prioritized over automation, as the set can be created and modified within hours. For this stage, it might appear that not much infrastructure investment is needed, but still we would advise practitioners even at this early stage to think through future system design and enforce some data formatting standardization.
- **Stage 2**: Public Benchmarks: As models improve, teams transition to well-vetted public benchmarks like MMLU (Hendrycks et al. 2021), HumanEval (Wang 2024), or HELM (Stanford) (Kaiyom et al. 2024) to establish credibility and baseline capabilities across diverse areas. While these sets are easily accessible and available for public use, data pipelines and appropriate wiring of these sets are needed to enable smooth and efficient usage of these sets.

- **Stage 3**: Custom Evaluation Sets: Once benchmarks saturate, the mindset shifts from academic performance to user utility ("how well does this system serve our users?"). This stage requires substantial infrastructure for tracking, automated inference, and integration with human data components to manage hundreds or thousands of domain-specific scenarios. The reference to human data components at this stage deserves further elaboration. Custom evaluation at scale typically requires dedicated annotation infrastructure: platforms for distributing labeling tasks, interfaces tailored to the specific rubric being applied, and systems for tracking inter-annotator agreement to ensure consistency. Managing annotator pools, monitoring label quality over time, and handling disagreement resolution workflows are all engineering problems that compound as evaluation sets grow. Teams that treat annotation as an informal process at Stage 3 often find it becomes the primary bottleneck to evaluation velocity.
- **Stage 4**: Real-World Examples (Online Evaluation): The final stage involves controlled experimentation in the wild, such as A/B testing and gradual rollouts. This requires highly specialized infrastructure capable of capturing actual user distributions and intermediate telemetry data while managing the noise inherent in production environments.

Beyond the stages themselves, a foundational infrastructure requirement that spans all stages is versioning of evaluation sets. Evaluation sets are not static; items are added, removed, or reweighted as understanding of model behavior deepens. Without disciplined version control, it becomes impossible to determine whether a change in model performance reflects genuine improvement or simply a shift in what is being measured. Every evaluation run should be traceable to a specific, immutable snapshot of the set it was run against (Fig. 5.1).

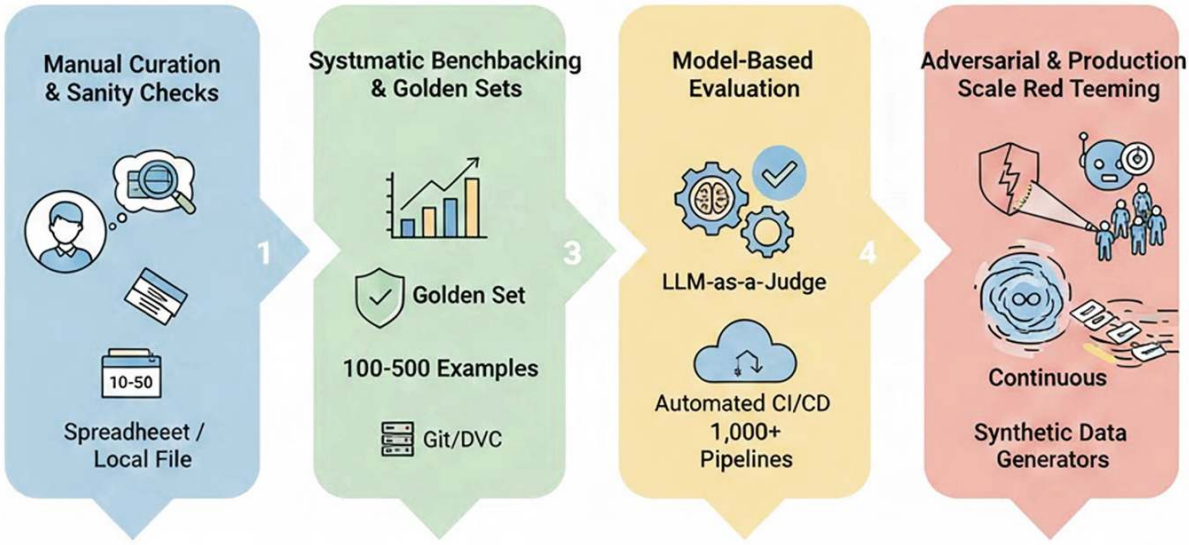

Fig. 5.1 Four stages of evaluation infrastructure needs

A critical infrastructure requirement arises when custom sets (Stage 3) or online sets (Stage 4) are derived from production logs. While these logs offer the best reflection of real-world intent, they are often laden with personally identifiable information (PII) and require specialized privacy-preserving layers, as discussed in the following sections.

A related concern is data contamination. As evaluation sets grow, particularly when sourced from public or semi-public corpora, the risk increases that test items have appeared in a model's training data, inflating performance estimates. Infrastructure should include automated contamination detection, such as n-gram overlap checks or embedding similarity scans between evaluation items and known training sources. This is especially important for custom sets (Stage 3) that may draw from the same domain-specific content used for fine-tuning.

Templates and Metrics design efforts can typically be happening in parallel with set creation work. As we have detailed in Sect. 3.2.3, creating deconstruction of Chain-of-Thought templates remains a deeply human endeavor and requires deep expertise and careful crafting. Because Template (or Chain-of-Thought) rubrics define the core requirements of a model and do not change daily, the urgency for

automating their creation is low. The focus for infrastructure at this stage is on version control and management, ensuring that as rubrics evolve, they can be consistently applied to both historical and new model iterations.

5.1.2 Running Evaluations at Scale

The execution phase is the most infrastructure-heavy and requires thoughtful system design and automation from the get-go. At the orchestration level, a production evaluation system rarely runs a single evaluation in isolation. At any given time, multiple evaluation runs may be in flight: a new model checkpoint tested against the core set, a regression suite triggered by a code change, and a periodic re-evaluation of production models running on a schedule. This requires a job orchestration layer that manages scheduling, priority, and resource allocation across concurrent runs. Without it, teams quickly find themselves in a situation where urgent evaluations are blocked behind long-running batch jobs, or where competing runs degrade each other's throughput.

A production-ready system must automate three subtasks:

- **Obtaining model responses**: High-throughput inference engines are needed to query models across thousands of prompts in parallel. Model responses must be representative of the actual user experience, so for certain tasks model output as-of-time of query issue is needed, and this increases complexity even further. Obtaining a time-agnostic model response for a prompt: "what is 2+2?" is very different from obtaining a highly time-dependent response to the prompt: "what time is it now?". Infrastructure and model calling should account for this.

  Any system making thousands of inference calls must be designed with the assumption that some will fail. API rate limits, network timeouts, model service outages, and malformed responses are not edge cases at scale; they are routine. Infrastructure must support automatic retries with backoff, graceful handling of partial failures, and the ability to resume an interrupted evaluation run from where it

left off without re-processing completed items. A single failed call should never invalidate an entire evaluation run, nor should it silently produce incomplete results that are later treated as complete.

- **Obtaining labels**: Obtaining labels is the key step of evaluations. For early stages of model development (see Sect. 3.1) reliance on human raters is typical. Engineers working on evaluation systems need to think through the process of labels collection and storing. As systems mature into later stages and scale beyond initial phases a need to bypass the human rater bottleneck arises, thus systems start relying on "LLM-as-a-Judge" frameworks (see Sect. 3.4). Infrastructure must evolve to allow this much more computationally expensive process.

  The computational cost of LLM-as-a-Judge evaluation deserves more than a passing acknowledgment, as it often becomes the dominant cost in the entire evaluation pipeline. A single complex rubric applied by a frontier model to ten thousand examples can cost more than the inference run it is evaluating. Practical infrastructure must support strategies for managing this: batching requests to maximize throughput, selecting judge models of appropriate capability for the difficulty of the task, and parallelizing judge calls across available compute. Teams should also consider whether a tiered approach is viable, where a faster, cheaper model handles straightforward cases, and a more capable model is reserved for ambiguous or borderline examples.

  In practice, the transition from human raters to automated judges is rarely a clean switch. Most mature evaluation systems operate in a hybrid mode where human and automated judgment coexist. This requires routing infrastructure that can direct individual evaluation items to the appropriate judge based on predefined criteria: confidence thresholds from the automated judge, domain sensitivity, or random sampling for ongoing calibration. When an LLM judge produces a low-confidence label, the item should be automatically escalated to a human reviewer. Conversely, human-labeled items can serve as a continuous calibration signal, allowing teams to monitor whether the automated judge is drifting from human standards over

time. Designing this routing as a first-class component of the system, rather than bolting it on later, significantly improves both the quality and cost-efficiency of evaluation at scale.

