ChatGPT

Chapter 1 – Fundamentals & Lifecycle of LLM Application Evaluation

1 Why evaluation matters

Large language models (LLMs) have moved from research curiosities to business-critical components of chatbots, content-generation tools and decision-support systems. Without systematic evaluation, organisations risk shipping unstable products that hallucinate facts, expose sensitive information or behave unpredictably when faced with novel inputs. Observability experts emphasise that LLMs involve **intricate chains of operations**, and monitoring those chains allows teams to **understand and optimise each step** [1]. Evaluation directly influences:

- **Business impact and trust** – A customer-service bot giving incorrect legal advice can expose a firm to legal claims. A research assistant that invents citations undermines user confidence and harms brand reputation. Consistent evaluation mitigates these risks by flagging hallucinations and style drift early. SigNoz notes that observability supports **quality assurance** by detecting hallucinations or inaccuracies, **performance optimisation** by spotting bottlenecks and **cost control** by tracking token usage [1].
- **Risk mitigation and compliance** – LLM outputs may leak personal data or break content-policy rules. Monitoring and evaluation pipelines help detect policy violations and enable content filtering. The SigNoz guide highlights that observing token usage and resource demand allows firms to **optimise resource allocation**, improving **cost-effectiveness and system throughput** [1].
- **Iterative improvement** – Evaluation is not a one-time event. Models must be re-tested after prompt changes, new retrieval strategies or fine-tuning. A robust evaluation regime helps teams quantify improvements and catch regressions before releasing to production.

2 Challenges unique to evaluating LLM outputs

Traditional machine-learning evaluation—where models are measured once on a static test set—does not transfer cleanly to generative AI. LLM evaluation faces several unique challenges:

2.1 Non-deterministic behaviour and context dependence

LLMs are **probabilistic** by design; identical prompts can yield different outputs. Context matters: responses depend on previous turns, system prompts and retrieved documents. The Coralogix guide notes that **accuracy and reliability** are a common challenge because LLMs can produce **"hallucinations"**, making it critical to manage and evaluate outputs for misleading content [2]. Another challenge is **non-determinism**; the same model may respond differently across sessions, complicating reproducibility [3].

2.2 Complex failure modes

LLMs fail in many ways. Common failure modes include:

Failure mode	Description	Example
Hallucination	Generating false or fabricated information	A travel assistant invents a non-existent train route.
Off-topic or irrelevant responses	Misunderstanding user intent or drifting away from the topic	A finance bot answering a question about stock prices with a poem.
Shallow reasoning	Failing to perform multi-step reasoning or basic arithmetic	An assistant incorrectly sums numbers in a spreadsheet.
Prompt injection/ jail-breaks	Users exploit system prompts to extract hidden instructions	A user instructs the model to ignore safety guidelines and reveal confidential information.
Bias and toxicity	Generating harmful or biased content due to training data biases	A hiring assistant recommending candidates based on gender or ethnicity.
Tool misuse	Selecting the wrong tool or API when acting as an agent	A retrieval-augmented system calling a weather API instead of a finance API.

Some of these errors are subtle. For instance, a retrieval-augmented generation (RAG) system may return relevant documents yet misattribute information, or an agentic pipeline may propagate a small error through multiple steps. Because failure modes are context-dependent, evaluation must look at entire traces rather than isolated outputs.

2.3 Lack of standardised metrics

Unlike classification tasks, there is no single metric to quantify generative quality. The AIMultiple review warns that relying on one metric like perplexity is inadequate; teams should use multiple metrics capturing **fluency, coherence, relevance, diversity and context understanding** [4]. Human evaluation remains essential but is **subjective and costly** [5]. Automated metrics may suffer from **biases**, such as preferring longer responses (attention bias) or favouring initial options (order bias) [6]. Consequently, evaluation strategies must combine automated metrics with structured human reviews, and they must monitor for biases and adversarial behaviour.

