# Observability for LLMs

Understanding what is happening under the hood

Salman Munaf
Lead SRE @ TikTok
LinkedIn

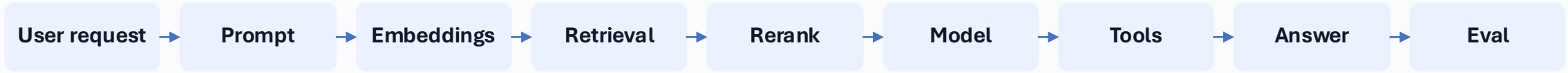
usenix
SRE CON_

# Why this matters now

- LLMs are moving into real products and operational workflows
- Traditional service dashboards do not explain model behavior
- A request can return 200 and still be wrong, unsafe, or useless
- Observe the workflow, not just the endpoint

# How an LLM application works

One user request often becomes many backend steps



# What is LLM observability

### Operational health

Latency
Errors
Availability
Throughput

### Workflow visibility

Model calls
Tokens
Cost
Retrieval
Tools

### Semantic health

Grounding
Quality
Safety
Usefulness

**Classical observability plus AI workflow visibility**

# Why LLM observability is different

**200 OK ≠ good answer**

**Work scales with tokens**

**Streaming changes latency**

**Pipeline, not one call**

**Same endpoint, many workflows**

# What we learned in practice

### What we tracked first

Request rate
Latency
Errors
CPU and memory

### What still looked wrong

Inconsistent quality
Rising cost
Unpredictable workflow behavior

### What was missing

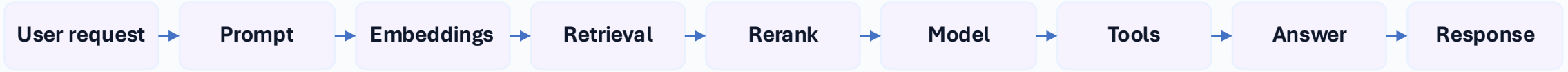
Token usage
Time to first token
Retrieval behavior
Tool success
Finish reason
Quality signals