- **Results compilation and analysis**: Effective evaluation infrastructure must go beyond simple point estimates to compute confidence intervals and support multidimensional "slices" (like language or region) by storing rich metadata from the outset. Because model improvements often involve trade-offs such as safety versus helpfulness, the system must enable multi-objective optimization to map various trade-offs, allowing teams to choose the best model checkpoint based on competing requirements. To make these results actionable, dashboards must provide intuitive, stakeholder-specific views that highlight topline metrics and cross-model comparisons. Finally, the analysis layer should automate regression detection to catch statistically significant drops immediately and provide longitudinal tracking to ensure that performance gains remain stable over months of iterations and architectural shifts. In the next section, we will discuss all of these components in detail.

5.1.3 Results Compilation and Analysis

Once labels (human or machine) are collected, the system must compute statistical metrics per instructions in metrics design (see Sect. 3.3). Infrastructure must be designed to allow computation of not only simple aggregation point estimates metrics, but also confidence intervals around them and other descriptive statistics that are needed for decision-making. Ideally, this analysis needs to be robust enough and efficient enough to produce topline metrics fast and reliably, but also flexible to allow ad-hoc analysis and deep-dives. For example, a common deep-dive analysis is assessment of "helpfulness," as a function of model response length.

The helpfulness-by-response-length example illustrates a broader infrastructure requirement: the ability to slice evaluation results along arbitrary dimensions without re-running the evaluation. Teams

routinely need to analyze performance by language, geographic region, task category, user segment, input complexity, or any combination of these. This is only possible if the evaluation system stores rich metadata alongside each evaluation item and its corresponding label, and if the analysis layer provides flexible querying and grouping over that metadata. Designing this capability in from the start is essential; retrofitting it after the fact typically means re-running months of evaluations with additional metadata attached, which is costly and often impractical.

As we have highlighted many times throughout this book: model evaluation is almost never a single-dimension task. This multidimensionality of evals results in multi-vector optimization needs.

Imagine very common examples of trade-offs: Safety and Helpfulness (the more the model refuses to fulfill users' response the less likely it produces unsafe response). We talked about this trade-off in Sects. 3.2 and 3.3. Another common trade-off is between verbosity and completeness, or between factuality and latency. In most cases, evaluations are done across a panel of metrics, where the same model is being evaluated for multiple capabilities and characteristics. Without proper architectural solutions, making decisions when presented with dozens of evaluation results is virtually impossible and evaluation metrics become not instruments for decision-making, but a burden.

Thus, effective infrastructure must support Multi-Objective Optimization to map the Pareto Frontiers, which represent the set of optimal model configurations where no objective can be improved without degrading another. Technically, this requires running parallel evaluation pipelines that test different preference weights, allowing teams to visualize the trade-off curve and select the model checkpoint that best meets their specific safety and capability requirements.

Infrastructure for computing metrics is only half the equation; the other half is presenting them in a way that enables decisions. Evaluation dashboards are often the primary interface through which engineering leads, product managers, and safety teams interact with evaluation results. A well-designed dashboard should surface topline

metrics at a glance, allow drill-down into individual metric dimensions, and make cross-model comparison intuitive. Poor dashboard design (cluttered views, missing baselines, no ability to filter) can undermine even the most rigorous evaluation pipeline by burying actionable insights under noise. Dashboard infrastructure should also support customizable views for different stakeholders since a safety reviewer and a product lead are looking for fundamentally different signals in the same data.

Computing confidence intervals is a necessary foundation, but a production system should not rely on humans to manually inspect every interval for concerning shifts. Automated regression detection should be a core component of the analysis layer. When a new model checkpoint shows a statistically significant degradation on any tracked metric relative to the established baseline, the system should generate an alert without waiting for someone to open a dashboard. This requires defining acceptable thresholds per metric, selecting appropriate statistical tests for comparison, and routing alerts to the right team. Without this, regressions that are clearly visible in the data can go unnoticed for days or weeks, especially when teams are evaluating across dozens of metrics simultaneously.

Beyond detecting regressions between two consecutive checkpoints, infrastructure must support longitudinal analysis across many evaluation runs over time. Questions like “how has factuality trended over the last twenty model iterations?” or “did the safety improvements introduced in March hold through subsequent training runs?” require time-series storage of evaluation results, consistent metric definitions across runs, and visualization tools that can plot metric trajectories alongside key development milestones such as training data changes, fine-tuning experiments, or architectural modifications. This historical view is often where the most strategically valuable insights emerge, revealing gradual drifts that no single-run comparison would catch.

Table 5.1 below summarizes key points of the evaluation automation needs within the evals framework lifecycle:

Table 5.1 Evaluation automation needs within the evals framework lifecycle

Evaluation framework component	Automation urgency	Infra need to include
Evaluation sets creation	Medium	PII protection and protocols. Access control. Version control of sets with immutable snapshots. Contamination detection against known training sources.
Templates and metrics creation	Low	Version control of templates and metric. Change tracking with backward compatibility to historical evaluation runs.
Evaluations execution	High	Job orchestration and scheduling across concurrent runs. High-throughput inference with retry logic and partial-run recovery. LLM-as-a-Judge pipelines with cost management (batching, model tiering). Human-in-the-loop routing for hybrid labeling workflows. Caching and deduplication keyed to model, set, and rubric versions.
Evaluations analysis	High	Metrics computation with confidence intervals and descriptive statistics. Automated regression detection and alerting against baselines. Historical trending and longitudinal comparison across runs. Segmented analysis with flexible metadata-based slicing. Version control of analysis code. Dashboarding for results communication.

Every analysis, whether a scheduled topline report or a one-off deep-dive, should be reproducible. In practice, this means analysis logic must be captured as versioned code tied to specific evaluation run snapshots, not as ad-hoc notebook cells or spreadsheet formulas that exist only on one engineer's laptop. When a stakeholder questions a result or a finding needs to be revisited months later, the team must be able to re-execute the exact same analysis against the exact same data and arrive at the same conclusion. Infrastructure should enforce this by design, treating analysis artifacts with the same rigor as evaluation sets and templates.

Capabilities of running evaluations at scale form the backbone of the evaluation lifecycle, and in the next chapter, Chap. 6, we will explore how these components are deeply interconnected, and we will detail

how robust infrastructure is the prerequisite for the continuous feedback loops that drive model progress.

5.1.4 Architectural Design: Isolation of the Model from the Evaluation Framework

Designing a robust evaluation system is as much an engineering challenge as it is a statistical one. It is a common pitfall to treat evaluation as a secondary script that runs alongside the model; however, as LLMs scale, the infrastructure required to measure them must be thought through with high rigor. A poorly architected system can lead to "eval-drift," where the environment itself introduces bias or failure.

To understand the stakes of these design choices, we find it incredibly insightful to look at the LLM competition (Kaiyom et al. 2024). This competition was conducted to introspect the evaluation of large language models fine-tuned on specialized data. According to the study report by Vicki Boykis: "The idea for the competition started from the premise that current offline evaluation of large language models is an extremely complex and nuanced task, compounded by the speed of model development, continuously increasing options for evaluation metrics and their implementation, as well as lack of community consensus around metrics and pipelines. <…> In this light, one of the primary goals of the competition was to shed light on how people perform this kind of evaluation in practice. <…> The primary machine learning objective was to be able to fine-tune LLMs in a 24-h period on a single GPU."

Among other findings, this study highlighted the necessity of cloud-native architectures that isolate the model from the evaluation framework, thus ultimately shifting evaluations toward "Evaluation as a Service" (EaaS) paradigm.

Retrospective analysis demonstrated four critical failure modes that occur when isolation is neglected:

- **Environmental contamination**: When evaluation code is "in-process" (nested within the model's environment), model

dependencies can interfere with metrics or create “dependency hell”. Decoupling ensures the “ruler” (the framework) remains consistent even as the “subject” (the model) changes.

- **Bloated artifacts**: Competition entries that bundled the evaluation server inside the model container resulted in massive artifacts (up to 100 GB) making it nearly impossible to swap or version models efficiently.
- **Resilience and persistence**: Without isolated storage, a single pod failure during a long run results in a total loss of progress; isolation allows the evaluation state to be preserved independently of the model instance.
- **Resource contention**: Co-location of inference and evaluation forces these processes to compete for the bandwidth, leading to increased latency.

