Debugging and Monitoring LLMs in Production

Abi Aryan

ODSC East, 2025

About Abi Aryan

Abi Aryan is the founder of Abide AI and a machine learning engineer with over eight years of experience in the ML industry building and deploying machine learning models in production for recommender systems, computer vision, and natural language processing—within a wide range of industries such as e-commerce, insurance, and media and entertainment.

Previously, she was a visiting research scholar at the Cognitive Sciences Lab at UCLA where she worked on developing intelligent agents. She has also authored research papers in AutoML, multi-agent systems, and LLM cost modeling and evaluations.

Books:

- LLMOps: Managing Large Language Models in Production, O'Reilly Publications (July 2025)
- GPU Engineering for AI Systems, Packt Publications (Sept 2026)

Agenda

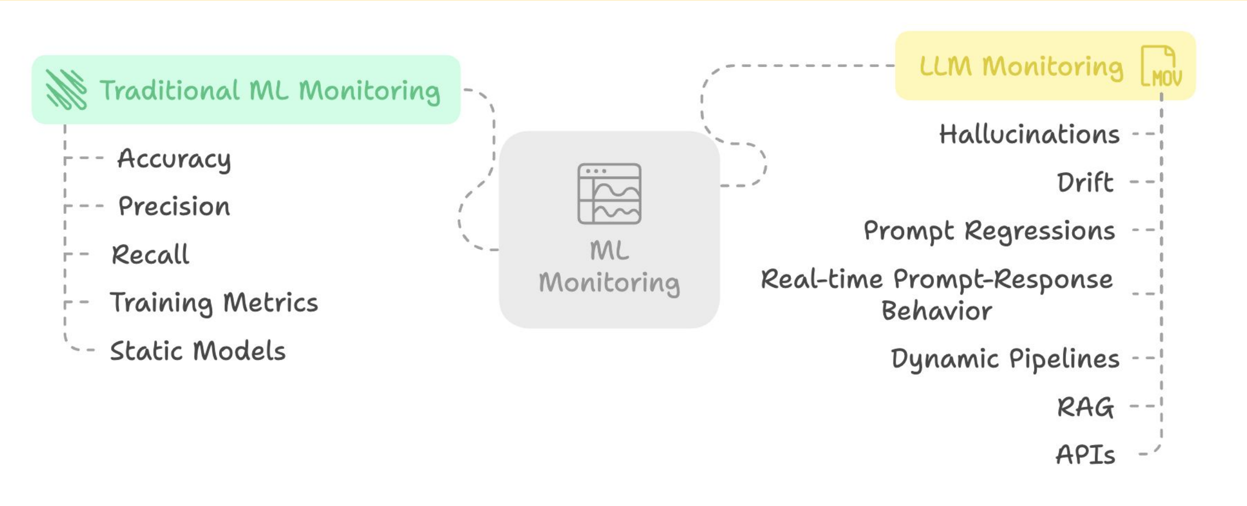
01. Why - Understanding the Challenges
02. What - Monitoring & Debugging Techniques
03. How - Tooling & Feedback Loops

01. Understanding the Challenges

Why “Observability” matters

LLMs are powerful but unpredictable. **Observability ensures control, safety, and performance in real-world use.**