**Healthy service metrics did not guarantee healthy AI behavior.**

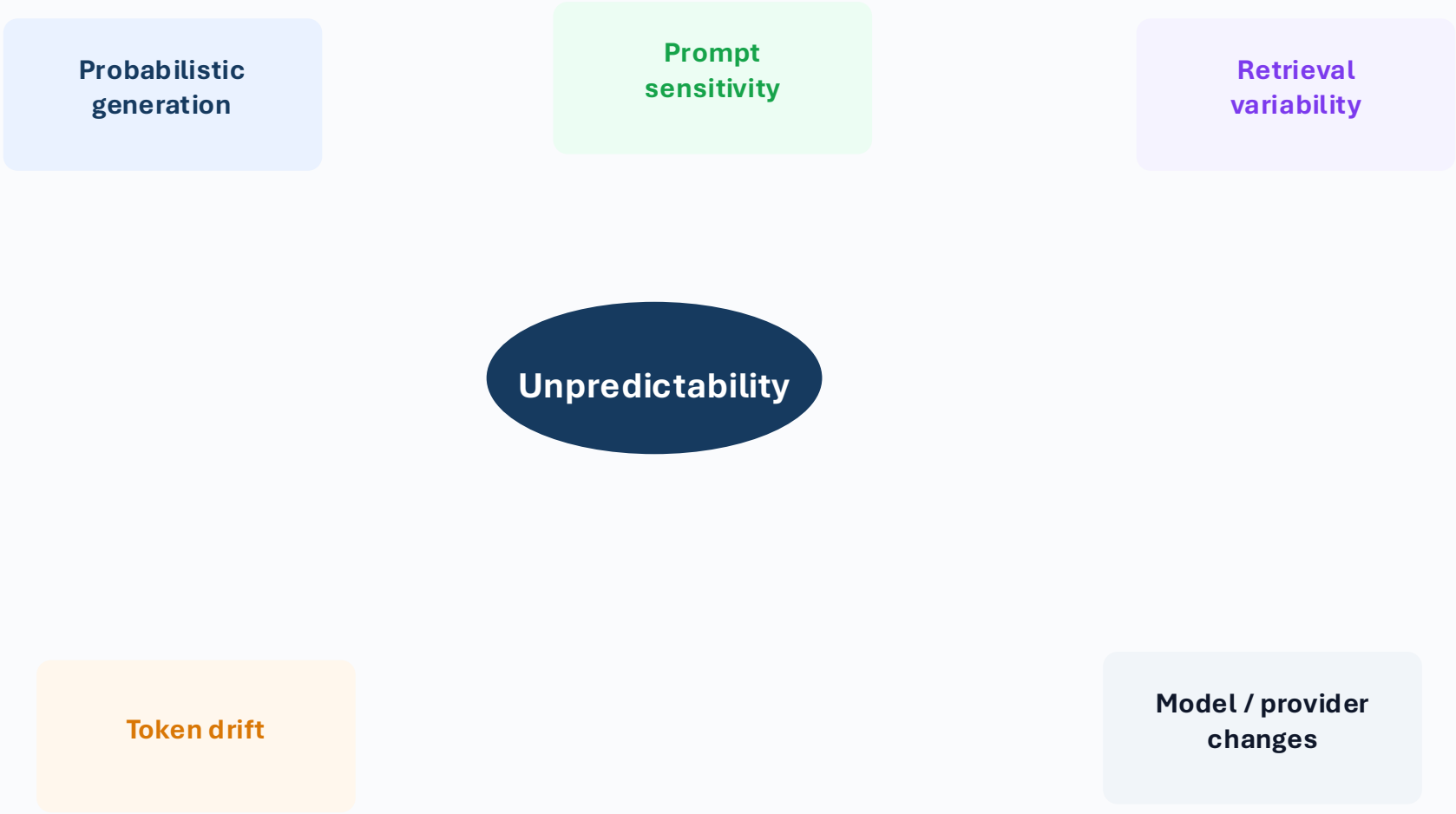
# Classical service vs LLM application

## Classical service


## LLM application



# What makes LLM systems unpredictable



# The four layers you need to observe

A healthy LLM system needs all four layers to look healthy together

User experience

Model and tokens

Retrieval and tools

Infrastructure and serving

# Layer 1: User experience signals

| What this layer tells you | Key signals | Lesson |
| --- | --- | --- |
| What the user feels | End to end latency<br>Time to first token<br>Time per output token<br>Success rate<br>Feedback<br>Retry / abandonment | The backend can look healthy while the UX still feels bad. |

# Layer 2: Model and token signals

| What this layer tells you | Key signals | Lesson |
| --- | --- | --- |
| What the model did and how much work the request required | Input tokens<br>Output tokens<br>Total tokens<br>Cost per request<br>Finish reason<br>Model / provider<br>Prompt version<br>Model calls per request | Requests are not equal. Tokens are a first class workload signal. |

# Layer 3: Retrieval and tool signals

## What this layer tells you

What context and external actions shaped the answer

## Key signals

Retrieval latency
Empty retrieval rate
Retrieved docs count
Retrieval relevance
Tool call count
Tool success rate
Tool latency

## Lesson

The model is only one part of the workflow.

# Layer 4: Infrastructure and serving signals

## What this layer tells you

What was happening in the model server and hardware

## Key signals

GPU utilization
GPU memory
KV cache usage
Queue time
Waiting requests
Running requests
Token throughput
Node health

## Lesson

App signals tell you what happened. Infra signals help explain why.

# Putting the layers together

**Latency rise**

Prompts bigger, retrieval slower, or server queuing?

**Quality drop**

Retrieval weakened, truncation increased, tool failure, or model change?

**Cost rise**

token growth, extra model calls, or prompt bloat?

**Throughput drop**

traffic, context growth, or GPU memory pressure?