5.1.5 The Path Forward: Evaluation as a Service (EaaS)

By analyzing these failure modes, it becomes clear that the only way to achieve industrial-scale reliability is to treat the evaluation framework as a sovereign entity, completely decoupled from the model’s ephemeral training or inference environment. Moving toward an Evaluation as a Service (EaaS) paradigm isn’t just about clean code; it’s about building a “ruler” that doesn’t bend depending on what it is measuring.

In a mature EaaS architecture, the model and the evaluator live in isolated, cloud-native containers that communicate over standardized APIs. This setup solves dependencies of environmental contamination and ensures that a crash in the model’s inference pod doesn’t wipe out hours of evaluation data.

The mention of standardized APIs deserves significant elaboration, as the interface contract between model and evaluator is arguably the most consequential architectural decision in an EaaS system. At minimum, the contract must define a uniform request format (prompt, configuration parameters, any required context), a uniform response format (generated text, token-level probabilities if needed, latency

metadata), and a clear error schema. This contract must be model-agnostic: whether the underlying model is an open-weight model running on local infrastructure, a proprietary model accessed through a vendor API, or a custom-served fine-tune behind an internal gateway, the evaluation service should interact with all of them through the same interface. Achieving this requires a model abstraction layer that normalizes the differences between serving backends, translating each model's native interface into the evaluation system's canonical format. Without this layer, every new model integration becomes a bespoke engineering effort, and the promise of decoupled, reusable evaluation infrastructure breaks down at the first point of contact.

Two design principles should govern the evaluation service's behavior: statelessness and idempotency.

- Statelessness: The evaluation service should maintain no internal state tied to any particular model or evaluation run; all necessary context (the evaluation set snapshot, the rubric version, the model endpoint) should be passed in with each request or resolved from versioned references. This makes the service horizontally scalable, since any instance can handle any request, and eliminates an entire class of bugs where stale internal state produces inconsistent results.
- Idempotency complements statelessness: Submitting the same evaluation request twice should produce the same outcome without side effects such as duplicate records or double-counted metrics. Together, these properties make the system resilient to the retries and partial failures that are inevitable at scale, and they make debugging dramatically simpler since any result can be reproduced by replaying the same inputs.

When the model and evaluator are decoupled services communicating over a network, the security of that communication becomes an explicit architectural concern rather than something handled implicitly by process isolation. Authentication and authorization must ensure that only approved evaluation services can query model endpoints and that only authorized pipelines can access evaluation sets, particularly those

derived from production logs that may contain sensitive or PII-adjacent data. Data in transit between services should be encrypted, and access logs should record which evaluation service queried which model with which evaluation set, creating an audit trail. In regulated industries or high-sensitivity deployments, these requirements may extend to data residency constraints, where evaluation data must remain within specific geographic or network boundaries. Treating security as an afterthought in evaluation architecture is especially dangerous because evaluation sets, by their nature, are designed to probe a model's boundaries, and a leaked evaluation set reveals precisely where those boundaries are.

While the core principle of isolation is non-negotiable, the specific deployment topology that implements it involves genuine trade-offs that teams must navigate. At one end of the spectrum, the evaluation service can run as a lightweight sidecar container alongside each model instance, minimizing network latency but coupling the evaluator's lifecycle to the model's. At the other end, the evaluator can be a fully independent managed platform, shared across multiple model teams, which maximizes reuse and consistency but introduces organizational complexity around ownership, prioritization, and SLAs. Between these poles, a common middle ground is a dedicated microservice per team or product area, offering isolation from the model while avoiding the governance overhead of a centralized platform. The right choice depends on organizational size, the number of models being evaluated concurrently, and how much variation exists in evaluation requirements across teams. What matters is that the choice is made deliberately rather than emerging accidentally from whatever deployment pattern happened to be convenient during prototyping.

Furthermore, this isolation enables Multi-Objective Optimization. Because the evaluation is decoupled, you can run parallel pipelines to test the model against different preference weights without increasing the latency of the primary inference task. Ultimately, this architectural rigor transforms evaluation from a final, manual hurdle into a

continuous, high-throughput engine that drives the safety and capability of the next generation of AI.

To make these architectural components concrete, it is useful to outline a reference topology. At a high level, a mature EaaS deployment consists of four layers.

1. A model serving layer that exposes one or more models behind a uniform abstraction API, normalizing differences between serving backends.
2. An evaluation orchestration layer that manages job scheduling, distributes evaluation items across workers, handles retries and partial failures, and enforces versioning constraints to ensure that each run is tied to specific snapshots of the evaluation set, rubric, and model.
3. A storage layer that persists evaluation inputs, model responses, labels (human or automated), and computed metrics in a queryable, versioned format that supports both real-time dashboarding and historical analysis.
4. A security and governance layer that controls access, encrypts data in transit and at rest, and maintains audit logs.

These layers communicate through well-defined API contracts and can be scaled, updated, or replaced independently.

5.2 Democratizing Evaluation: Low-Code and Intelligent UIs

A significant bottleneck in AI development is the dependency on a small team of engineers for every evaluation run or prompt test. To scale, evaluation infrastructure must provide user-friendly interfaces that allow both technical experts and less-technical stakeholders, such as product managers and subject matter experts (SMEs), to communicate

with the system. For example, when designing eval sets for **Stage 1** (Manually Curated Sets), quick iterations to understand model performance are required and reliance on complex integration and without proper interface creates unnecessary barriers between SMEs and Engineers.

Opening evaluation to a broader set of stakeholders introduces an immediate architectural requirement: role-based access control. Not every user should have the same privileges within the system. An SME reviewing model outputs needs read access to results and the ability to tag responses, but should not be able to modify the evaluation set used for production gating. A product manager may need the ability to launch evaluation runs against pre-approved configurations, but should not be able to alter the underlying rubric or override scoring thresholds without review. The system must define clear permission tiers that map to organizational roles, with approval workflows for actions that carry significant consequences such as modifying production evaluation sets, launching high-cost evaluation runs, or changing metric definitions. Without these guardrails, democratization risks become a source of inconsistency and uncontrolled cost rather than a productivity gain.

Closely related is the need for comprehensive audit trails that cover actions taken through the interface with the same rigor applied to code-based operations. When an engineer modifies an evaluation pipeline through version-controlled code, every change is traceable by default. When a product manager adjusts a temperature parameter through a slider and launches a run, that action must be logged with equal precision: who made the change, what the previous value was, when the run was initiated, and which evaluation set and rubric versions were in effect. Without this, the reproducibility guarantees established in earlier sections erode the moment non-technical users interact with the system. The interface should make audit logging invisible to the user while ensuring that every configuration change and evaluation launch is recorded in a queryable log tied to the same versioning infrastructure that governs evaluation sets and templates.

5.2.1 Unified Configuration and Orchestration

Modern platforms need to allow non-technical users to initiate and manage the entire evaluation lifecycle via a centralized hub. Key capabilities must include:

Hyperparameter tweaking: Guided interfaces allow users to adjust settings like temperature, batch size, or sampling parameters without editing configuration files or using a command line.
Visual workflow builders: Drag-and-drop designers enable users to connect model endpoints, dataset inputs, and evaluation metrics into a cohesive flowchart. This replaces complex code-based orchestration with a visible, structured execution plan.

There is an inherent tension in making evaluation accessible through simplified interfaces. Every abstraction that makes the system easier to use also hides decisions that may matter. A drag-and-drop workflow builder that lets a user connect a model endpoint to an evaluation metric in three clicks might obscure which evaluation set version is being used, whether result caching is active, how the judge model is configured, or what statistical test will be applied to determine significance. The design challenge is not simply to hide complexity but to hide it at the right layers. The interface should present sensible defaults while making critical parameters visible and adjustable, even if most users never change them. One effective pattern is progressive disclosure: the initial view shows only the decisions the user must make (which model, which evaluation set, which metrics), while an "advanced" panel exposes the full configuration for users who need it. This ensures that accessibility does not come at the cost of transparency, and that every run launched through the interface is fully specified even when the user only interacts with a handful of controls.