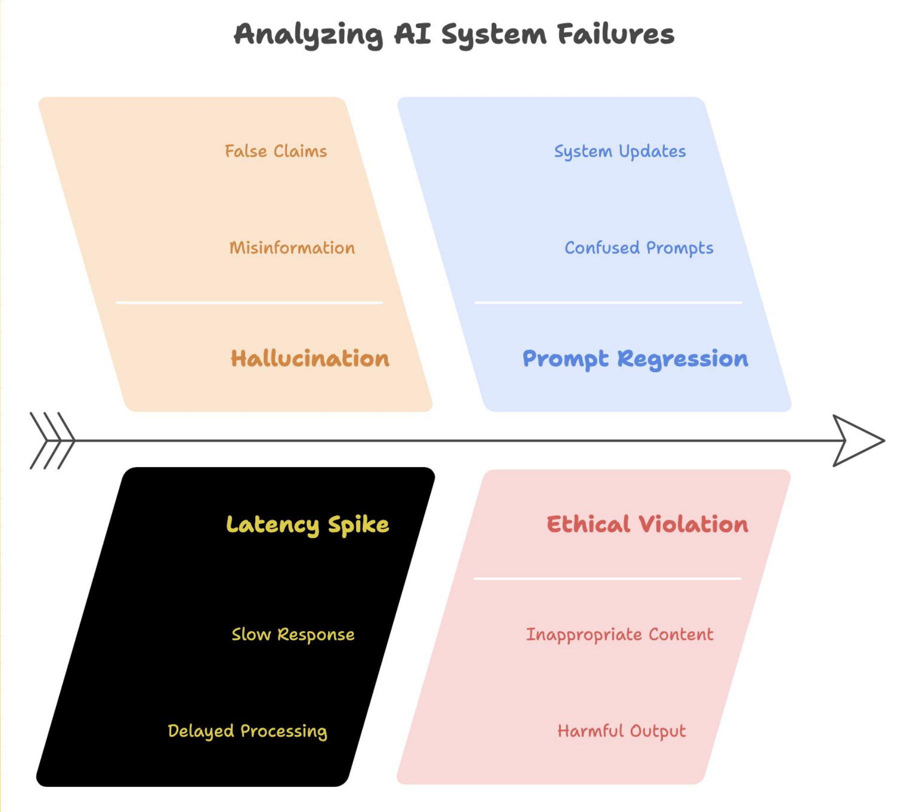
1. LLMs are non-deterministic
2. Failure is inevitable in production
3. LLMs interact with real users
4. Traditional ML monitoring ≠ enough
5. Regulatory and ethical scrutiny is rising

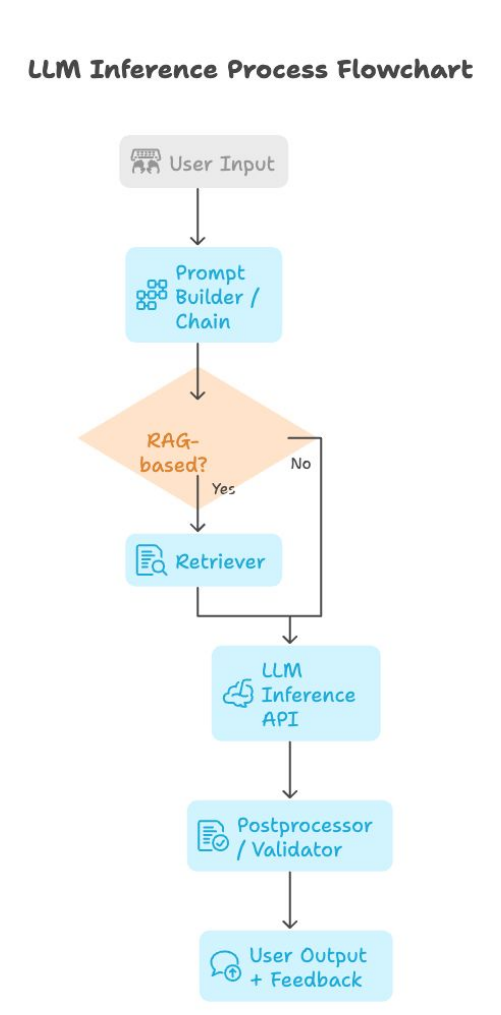


Common Failure Modes in LLM Pipelines

LLMs break in subtle and not-so-subtle ways and knowing **where and how** helps you detect issues early.

1. **Hallucinations**
2. **Prompt Regressions**
3. **Latency Spikes**
4. **Data Drift**
5. **Inconsistent Behavior**
6. **Ethical & Compliance Risks**





Key Components of an LLM Pipeline

Input Interface - Unexpected formats, malicious content

Preprocessing / Orchestration Layer -Prompt bugs, formatting errors, missing context