3 Lifecycle approach: from prototype to production

A holistic evaluation strategy spans the entire development lifecycle:

1. **Ideation and requirement definition** – Identify the intended use case and user personas, define success criteria (e.g., factual accuracy, tone) and clarify unacceptable behaviours. At this stage, draft a rubric describing what constitutes a good response and what failures to watch for. This rubric will guide annotation later.

2. **Rapid prototyping and offline evaluation** – Build a minimal prototype and test it on internal test prompts and synthetic data. Use automated metrics (e.g., BLEU, ROUGE, semantic similarity) and LLM-judge approaches (ask a strong model to grade outputs) to get initial quality signals. SigNoz recommends generating **responses to a set of test prompts**, applying automated metrics and routing low-scoring outputs for human review [7].

3. **Red-team testing and user feedback** – Expose the prototype to adversarial prompts and subject-matter experts to uncover failure modes (e.g., prompt injections, biases). Incorporate early user feedback to adjust prompts, retrieval filters and tool usage. Using **A/B tests** or canary deployments helps measure impact on user experience in a controlled way.

4. **Canary / A–B rollout** – Before full deployment, release the system to a small percentage of real users or sample conversations. Evaluate performance against baseline models and metrics. This stage often reveals real-world issues (e.g., ambiguous queries, heavy dialects) that synthetic tests miss [8]. Carefully monitor safety and revert if regressions appear.

5. **Production monitoring and continuous evaluation** – Once live, instrument the application (see §4) to collect traces, metrics and user feedback. Schedule automated evaluations on new versions, maintain evaluation datasets and implement quality guardrails (e.g., toxicity detectors) to prevent unsafe outputs. Nightly or weekly evaluation jobs ensure that updates do not degrade performance.

6. **Iterative improvement** – Feed evaluation results back into development. Failure analysis (see §5) guides which prompts, retrieval strategies or model variants need refinement. Repeat the cycle, updating evaluation datasets and rubrics as the system evolves.

This lifecycle underscores that evaluation is a continuous process rather than a one-off event. Coupling offline testing with **real-world evaluation** improves generalisation [8].

4 Instrumentation and observability

Effective evaluation requires comprehensive instrumentation to capture what happens inside an LLM application. Observability goes beyond simple monitoring: it provides fine-grained traces of the request–response cycle, enabling root-cause analysis and performance optimisation.

4.1 Traces, spans and sessions

According to the Datadog documentation, a **span** represents a unit of work in an LLM application and is the building block of a **trace** [9]. Each trace captures the work involved in processing a single request and consists of one or more nested spans. A root span marks the beginning and end of a trace [9]. A span records attributes such as:

- **Name, start time and duration**;
- **Error information** (type, message and traceback);
- **Inputs and outputs** (prompts and completions);
- **Metadata** like model temperature or maximum token limits;
- **Metrics** such as input and output token counts [10].

Span kinds provide additional context: LLM spans correspond to direct model calls, **workflow spans** group together an LLM call with supporting operations, **agent spans** represent a dynamic sequence of operations, and **tool**, **task**, **embedding** and **retrieval** spans denote calls to external tools, data preprocessing steps, embedding functions and vector database searches respectively [11]. Understanding these categories helps engineers design more targeted tests; for example, evaluating RAG systems requires inspecting retrieval spans to ensure relevant documents are fetched.

A **session** groups together multiple traces for a user conversation, enabling analysis across multi-turn interactions. Instrumentation libraries such as OpenTelemetry, OpenLLMetry or vendor tools like Datadog and SigNoz automatically create spans and traces and export them to a backend for visualisation. The SigNoz guide notes that **instrumenting code to create spans for each significant operation**, collecting metadata such as prompt details and token counts and **visualising traces to understand the flow** are key steps [12]. This telemetry allows engineers to compute metrics like latency, token usage and error rates, and to attribute failures to specific components (e.g., retrieval vs. generation).