A powerful enabler of non-technical participation is a curated library of evaluation templates and presets. Rather than asking a product manager to assemble an evaluation pipeline from individual components, the system can offer pre-built configurations that encode best practices for common scenarios: a safety evaluation suite for

conversational agents, a factuality check for retrieval-augmented generation pipelines, a regression test for core capabilities after fine-tuning. Each template should specify the evaluation set, rubric, judge configuration, and metrics as a bundled, versioned package that users can select and run with minimal modification. SMEs can then customize specific parameters (swapping in a domain-specific evaluation set, adjusting a scoring threshold) without needing to understand the full pipeline. Over time, templates that prove effective can be promoted to organizational standards, while teams retain the flexibility to create new templates as evaluation needs evolve. This pattern bridges the gap between the rigor demanded by the infrastructure and the speed demanded by practitioners.

5.2.2 Side-by-Side Analysis and Exploration

To facilitate rapid iteration, infrastructure must present results in an intuitive format that moves beyond raw logs.

Topline metrics: Aggregate statistics presented with an adequate level of detail, such as confidence intervals, color-coding for significance levels and deltas, need to be established for "zoomed-out" summary view of the evaluation results.

Drill-down diagnostics: Interactive tools enable users to "point-and-click" their way from aggregate metrics down to individual responses, allowing them to inspect the qualitative language and reasoning chains behind the scores.

5.2.3 Human-in-the-Loop

Evaluation is a continuous process that requires expert intervention to align model behavior with business goals.

Expert tagging and feedback: GUIs allow SMEs to review model outputs directly and "tag" specific responses for further analysis and work.

For example, there will always be some share of model responses that are being mislabeled by either human raters or LLM-as-a-judge

systems. It is important to include capabilities for experts to flag such responses for either relabeling, or exclusion from down-the-line metrics calculations or further evaluators' training.

In practice, evaluation review is rarely an individual activity. An SME may flag a response as mislabeled, but that judgment may itself need validation before it affects downstream metrics. The interface must support structured collaboration workflows: the ability for one reviewer to propose a label change, for a second reviewer to confirm or dispute it, and for the system to track the resolution. For high-stakes evaluations, the system should enforce minimum agreement thresholds before accepting a label override, mirroring the inter-annotator agreement standards applied during initial data creation. Discussion threads attached to individual evaluation items allow reviewers to document their reasoning, creating a record that informs future rubric refinement. Batch review interfaces, where multiple reviewers work through the same queue of flagged items independently before results are reconciled, further strengthen the reliability of human-in-the-loop processes. Without these collaboration mechanisms, expert review becomes an unstructured process where individual judgments accumulate without consistency checks.

Verdict tuning: Most advanced systems can not only perform evaluations, but also derive conclusions about model performance and deployment readiness. It is important to preserve the ability for humans to intervene with sign-offs, comments, or possibly rewrites when reviewing evaluation decisions.

It is worth emphasizing that the interfaces described in this section are not standalone applications; they are a presentation layer built on top of the orchestration, storage, and analysis infrastructure outlined in Sects. 5.1 and 5.2. When a user launches an evaluation through the GUI, that action translates into a job submitted to the orchestration layer's scheduler. When a user drills down from an aggregate metric to individual responses, that query runs against the versioned storage layer. When an SME tags a response for relabeling,

that tag is recorded in the same metadata store that supports segmented analysis. Designing the interface as a thin layer over well-defined APIs, rather than as a parallel system with its own data stores and execution logic, ensures consistency between what technical and non-technical users see and avoids the common failure mode where the GUI and the programmatic interface produce divergent results for the same evaluation. This also means that any action a user can take through the interface can, in principle, be reproduced through the API, preserving the scriptability and automation capabilities that engineering teams rely on.

5.3 Conclusions

The transition of LLM evaluation from research prototypes to production systems requires purpose-built infrastructure, not ad-hoc scripts bolted onto training pipelines. This chapter has outlined the architectural foundations for that infrastructure across three dimensions.

First, we examined the evaluation lifecycle and the automation requirements at each phase. Evaluation set creation demands moderate automation but critical attention to PII protection, contamination detection, and version control. Template and rubric design remains a fundamentally human endeavor where infrastructure serves primarily as a management and versioning layer. Execution, by contrast, is the most automation-intensive phase, requiring orchestration, retry logic, caching, and hybrid routing between human and automated judges. Analysis closes the loop with metrics computation, regression detection, historical trending, and dashboards that surface actionable insights to the right stakeholders.

Second, we argued that reliable evaluation at scale requires strict architectural isolation between the model and the evaluation framework. The failure modes observed in the NeurIPS Efficiency Challenge (NeurIPS Large Language Model Efficiency Challenge, 2023), from environmental contamination to resource contention,

demonstrate what happens when this boundary is neglected. The Evaluation as a Service paradigm addresses these failures through a four-layer architecture: a model serving layer with a uniform abstraction API, an orchestration layer managing job scheduling and labeling workflows, a versioned storage and analysis layer, and a security and governance layer that spans all components.

Third, we addressed the need to make this infrastructure accessible beyond engineering teams. Democratization through low-code interfaces, template libraries, and progressive disclosure enables product managers and subject matter experts to participate meaningfully in the evaluation process. However, accessibility must be paired with governance: role-based permissions, audit trails, and collaboration workflows ensure that broader access does not compromise the rigor and reproducibility on which the entire system depends.

Across all three dimensions, several principles recur. Versioning is not optional; every artifact, from evaluation sets to rubrics to analysis code, must be immutably snapshotted and traceable. Reproducibility must be enforced by design, not assumed by convention. And cost awareness must inform every architectural decision, from judge model selection to caching strategies, because evaluation infrastructure that is technically sound but economically unsustainable will not survive contact with production realities.

These architectural foundations, however, are not the end of the story. Infrastructure determines whether evaluation can run reliably at scale, but it does not address the equally important question of what to evaluate and how to interpret the results in the context of deployment decisions. The next chapter turns to this question, examining how evaluation outputs translate into the governance frameworks and release criteria that ultimately determine whether a model reaches users.

References

Hendrycks, D., et al.: MMLU (Massive Multitask Language Understanding) (2021)

Kaiyom, F., Ahmed, A., Mai, Y., Klyman, K., Bommasani, R., Liang, P.: HELM Safety: Towards Standardized Safety Evaluations of Language Models (2024)

NeurIPS: NeurIPS Large Language Model Efficiency Challenge (2023)

Wang, Z.: HumanEval. Deepgram. https://deepgram.com/learn/humaneval-llm-benchmark (2024)

A. Robsky et al., *Reliable Evaluations for LLMs and AI Agents*
https://doi.org/10.1007/978-3-032-26749-8_6

6. The Evaluation Flywheel for Continuous Improvement

Alexei Robsky[1] ✉, Liliya Lavitas[2] and Yueqing Wang[3]
(1) Microsoft, Sammamish, WA, USA
(2) Google (United States), Newton, MA, USA
(3) Microsoft, San Francisco, CA, USA

6.1 Evaluations as a Vehicle for Continuous Improvement

6.1.1 The Intentionality of System Design

Evaluations do not inherently facilitate model growth unless they are intentionally designed to do so. As discussed in Chaps. 3 and 4, a simple "verdict" or binary pass/fail score provides a summary judgment but fails to serve the core purpose of assessment: providing a roadmap for iteration. If an evaluation pipeline merely acts as a gatekeeper, it yields a static scoreboard that informs engineering teams that a model is failing but offers no diagnostic data regarding the underlying failure mechanisms or how to engineer a solution.

Moreover, Chap. 4 has established that the transition from simple chatbots to autonomous agents fundamentally changes what evaluation must capture: moving from "single-turn" snapshots to "multi-turn trajectories." To act as a vehicle for growth, the evaluation system must be integrated into the development lifecycle from day one, evolving from a post-hoc auditing tool into a high-fidelity diagnostic vehicle that continuously maps failure modes to specific architectural interventions.

6.1.2 Essential Properties for Diagnostic Evals

To power a diagnostic flywheel, evaluations must possess specific technical properties that allow engineering teams to trust, interpret, and act on the signals they receive. Without these properties, the feedback loop breaks down, leading to optimization based on statistical noise rather than genuine capability improvement

- **Reproducibility and Stability**: Reproducibility is the bedrock of scientific inquiry. However, the machine learning field has historically struggled with a reproducibility crisis due to the inherent non-determinism of generative outputs, extreme sensitivity to hyperparameters, and environmental differences across computational hardware. When an experimental process does not yield consistent conclusions across different environments, it becomes mathematically impossible to determine if a performance increase is the result of a deliberate architectural change or merely the product of a favorable random seed. In 2019, NeurIPS introduced a reproducibility program (Pineaum et al. 2020) to standardize how the community communicates findings, emphasizing that

experimental processes must yield consistent conclusions across different environments.