Retriever (Optional for RAG) - Retrieval latency, irrelevant or outdated context

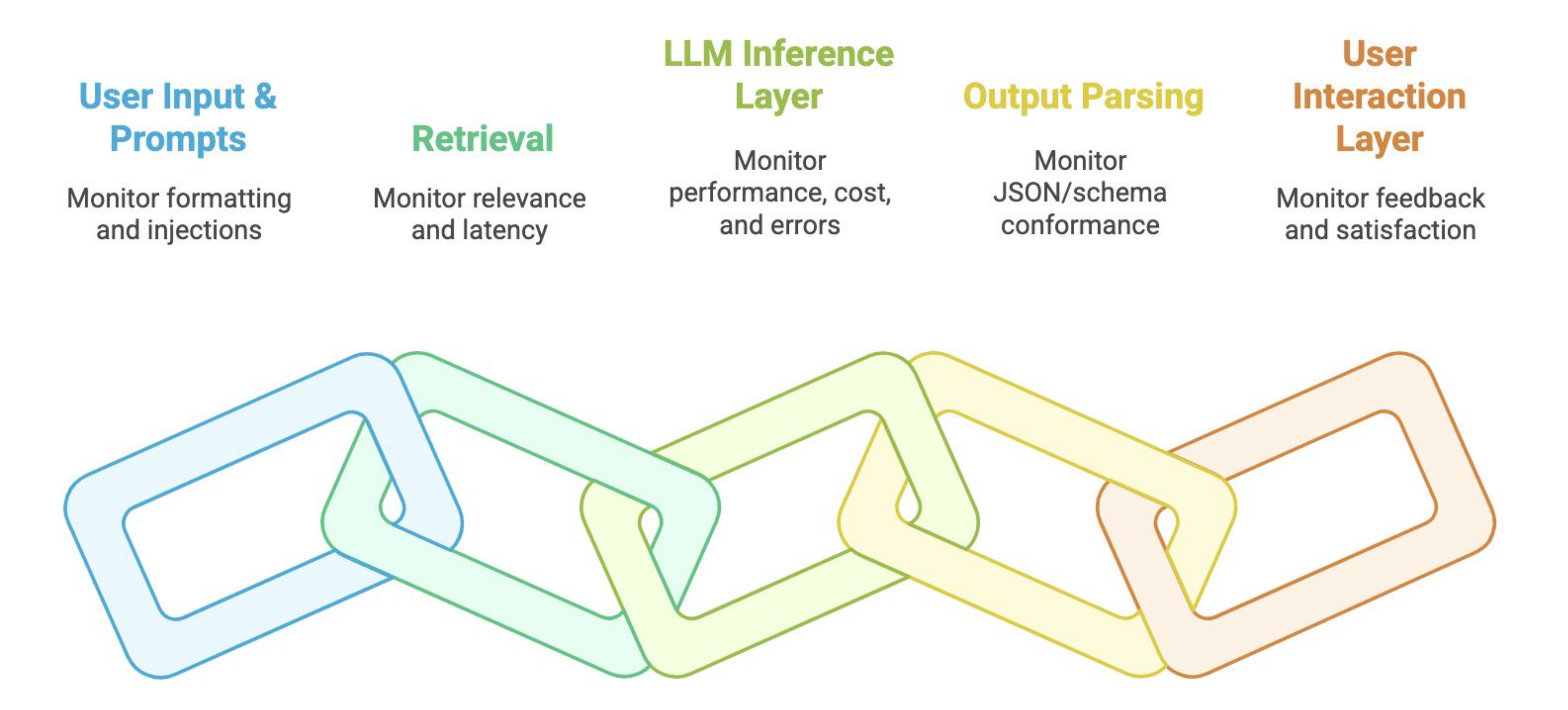
LLM Inference -: Hallucinations, long responses, API instability