4.2 Metrics and logging

Observability also involves capturing key performance indicators:

- **Latency and throughput** – Track response times for each span and overall conversations. Identify slow stages, such as retrieval or tool calls, and optimise them.
- **Token usage and costs** – Log the number of input and output tokens per call to understand resource consumption. This supports cost control and alerts when token counts exceed thresholds.
- **Error rates and quality checks** – Record errors such as invalid tool calls, exceptions in code or failed safety checks. Tag spans with evaluation scores (e.g., factual accuracy, topic relevance) to correlate quality issues with system components.
- **User feedback and satisfaction** – Integrate rating widgets or classification systems (e.g., thumbs-up/down) to collect human feedback. Use this data to adjust prompts and training.

LLM observability platforms provide dashboards for these metrics, support alerts and allow drill-downs into individual traces. SigNoz emphasises the need for **clear metrics**, **comprehensive logging** (including prompts, outputs and post-processing steps) and **real-time monitoring** [13]. Integrating observability with CI/CD pipelines ensures that new releases meet quality standards before deployment.

4.3 Session tracking and redaction

For safety and privacy, sessions should be tagged with user identifiers and contextual metadata, but sensitive data must be redacted or anonymised. Tools like LangChain's `Callbacks` or OpenTelemetry interceptors allow injection of custom logic to scrub personally identifiable information. When evaluating multi-turn conversations, track memory context and ensure that models do not leak past messages in their outputs.

5 Introduction to error analysis and failure categorisation

Evaluation metrics quantify performance, but they do not explain *why* models fail. **Error analysis** is a systematic process for identifying, categorising and prioritising failures. Alex Strick van Linschoten's blog, summarising the Hamel–Shreya evaluations course, describes a five-step loop for error analysis [14]:

1. **Create an initial dataset** – When production logs are scarce, bootstrap your evaluation with synthetic data. Generate around **100 inputs** covering different user personas, features and query complexities [15]. Combining human-written and LLM-generated prompts ensures diversity.

2. **Open coding** – Run the queries through your system and collect full traces (including intermediate tool calls). Read each trace and write descriptive notes about what went wrong.

Importantly, let categories emerge from the data rather than imposing preconceived labels [16]. Focus on the earliest observable error in complex pipelines and ignore root-cause analysis at this stage [17].

3. **Axial coding (clustering)** – Group similar notes to create an **emergent failure taxonomy** [18]. Use your judgment to refine and merge categories; automated clustering can help but human review is essential. Binary definitions (yes/no) make categories clearer [19].

4. **Iterate and expand** – Label more traces and continually refine your taxonomy. It is normal for categories to evolve during annotation. Keep looping between open coding and clustering until you reach **saturation** and aren't learning anything new [20].

5. **Prioritise and act** – Once failures are categorised, prioritise them by frequency and severity. Address high-impact clusters first—e.g., if most errors stem from retrieving the wrong document, improve your retrieval filter or vector store indexing.

5.1 Common pitfalls

Error analysis is labour-intensive and can be undermined by several pitfalls:

- **Narrow sampling** – If the synthetic dataset covers only a few scenarios, you will miss important failure modes [21]. Vary persona, query type and complexity.
- **Skimping on annotation** – Half-heartedly coding a few examples leads to superficial insights [22]. Allocate adequate time to read and annotate traces.
- **Automating too early** – Delegating clustering to algorithms before understanding the data can miss nuanced patterns [23].
- **Skipping iteration** – You must revisit and refine categories; otherwise early misconceptions become entrenched [24].
- **Excluding domain experts** – For specialised domains (e.g., medical advice), involve subject-matter experts to interpret errors [25].

5.2 Error categorisation examples

Suppose you are building a legal assistant. After running 100 synthetic queries, you identify these categories:

- **Hallucinated statute** – The assistant cites statutes or cases that do not exist. Fix: integrate an authoritative legal database and cross-check citations.
- **Missing citations** – The model summarises case law but fails to cite sources. Fix: adjust prompts to instruct the model to provide citations; evaluate retrieval spans for relevance.
- **Confused jurisdictions** – The assistant mixes UK and US law in its answers. Fix: include jurisdiction metadata in the prompt and retrieval filter.
- **Unsafe advice** – The model offers actionable legal advice rather than general information. Fix: implement guardrails to instruct the model to provide informational content and include a disclaimer.