Reproducibility was categorized into three tiers (Gundersen 2021), which serve as the gold standard for defining evaluation stability in non-deterministic systems:

1. **Outcome Reproducibility**: This represents the most stringent tier. It requires that a reproduced experiment yields the exact same, or adequately similar, computational outcome as the original experiment (Semmelrock et al. 2024) When the outcome is identical, the exact same downstream analysis can be conducted, strongly supporting the original hypothesis. In the context of language models, achieving strict outcome reproducibility is notoriously difficult due to floating-point arithmetic variations and sampling stochasticity.
2. **Analysis Reproducibility**: This tier acknowledges that non-deterministic systems may not produce the exact same raw output sequence. Analysis reproducibility does not require the reproduced experiment to have the identical outcome; however, if the exact same statistical analysis can be applied to the new data sample and leads to the identical interpretation, the experiment successfully achieves this tier (Semmelrock et al. 2024).
3. **Interpretation Reproducibility**: This is the most general and robust requirement for practical application. Neither the raw outcome nor the specific statistical analysis needs to be strictly identical, provided that the broader interpretation of the results leads to the same scientific or engineering conclusion (Semmelrock et al. 2024).

In the context of the Evaluation Flywheel, methods reproducibility when combined with robust infrastructure (see Chap.

5) ensures that when a model improves, it is due to a deliberate architectural or prompting change rather than a random seed.

- **Designed for Diagnosis**: Beyond stability, evaluation sets, templates, and metrics must be explicitly constructed to allow for multi-dimensional data slicing and deep-dive diagnostics. As discussed in Sects. 3.2 and 4.3, isolating rubric dimensions (e.g., separating "efficiency" from "safety") is critical. A monolithic score obscures the root causes of sub-optimal performance. Datasets must be large enough to provide statistical power when broken into granular sub-categories (such as "Negative Sentiment," "Multi-turn extraction," or "Code Generation") yet remain streamlined enough to permit high-frequency testing during the development cycle. Metrics must be highly sensitive to detect subtle regressions before they reach production, and templates must be deconstructable to isolate specific dimensions, separating superficial formatting compliance from underlying factual accuracy. Please refer to Chap. 3 where these concepts were introduced and discussed in detail (Fig. 6.1).

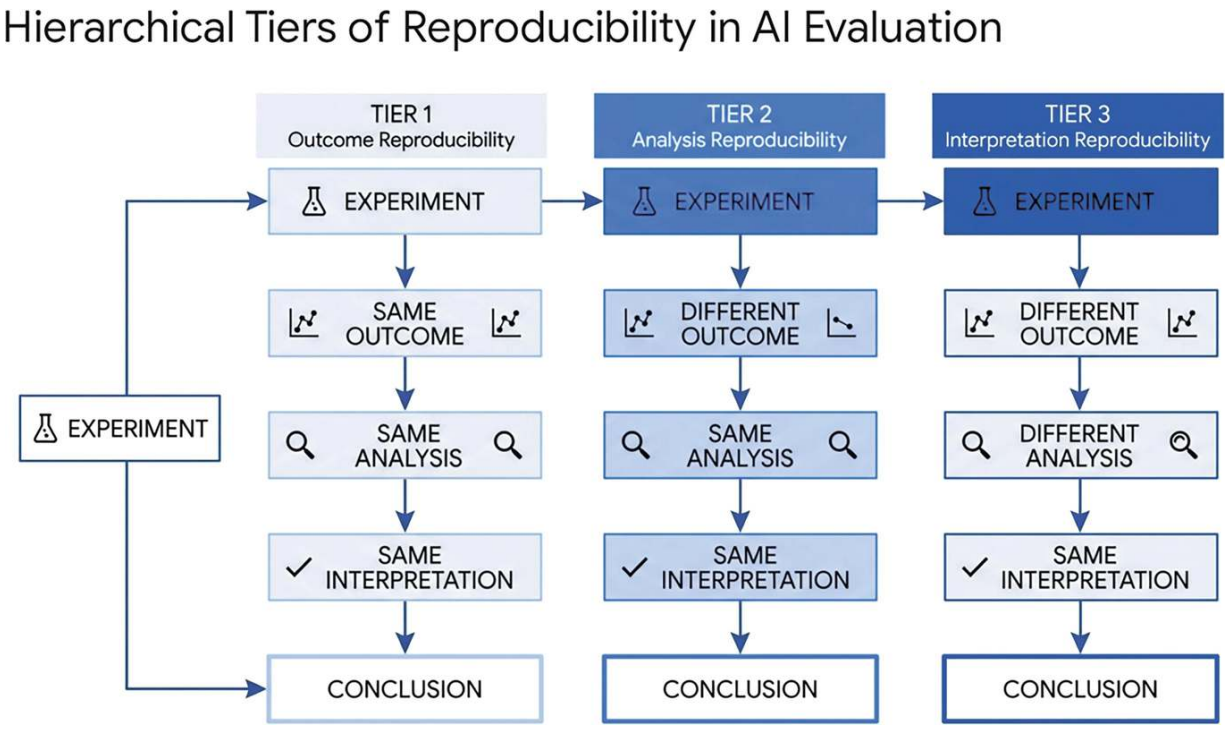

Fig. 6.1 Three tiers of reproducibility

6.2 Flywheel Design: Components and Process

6.2.1 The Mechanics of the Flywheel

The Evaluation Flywheel represents a circular, self-reinforcing process where the analytical output of one development stage becomes the targeted input for the next. As this process iterates, the friction of development decreases, and the momentum of model quality exponentially increases. Instead of relying on randomized prompt modifications, this lifecycle enforces structured engineering discipline, establishing a continuous pipeline to diagnose, measure, and solve systemic problems.

1. **Evaluation Step**: The cycle initiates by capturing a comprehensive snapshot of performance using predefined, version-controlled evaluation sets, templates, and objective metrics. This step relies entirely on the reliable infrastructure (such as automated judge pipelines and robust orchestration) to execute high-throughput inference across baseline datasets without environmental contamination. We discussed in Chap. 5 the critical importance of automation of obtaining Model Responses and Obtaining Labels.
2. **Results Analysis**: Identify "losses" (errors) and weak spots. This is where we look for patterns in the model's failures. In order to establish trustworthy insights, confidence in evaluation results is paramount and this confidence must be rooted in knowledge that eval sets are the representative samples, metrics are statistically sound and labels are reliable. Later in this chapter, we will discuss multiple dimensions of Results Analysis.
3. **Optimization**: Guided by the structured data from the analysis phase, engineering teams apply specific, targeted technical levers to mitigate the identified losses. Adjustments may range from prompt engineering and context retrieval tuning to complex supervised updates. The exact solutions here depend highly on the nature of the losses and engineering capabilities to improve them.
   Validation and Recalibration: Re-run the evaluation. If results are

4. near-perfect, the evaluation has reached saturation. This signals that the "bar" must be raised by introducing more challenging cases or more granular metrics to start the next cycle. If no results improvement is detected, a diagnostic of why is required. Unless the evaluation framework is solid enough to exclude possibility of evaluation not being sensitive enough to detect true model improvements, model improvements will be nearly impossible.

Note: Engineers often run "Micro-Evals" (or "Training" evals) during the Optimization step. These are fast, inexpensive, and highly specific unit tests designed to validate a single, isolated fix before running the computationally expensive full suite. While vital for rapid development velocity, Micro-Evals are not suitable for actual model validation because the model or prompt has been iteratively adjusted against them, the system is inherently overfitted to these specific test cases, rendering the scores invalid for general capability assessment (Fig. 6.2).