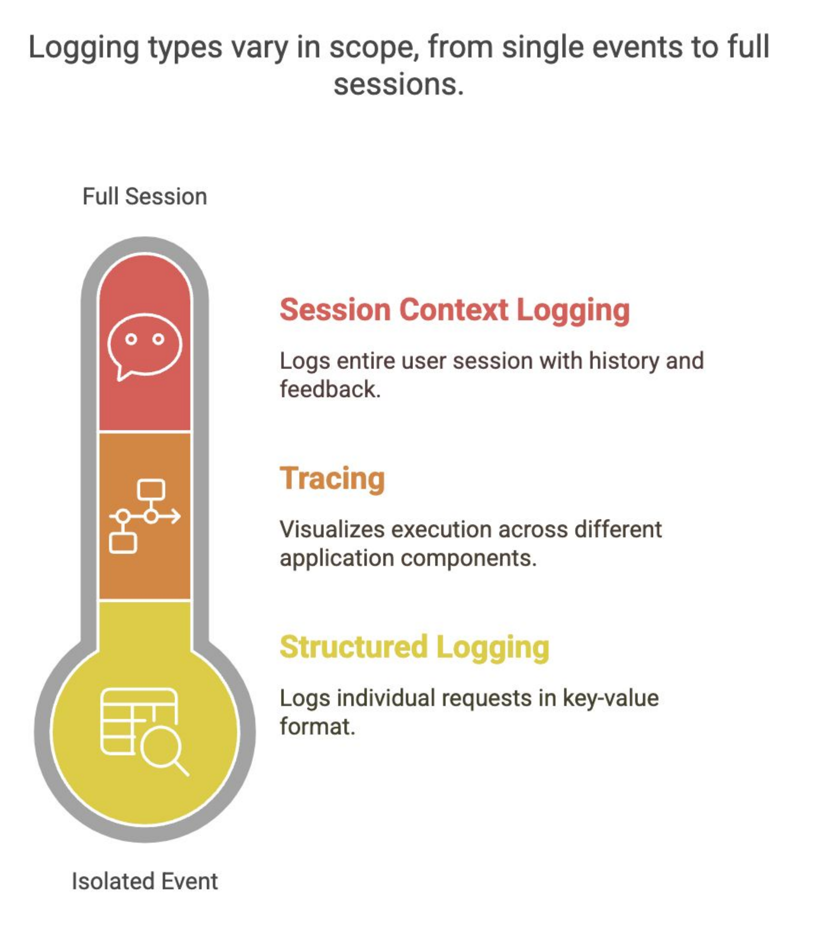
Post Processing - Structure mismatches, broken JSON, empty completions

User-Facing Output + Feedback Capture - Risk: User misunderstanding, missing feedback data

02. Monitoring & Debugging Techniques

What to Monitor – Observability Categories



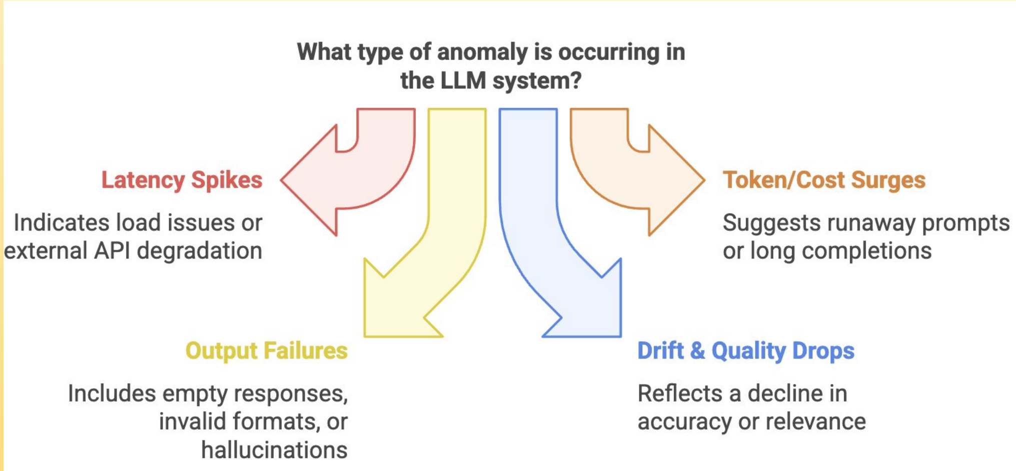


Structured Logging & Tracing for LLMs

Best Practices:

- Standardize log schemas across services
- Include version info (model, prompt, retriever, chain)
- Sanitize logs for PII before storing
- Use UUIDs to correlate inputs and outputs

Anomaly Detection in Production



Some methods:

1. **Threshold-Based Alerts**

e.g., response_time > 2s, token_count > 1024

2. **Statistical Anomaly Detection**

Rolling averages, standard deviations, z-scores

3. **Drift Detection**
   a. Monitor input distribution (e.g., query types)
   b. Monitor embedding similarity distributions (for retriever drift)