# Instrumenting LLM workloads

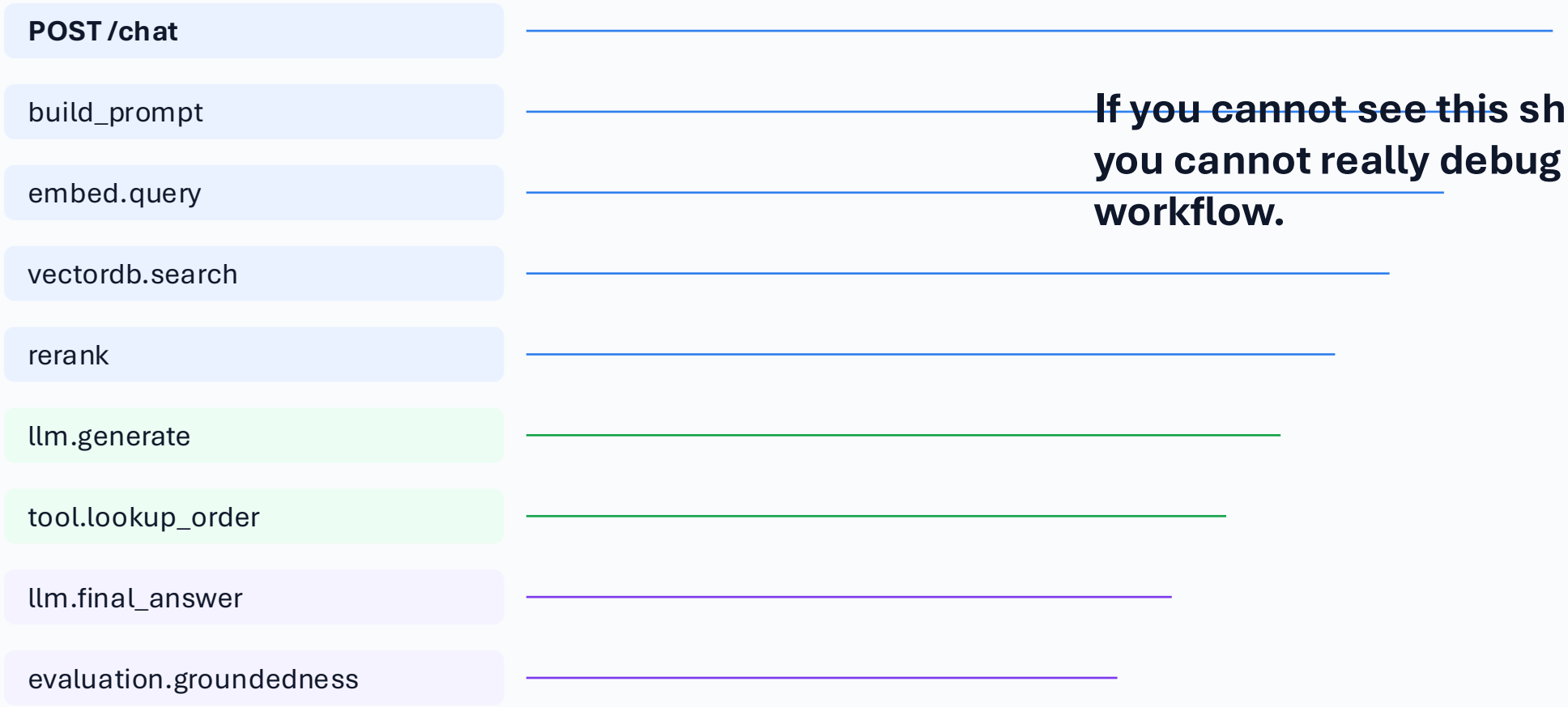
## Trace the full workflow

Do not instrument only the final model call.