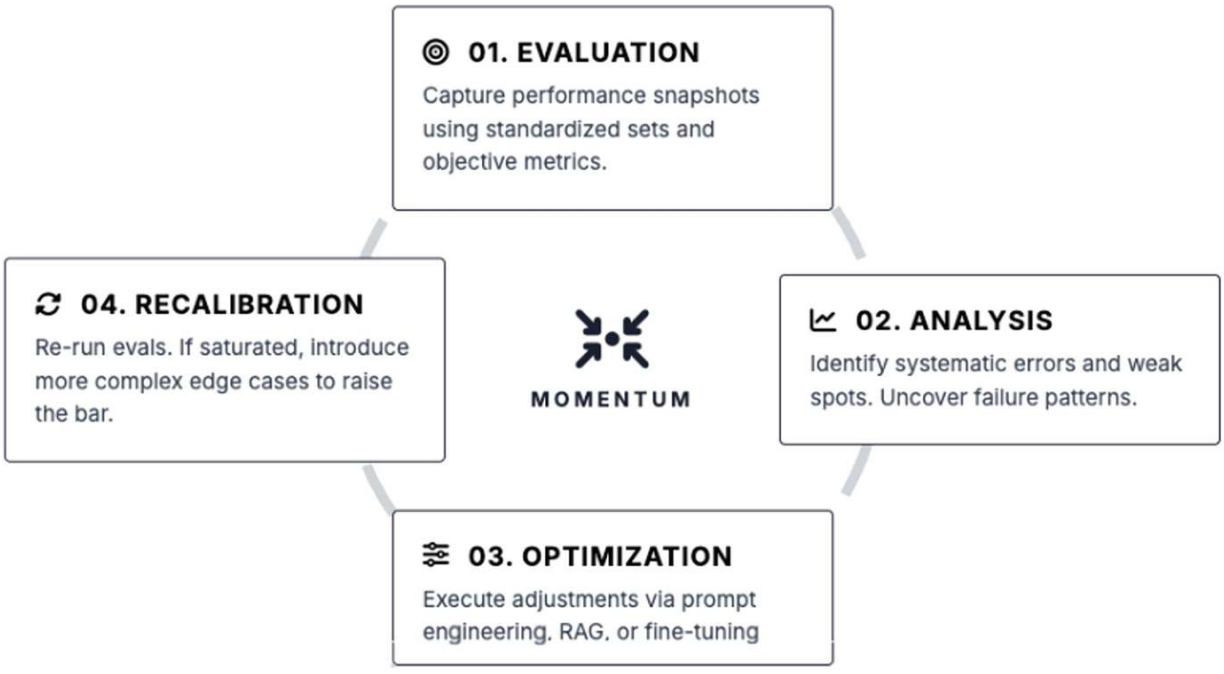

Fig. 6.2 Evaluation Flywheel phases

6.2.2 Losses Analysis: Interpretation and Slicing

To transmute raw numerical data into actionable engineering directives, evaluation results must be sliced and analyzed across three primary dimensions: the "Where," the "What," and the "Why."

- **Dimension 1: Query Slicing (The "Where")**
  By segmenting the query set into subsets (e.g., by language, user persona, task type, or geographic region) practitioners can identify exactly which use cases are prone to failure. However, this segmentation process introduces significant statistical risks that can actively mislead optimization efforts. Practitioners must navigate two critical mathematical traps:

  1. **Simpson's Paradox**: This statistical phenomenon occurs when a distinct trend appears within separate, isolated groups of data but disappears or completely reverses when those groups are aggregated into a single dataset. In the context of model evaluation, relying solely on an aggregate score can mask catastrophic failures in specific, highly sensitive subgroups. For example, consider an application evaluated on an aggregate metric combining both general helpfulness and policy safety. The overall aggregate score might show a marked improvement after an optimization cycle. However, upon slicing the data, one might discover that the model achieved perfect safety scores simply by adopting a hyper-conservative stance and broadly refusing to answer benign questions. Consequently, its helpfulness for complex tasks drastically plummeted. The confounder, which is often the varying distribution of simple versus complex prompts, skews the combined average, creating a dangerous illusion of overall improvement while the actual utility degrades. To neutralize Simpson's Paradox, teams must prioritize subgroup-level metrics and identify confounding variables before making deployment decisions
  2. **The Curse of Dimensionality**: Conversely, the desire to avoid aggregate metrics can lead to extreme over-segmentation. In data science, the Curse of Dimensionality refers to the

exponential growth in volume associated with adding extra dimensions to a mathematical space. As dimensions increase, the available data points become sparsely distributed, causing distances between points to become uniform and rendering clustering algorithms ineffective

In the Evaluation Flywheel, this manifests when teams slice data across too many overlapping categories (e.g., Language + Task + User Persona + Device Type). As the number of dimensional buckets grows exponentially, the sample size within any individual bucket shrinks to near zero. When the data becomes this sparse, statistical concepts like variance and density break down, leading to severe overfitting during the optimization phase because the algorithm attempts to learn from isolated outliers rather than representative distributions. To counteract the Curse of Dimensionality, it is imperative to establish a Minimum Sample Size for any data slice to be considered statistically actionable. Drawing on the Central Limit Theorem, a standard minimum boundary is between 30 and 50 samples per slice. Any subset containing fewer samples must be treated strictly as anecdotal, qualitative evidence rather than a statistically rigorous signal for systemic optimization.

- **Dimension 2: Template Deconstruction (The "What")**
  This involves breaking down the model's response based on atomic characteristics (e.g., Tone, Factuality, Formatting). Such breakdown is greatly facilitated by templates designed for deconstruction as discussed in Sect. 3.2. If the model consistently fails on "Quality," atomic signals collected will tell what dimensions of "Quality" are sub-optimal. It can be Tone, Lack of factual groundingI, Verbosity, and any other rubrics that are intentionally measured via rubric-based template.
- **Dimension 3: Engineering root cause (The "Why")**
  To make losses fully actionable, failures must be mapped to an established taxonomy of engineering root causes. Because generative failures are probabilistic rather than deterministic, they require

specialized categorization paradigms distinct from traditional software debugging. This dimension is typically most actionable for immediate next steps for the engineering team. Engineering experts supporting and developing the model typically have most expertise on all components of the process that lead to response generation and they are best equipped to determine why exactly sub-optimality of response is occurring. They next focus on improving the model components in a targeted way (Fig. 6.3).