Plotting a confusion matrix of failure categories versus user personas helps prioritise improvements (e.g., non-English speakers may experience more confusion). Documenting root causes (prompting, retrieval, tool selection) allows targeted fixes.

6 Putting it all together: an example pipeline

Consider a simple RAG-based knowledge assistant. Its pipeline comprises: user query → retrieval of relevant documents → summarisation via an LLM → tool calls (e.g., a calculator) → final answer. Implementing evaluation involves:

1. **Instrumentation** – Use OpenTelemetry or an LLM observability SDK to generate traces for each query. Each call to the retrieval function, summarisation LLM and tool (e.g., calculator) becomes a span, capturing inputs, outputs, token counts and latencies [26].

2. **Evaluation dataset** – Create a test set of queries covering different topics and difficulty levels. Provide reference answers or rubric items (e.g., factual correctness, citation requirement). For example, a query might ask, "What is the deadline for filing a VAT return in the UK?" The reference answer notes the deadline for VAT returns (usually one month and seven days after the accounting period ends). An evaluation script computes semantic similarity between the model's answer and the reference, checks whether the retrieved documents contain the answer and uses an LLM-judge to grade clarity.

3. **Metrics** – Combine automated metrics (exact match, ROUGE) with LLM-as-judge scoring. Evaluate per-component: retrieval hit rate, summarisation factuality and overall response quality. Where possible, measure **cross-modal metrics**; for multimodal tasks, metrics like Recall@K and mean Average Precision evaluate cross-modal retrieval [27], BLEU or CIDEr evaluate caption quality, and modality alignment scores assess how well information from different modalities is integrated [28].

4. **Error analysis** – After running the test set, cluster failures. You may find that 30 % of errors arise from irrelevant retrieval; 10 % from hallucinated details; 5 % from wrong tool calls. Prioritise improving the retrieval filter and adjusting prompts to reduce hallucinations.

5. **Monitoring in production** – Deploy the assistant in a canary environment. Collect traces for live queries and compute rolling metrics. Compare new metrics with baseline values. Set up alerts for spikes in hallucination rate or latency. Provide a user feedback mechanism (e.g., thumbs up/down) to gather qualitative data. Use that data to refine prompts and update evaluation datasets.

7 Conclusion

Evaluation is foundational to building reliable and trustworthy LLM applications. Because LLMs are non-deterministic and context-dependent, they demand a **multidimensional evaluation strategy** combining automated metrics, structured human reviews and continuous monitoring. Observability—through traces, spans and sessions—supplies the data needed to analyse performance and diagnose failures [26]. Systematic error analysis transforms raw data into actionable insights, guiding iterative improvements [14]. By embedding evaluation throughout the development lifecycle and aligning it with business objectives, teams can mitigate risks, control costs and deliver high-impact AI applications that users can trust.

---

[1] [7] [12] [13] Understanding LLM Observability - Key Insights, Best Practices, & Tools | SigNoz
https://signoz.io/blog/llm-observability/

2 3 LLM Observability: Challenges, Key Components & Best Practices - Coralogix
https://coralogix.com/guides/aiops/llm-observability/

4 5 6 8 Large Language Model Evaluation in 2025: 5 Methods
https://research.aimultiple.com/large-language-model-evaluation/

9 10 11 26 LLM Observability Terms and Concepts
https://docs.datadoghq.com/llm_observability/terms/

14 15 16 17 18 19 20 21 22 23 24 25 Alex Strick van Linschoten - Error analysis to find failure modes
https://mlops.systems/posts/2025-05-23-error-analysis-to-find-failure-modes.html

27 28 What are some common evaluation metrics for multimodal AI?
https://milvus.io/ai-quick-reference/what-are-some-common-evaluation-metrics-for-multimodal-ai