4. **Feedback Signal Analysis**

Sudden drop in thumbs-up ratio or user ratings

03. Tooling & Feedback Loops

Metrics That Matter – Performance, Drift, Hallucination, Ethics

Performance Metrics

Latency (avg, p95, p99)

Token usage (input/output split, cost tracking)

Throughput (requests per second)

Timeouts & retries

Completion length distribution

Data & Query Drift

Input drift: Changes in prompt shapes, user query types

Retriever drift: Shift in similarity scores, relevance of retrieved docs

Embedding drift: Embedding vector distribution shifts (esp. across model upgrades)

Hallucination & Output Quality

Factual accuracy (e.g., using eval sets or ground truth)

Consistency across reruns (same input → different answers?)

Response structure correctness (JSON, schema conformance)

Ethical & Safety Metrics

Toxicity levels

Bias detection (gender, race, nationality)

PII leakage detection

Safety classification scores

Different LLM architectures demand different observability strategies

RAG (Retrieval-Augmented Generation)

Monitor retrieval relevance

Use similarity thresholds and embedding distance histograms

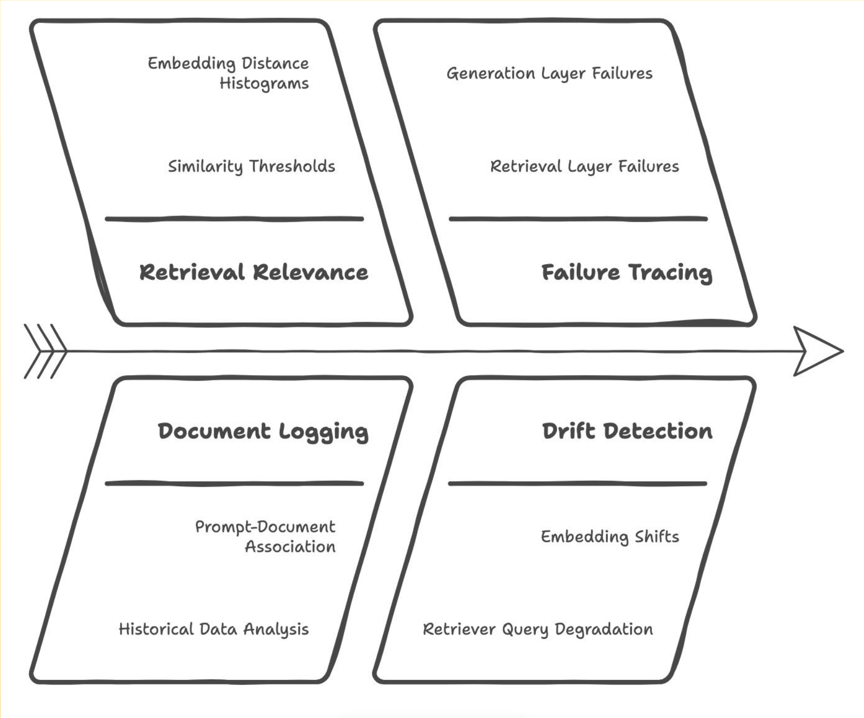
Auto-evaluate retrieved docs against ground truth

Log retrieved documents with each prompt

Trace failures to retrieval OR generation layer

Drift Detection: Embedding shifts, retriever query degradation

LangSmith + Vector Store Logs + Open Telemetry



Different LLM architectures demand different observability strategies

Chatbots

Session-based tracing: Track user flow, turn-by-turn behavior

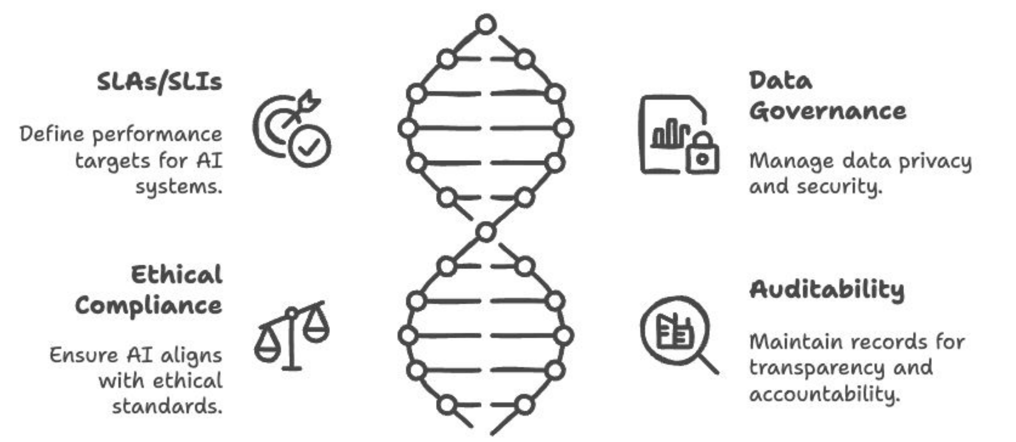
Escalation metrics: How often users retry or escalate to human agents