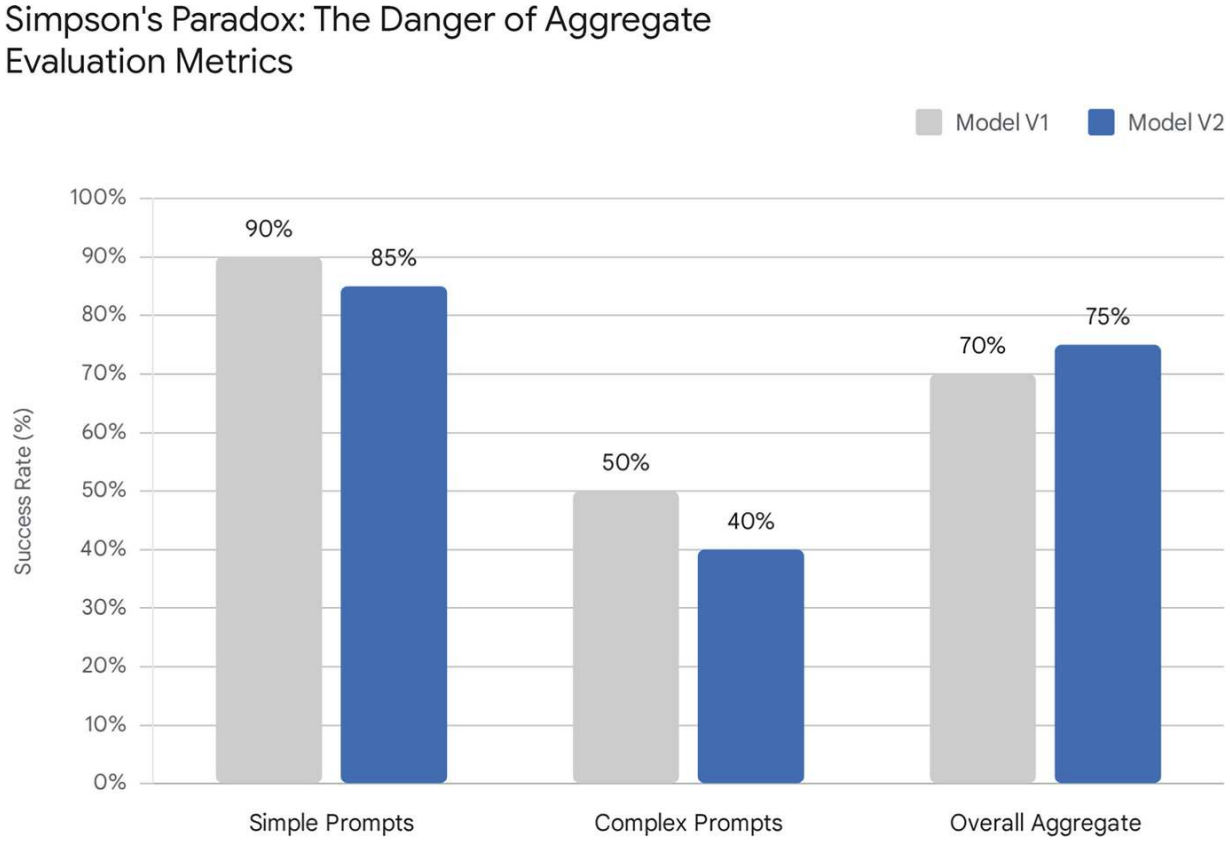

Fig. 6.3 An illustration of Simpson's paradox

6.2.3 Model Optimization

Once the debugging phase has identified the "where," "what," and "why" of model failure, the flywheel moves into the optimization stage. As mentioned above, this is where the engineering team applies specific technical levers to mitigate identified losses. These levers most

commonly include System Instructions improvements, RAG (Retrieval-Augmented Generation) adjustments, and model fine-tuning.

- **System Instructions Improvements**: Iterating on the foundational system prompts, adding dynamic few-shot examples, or enforcing strict step-by-step intermediate processing constraints. While this is the fastest and cheapest lever to pull, it is highly susceptible to prompt brittleness, where fixing one specific edge case inadvertently degrades performance on previously stable queries.
- **Retrieval-Augmented Generation (RAG) Adjustments**: When failures stem from hallucinations or missed context, the optimization must target the external data pipeline rather than the model weights. This involves adjusting document chunking strategies, tuning the embedding models, implementing advanced re-ranking algorithms to surface the most critical snippets, or expanding the context window to ingest more background data.
- **Fine-Tuning**: Post-training the model on curated datasets. Whether utilizing Supervised Fine-Tuning (SFT) or advanced alignment techniques like Direct Preference Optimization (DPO), this is the most difficult and resource-intensive approach. However, it offers the highest degree of stability for specific, repetitive tasks and is essential for fundamentally altering the model's behavioral alignment when prompt engineering reaches its ceiling. Emerging frameworks even allow for automatic differentiation via text, backpropagating natural language feedback to optimize individual components within compound systems. The fine-tuning approach is both the most powerful and a most dangerous path, since low-quality training data, or training data collected with biases might have catastrophic effects on the overall model performance.

6.3 Relationship Between Evaluation Flywheel and A/B Testing

The rigorous, offline Evaluation Flywheel does not replace traditional online A/B testing; rather, both are indispensable, complementary

stages of the modern Model Readiness Lifecycle. Relying solely on one methodology introduces severe, often catastrophic blind spots into the deployment pipeline (Table 6.1).

Table 6.1 Comparison of A/B testing and model evaluations

Feature	Offline Evaluations Flywheel	Online A/B testing
Environment	Controlled setting	Real-world, noisy production environments
Risk	Zero risk to users	High potential impact on user experience and brand reputation
Metric focus	Objective model quality: Accuracy, Safety, etc.	Subjective user behavior (Retention, Click-through rates, Session depth)
Scope	Best for testing new capabilities	Best for optimizing the user experience of existing, validated features
Iteration velocity	High-velocity iterations	Slower iterations requiring statistical significance over live traffic (days to weeks)

Why you need both:

1. **Safety**: Evals catch "catastrophic" errors (e.g., hallucinations or bias) before a single user sees them.
2. **Signal**: A model might be "objectively" better in an Eval (more accurate) but "subjectively" worse in an A/B test (users find the more accurate answers too long or verbose).

Both stages are required to bridge the gap between objective capability and subjective utility. The offline Flywheel acts as an uncompromising safety net, catching toxic regressions, formatting failures, and factual hallucinations in a zero-risk environment before a single user is exposed to them. However, a model might score objectively higher in an offline evaluation, for instance, by providing exhaustively accurate and highly detailed answers, but perform subjectively worse in an online A/B test because live users find the increased verbosity annoying, leading to an immediate drop in engagement metrics. This mismatch by itself presents a great opportunity for improvements to evaluations and

a deep dive to understand reasons for disagreement between evaluations and A/B testing, which can lead to improved understanding of the gap between what model developers optimize for and what typical users reward.

Furthermore, proactive online tracking focuses on metrics like acceptance rate (how often users accept generated output without edits) and implicit rejection rates. Explicit feedback, such as thumbs-down ratings, is rare; the true signal of friction often lies in actions like a user reverting an applied change or completely deleting an AI-generated draft. A/B testing reveals these nuanced user preferences and behavioral realities that offline automated metrics simply cannot capture.

Industry standard follows a sequential pipeline: Flywheel Iteration → A/B Testing → Production Launch. While the Flywheel catches "catastrophic" errors like safety violations, A/B testing reveals if an "objectively" better model is "subjectively" worse for users (e.g., being too verbose).

6.4 Flywheel Must Be Dynamic

Evaluation sets must evolve alongside the model. Infrastructure should support a "turn-key" refresh, incorporating new data samples, particularly "real failures" (see Chap. 4) sourced from bug trackers or manual testing without re-engineering the test harness. To remind the readers in Chap. 5, we have rediscussed that evaluation infrastructure must be decoupled from the model environment to avoid "eval-drift" or environmental contamination. Shifting toward an Evaluation as a Service (EaaS) paradigm ensures the "ruler" (the framework) remains consistent even as the "subject" (the model) changes. Following these principles is critical for establishing a dynamic flywheel.

6.4.1 Turn-Key Eval Sets Refresh

As referenced in previous chapters, evaluation sets must be highly reflective of the current stage of model development. The underlying

infrastructure must support a “turn-key” evaluations refresh, which requires inclusion of new data samples, new evaluations templates and new metrics into the flywheel without manual re-engineering of the test harness.

To achieve this automation, advanced pipelines employ dynamic validation strategies. Instead of utilizing a static, hard-coded set of examples for automated judges, modern systems leverage active learning protocols. By dynamically retrieving examples from a vector database that are semantically relevant to the specific instance currently being evaluated, the system ensures high contextual accuracy. Furthermore, prioritizing the retrieval of historical examples where human labels previously differed from automated predictions focuses the evaluation directly on edge cases and areas of potential misalignment. Weighting these retrievals by label recency ensures the evaluation criteria continuously adapt to the latest human preferences.

Additionally, as model capabilities increase, the metrics themselves must undergo capability-driven updates. Simple validation checks must mature into nuanced rubrics evaluating advanced logic and multi-step adherence to ensure the evaluation ceiling is continuously raised.

6.4.2 Rating Reliability Monitoring

A dynamic flywheel is only as valuable as the signal it produces. Continuous monitoring is required to explicitly decouple shifts in actual model performance from undetected degradation in labeler quality. Whether utilizing human annotators or automated frameworks, persistent quality audits are mandatory to detect rater drift, prompt fatigue, or rubric misinterpretation before corrupt data permanently pollutes the optimization phase. In Sect. 3.4, we have discussed in detail the challenges and solutions for maintaining high evaluation quality. Here we will briefly summarize two key aspects most relevant in the context of Evaluation Flywheel:

- **The Nuances of Automated Judges**: While deploying automated judges dramatically increases evaluation throughput and lowers per-unit costs, these systems exhibit well-documented systematic biases

that can severely skew results if left unmonitored. Automated judges frequently display *positional bias* (disproportionately favoring the first or last response in a pairwise comparison), *verbosity bias* (falsely equating longer, more loquacious responses with higher factual quality), and *self-enhancement bias* (preferentially scoring outputs generated by models sharing their own architectural lineage). To combat these systemic flaws, evaluators must be carefully calibrated using advanced prompting protocols. This involves forcing the judge to articulate its rationale against a strict rubric before issuing a final score, and regularly evaluating the judge against pre-graded anchor examples established by human domain experts.