## Attach metadata

Model
Provider
Prompt version
Route
Tenant
Retrieved doc IDs

# What a good trace should look like




**If you cannot see this shape, you cannot really debug the workflow.**

# Logging tokens and content safely

## Default keep

Token counts
Model identity
Prompt version
Doc IDs
Finish reason

## Be careful with

Raw prompts
Raw completions
Sensitive user data
Broad always on capture

**Use sampling + redaction for deeper debugging**

# Connecting to infrastructure signals

### App side

Traces
Latency
Tokens
Retrieval
Tools

### Serving side

TTFT
Queue time
Waiting / running requests
KV cache

### GPU side

Memory
Utilization
Thermals
Power

**Correlate by service • pod • namespace • node • model**

# Example incident: quality drop tied to infra pressure

## Symptoms

Error rate normal
Latency slightly worse
Quality drops
Complaints increase

## Correlated findings

GPU memory rises
Cache usage rises
Waiting requests rise
Fewer docs fit
More responses stop due to length

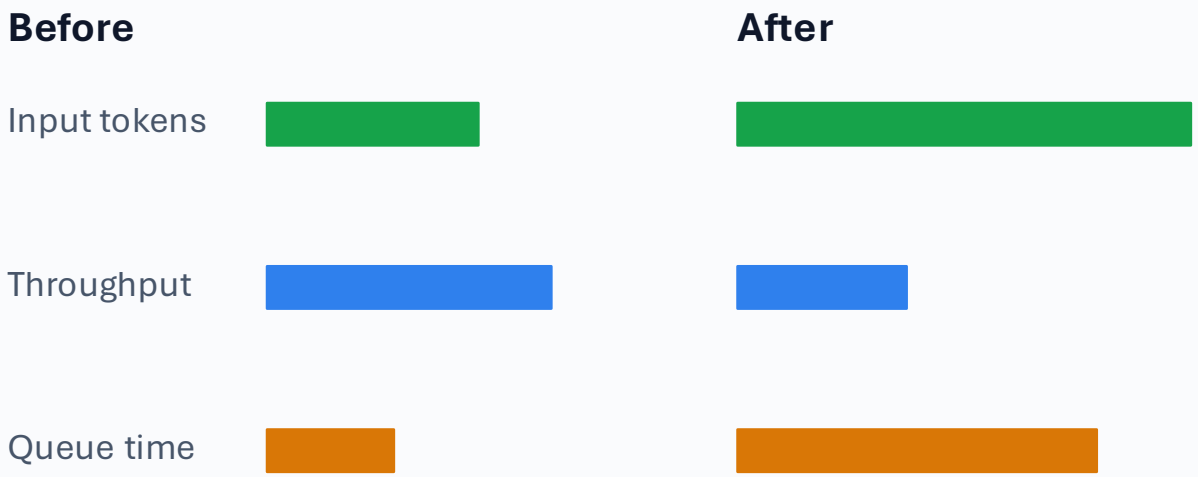
## Lesson

Infra pressure changed the workflow and degraded answer quality.

# Example incident: throughput regression tied to context growth

## Symptoms

Request rate stable
Throughput drops
TTFT rises
Queue time rises




**Lesson**
Requests are not the real unit of work.
Tokens are.

# Practical metrics and alerts

## Page or investigate quickly

p95 TTFT rising
Queue time rising
GPU memory near limit
Waiting requests increasing
Tool success dropping

## Watch for drift and regression

Input tokens per request jumping
Empty retrieval rate increasing
Finish reason length increasing
Quality score dropping

# A starter observability stack you can build tomorrow

| | |
|---|---|
| **Application** | • OpenTelemetry traces<br>• Prompt -> retrieval -> tools -> model -> eval |
| **Serving** | • vLLM metrics<br>• TTFT, TPOT, queue time<br>• Running/waiting requests |
| **Infrastructure** | • GPU telemetry (DGCM exporter)<br>• Memory, utilization |
| **Quality** | • Lightweight evals<br>• User feedback |
| **Storage & dashboards** | • Prometheus + Grafana |

# The mental model to remember

## Traditional observability

Is it up
Is it fast
Is it error free

## LLM observability

Did the workflow behave correctly
Did it answer well
Did it stay efficient
Did the serving layer support that outcome

# Closing

1

Workflows, not just endpoints

2

Tokens, retrieval, and quality are first class signals

3

App traces + model server metrics + GPU telemetry

**If your critical dependency is now a model,
your observability stack needs to observe reasoning pipelines,
not just request paths.**

# THANK YOU!