Toxicity & safety filters at response and prompt level

Feedback-driven improvement loop

LangSmith + Structured Logging + feedback dashboards

Different LLM architectures demand different observability strategies



Enterprise AI Applications

SLAs/SLIs: Define hard targets (e.g., response under 1s, 99.9% uptime)

Data governance logging: Log user input anonymization, PII redaction

Ethical compliance checks: Model outputs scored against company risk frameworks

Auditability: Retain trace logs for inspection

Grafana/Prometheus + MLflow/ZenML + internal logging frameworks

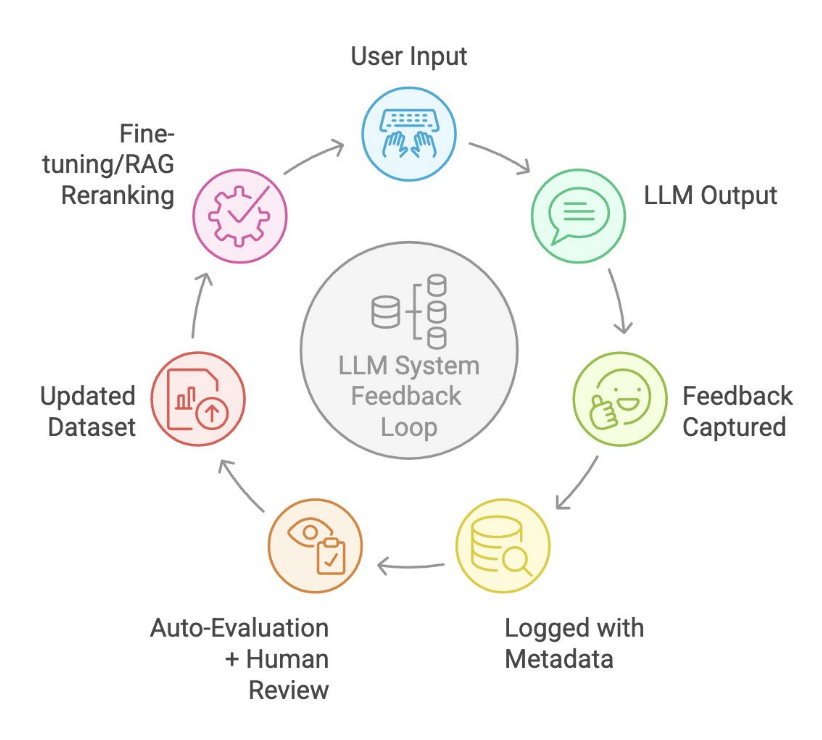
Some “in-general” best practices

- Use **LangSmith** or similar to trace full LLM pipeline execution
- Store logs with metadata: model version, prompt version, retriever version
- Include **user feedback hooks** for all user-facing LLM systems
- Tag and version all experiments — models, prompts, context logic

And finally, Feedback loops

3 types

1. **Explicit Feedback**
    - User thumbs-up/down, star ratings, comments
    - Helps label data for fine-tuning or re-ranking
    - Use in LangSmith or in-house dashboards
2. **Implicit Feedback**
    - User dwell time, retries, reformulations, click-throughs
    - Track where users abandon or escalate the task
3. **System Feedback**
    - Eval scores (e.g., accuracy, structure validity)
    - Logs flagged for anomalies or hallucinations



Thank you! *Time for Q & A?*

You can reach out to me at

abi@abiaryan.com

Socials: @goabiaryan

(LinkedIn, Twitter, Threads, Twitter)