- **Measuring Inter-Rater Reliability**: To mathematically quantify the reliability of the evaluation signal, engineering teams must utilize robust inter-rater reliability (IRR) metrics. Simple percentage agreement is entirely insufficient because it fails to account for instances where raters agree purely by random chance. The industry standard for measuring this true alignment is Cohen's Kappa, which provides a statistically sound measure of agreement between two raters (e.g., comparing an automated judge against a human expert benchmark). When the evaluation framework expands to include more than two raters, such as a jury ensemble of multiple models and human reviewers, the statistical methodology must be upgraded to use Fleiss' Kappa to accurately capture multi-rater consensus across categorical data. These metrics are not the only ones widely available and cited in the literature, but serve as a "Golden standard" and a solid starting point.

6.4.3 Agent-Specific Evaluation Flywheel

An agentic flywheel, unlike those for conventional LLMs, operates over multi-step trajectories, such as sequences of decisions, tool calls, and environmental interactions leading to an outcome. The model must not just produce a good response but make a series of good decisions in sequence. This fundamentally changes the structure of what the

flywheel is optimizing over and introduces challenges that simply don't exist in single-turn settings.

As a result, the most significant technical difference is credit assignment. Unlike conventional RL for LLM post-training in static, single-turn tasks, training LLM agents in interactive environments faces particular challenges. Rewards are typically sparse and delayed, complicating credit assignment to intermediate actions; trajectories are long and non-Markovian at the token level, with each step consisting of chain-of-thought and an executable action, inflating variance when credit is pushed to individual tokens; and environments are non-stationary, open-ended, and often come with unverifiable rewards.

In other words, the flywheel knows the agent succeeded or failed at the end of a long task, but not which of the twenty decisions made along the way actually caused that outcome. Standard RLHF sidesteps this entirely because there is only one decision to attribute. The agentic flywheel requires either process reward models (PRMs, Xi et al. 2025) that score intermediate steps, or methods like iStar (Liu et al. 2025) that jointly train an implicit PRM with the policy model using trajectories collected online, generating implicit step rewards for each action by measuring its relative preference over the previous policy snapshot. This provides dense feedback to guide exploration while staying coarse enough to keep variance under control.

The data collection, correspondingly, is no longer just collecting judgments on outputs. Instead, it curates rich behavioral traces, including sequences of observations, reasoning steps, tool invocations, and environmental responses, which are both harder to gather and much more expensive to annotate or filter.

The conventional flywheel is fundamentally a preference learning problem: make the model's outputs more aligned with what humans judge as good. The agentic flywheel is a sequential decision-making problem: teach the model to make better choices under uncertainty, across time, in environments it affects through its own actions. The two share architectural DNA. Both rely on rollouts, reward signals, and iterative policy updates, but the agentic case reintroduces the full

complexity and uncertainty of reinforcement learning that the single-turn RLHF paradigm had managed to simplify.

6.5 Conclusion

The transition of LLM evaluation from research prototypes to production systems requires purpose-built infrastructure, not ad-hoc scripts. Across this chapter, we have examined the mechanics required to turn static measurements into a well-functioning Flywheel engine for continuous improvement.

First, we examined the mechanics of the evaluation flywheel as a diagnostic vehicle for continuous improvement. Unlike static checkpoints, the flywheel operationalizes evaluation into a circular, self-reinforcing process of discovery, measurement, and targeted mitigation. This lifecycle shifts the focus from simple pass/fail verdicts to granular failure patterns, ensuring that every engineering intervention is grounded in objective data.

Second, we argued that a diagnostic flywheel is only as reliable as the stability and interpretability of its measurement signal. We explored the three tiers of reproducibility: Outcome, Analysis, and Interpretation as the framework for maintaining scientific rigor in non-deterministic systems. We also addressed the mathematical hazards of multi-dimensional data slicing, demonstrating how Simpson's Paradox and the Curse of Dimensionality can mask regressions or lead to overfitting if not countered by strict minimum sample size requirements and subgroup-level monitoring.

Third, we addressed the fundamental shift required for agentic evaluation. The architectural foundations of previous chapters form the backbone of this feedback loop. By enforcing immutable versioning of every artifact and maintaining cost awareness through strategies like judge model tiering, organizations can ensure their evaluation systems are technically sound and economically sustainable. Infrastructure determines if evaluation can run at scale, whereas the Evaluation Flywheel ensures that scale drives meaningful progress.

References

Gundersen, O.E.: The fundamental principles of reproducibility. Philos. Trans. A Math. Phys. Eng. Sci. **379**, 20200210 (2021)

Liu, X., et al.: Agentic reinforcement learning with implicit step rewards. arXiv preprint arXiv:2509.19199 (2025)

Pineaum, J., et al.: Improving reproducibility in machine learning research (a report from the NeurIPS 2019 reproducibility program). https://arxiv.org/abs/2003.12206 (2020)

Semmelrock, H., et al.: Reproducibility in machine learning-based research: overview, barriers and drivers. https://arxiv.org/abs/2406.14325 (2024)

Xi, Z., et al.: Agentprm: process reward models for LLM agents via step-wise promise and progress. arXiv preprint arXiv:2511.08325 (2025)

Afterword

The mechanics of evaluation are rapidly being automated away. Agent coding can already scaffold evaluation pipelines, generate test cases, write autoraters, and execute benchmark runs with minimal human intervention. As these capabilities mature, much of what this book has treated as careful engineering will simply be delegated to an agent: instrumenting inference, managing prompt templates, wiring up scoring logic. The implementation details that once demanded weeks of work will compress into a prompt.

But automation does not diminish the importance of evaluation; it raises the stakes for understanding. When a human engineer builds an evaluation by hand, their judgment is embedded in every decision: which examples to include, what success looks like, how edge cases should be handled. When an agent builds the evaluation, those same decisions still get made, just faster, at greater scale, and largely out of sight. The output is only as good as the specification provided to the agent, and writing a good specification requires the same foundational knowledge as writing a good evaluation. An agent instructed to "generate diverse evaluation cases for a customer support assistant" will do exactly that and produce something that looks comprehensive. Whether it actually tests the right things, at the right difficulty distribution, with the right success criteria, depends on the evaluator's grasp of the principles that no amount of scaffolding code can substitute for. Worse, errors in those principles propagate silently across thousands of generated examples rather than surfacing visibly during manual construction.

This points to a subtle but important shift in where human expertise needs to focus. Practitioners who once spent significant time on implementation will increasingly spend that time on evaluation of their evaluations: assessing whether the examples are representative, whether the metrics measure what they claim to, whether the coverage is broad enough to catch the failure modes that matter. Meta-evaluation, once a niche concern, becomes the core skill. The question

is no longer “did we build the evaluation correctly?” but “is this evaluation extracting the signal we actually need?” And as evaluation scales, so does the challenge of evaluating the evaluators themselves. Inter-rater reliability, calibration drift, and annotator fatigue become first-order concerns when human judgment is applied not to dozens of examples but to thousands, across teams and time zones.

The challenge compounds from the other direction as well. As agents grow more capable, tackling longer horizons, more ambiguous tasks, and multi-agent coordination, the evaluation problems themselves grow harder. Both the object of evaluation and the means of building evaluations are shifting simultaneously, which makes principled understanding not just useful but essential. And the principles themselves will not stand still. New model capabilities, from multimodal reasoning to real-time tool use to long-context processing, will surface failure modes that don’t map cleanly onto today’s frameworks. The field’s vocabulary and taxonomies will keep expanding. What remains constant is the discipline of asking the right questions: what behavior matters, how we would recognize it, and what evidence would change our mind.

This book has tried to build that discipline from the ground up. The foundations of measurement give structure to what would otherwise be subjective impressions. Extending those foundations to agents introduces the complexities of autonomy, environment interaction, and compounding error. Operationalizing evaluation at scale turns those ideas into systems that can run continuously and inform real decisions. These are not separable concerns. A metric that is theoretically sound but operationally fragile tells you nothing. An infrastructure that runs flawlessly but measures the wrong thing is worse than no infrastructure at all. The chapters form a single argument: that rigorous evaluation requires getting the principles, the methods, and the systems right together.

Answering these questions requires internalizing the principles of good evaluation design thoroughly enough to audit work that an agent produced. That is a judgment call that no amount of automation

removes. But for practitioners who have built that foundation, the shift is not a threat. It is a multiplier. The same understanding that once let you build one careful evaluation now lets you direct an agent to build hundreds, with the confidence to know whether those hundreds are actually worth running. The tools will keep changing. The question of whether your system actually works never will.