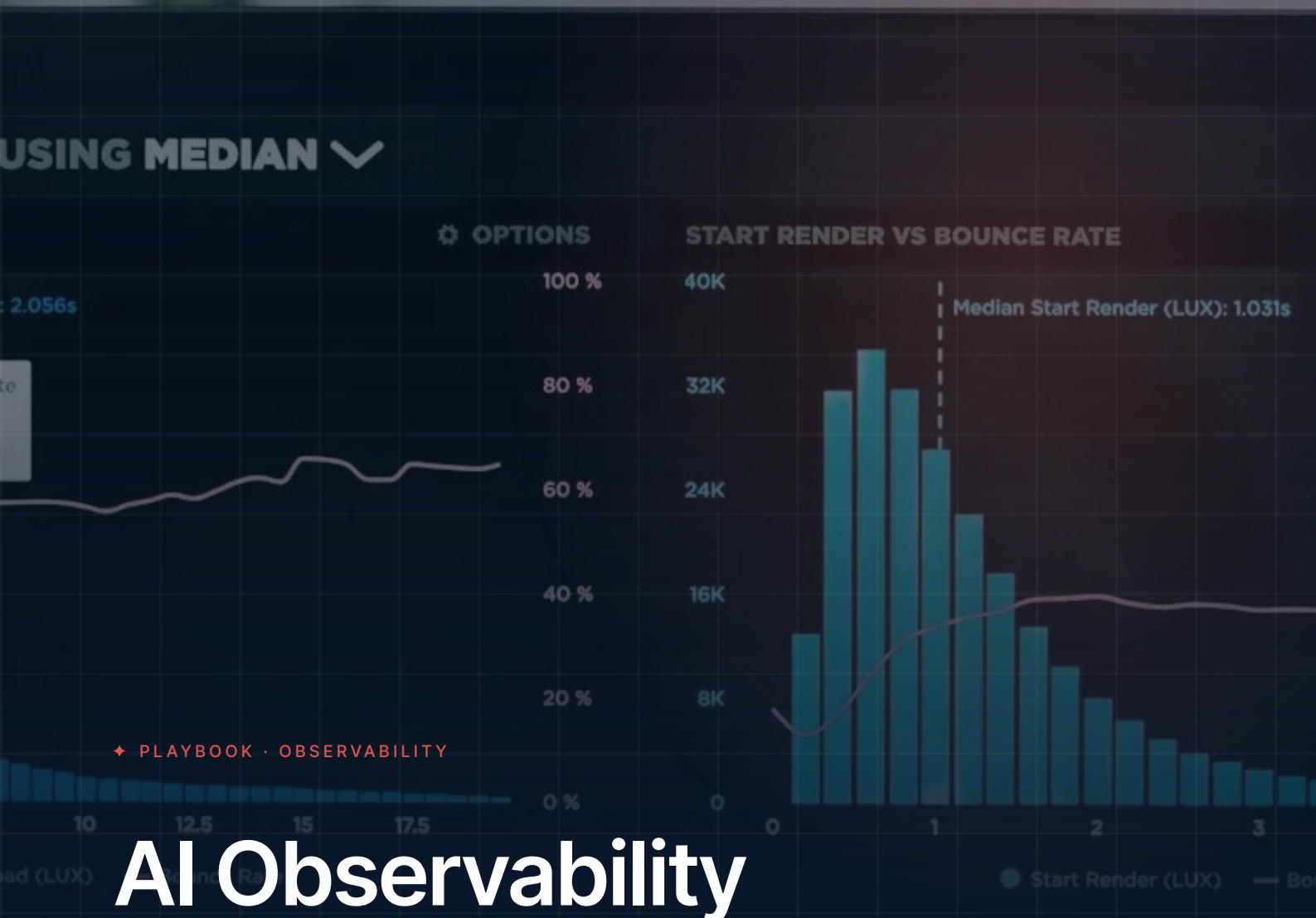


✦ PLAYBOOK · OBSERVABILITY

AI Observability & Tracing

How to trace LLM, retrieval and tool calls, what to log and what to redact, online evaluation, drift detection, and a debugging workflow that starts from a single bad answer.

FORMAT	LEVEL	READ TIME	UPDATED
Playbook	Intermediate	16 min	Mar 2026

OVERVIEW

What this playbook covers

Traditional monitoring tells you a request succeeded in 2 seconds with a 200. For an AI system that is the least interesting part: the answer may still be wrong, ungrounded or unsafe. This playbook shows how to instrument AI requests as traces of model, retrieval and tool spans, protect sensitive data in them, score quality online, and turn a user's "this answer is wrong" into a root cause in minutes.

KEY TAKEAWAYS

- Trace every request as a tree of spans: one per model call, retrieval, rerank, tool call and guardrail.
- Log prompts, retrieved IDs, outputs, tokens and versions — enough to replay the request exactly.
- Redact PII before traces leave the service, and control who can read raw prompts.
- Run online evals on sampled production traffic so quality is measured continuously, not only before release.
- Watch for drift in inputs, retrieval scores and output behavior, not just error rates.
- Follow OpenTelemetry conventions so AI traces sit alongside the rest of your telemetry.

WHO IT'S FOR

- AI and platform engineers
- SRE and on-call teams
- ML and evaluation engineers
- Security and privacy reviewers

HOW TO USE IT

Use sections 01–02 to design your span model and logging fields, 03 for privacy controls, and 04–06 to set up evals, dashboards and the on-call debugging workflow.

CONTENTS

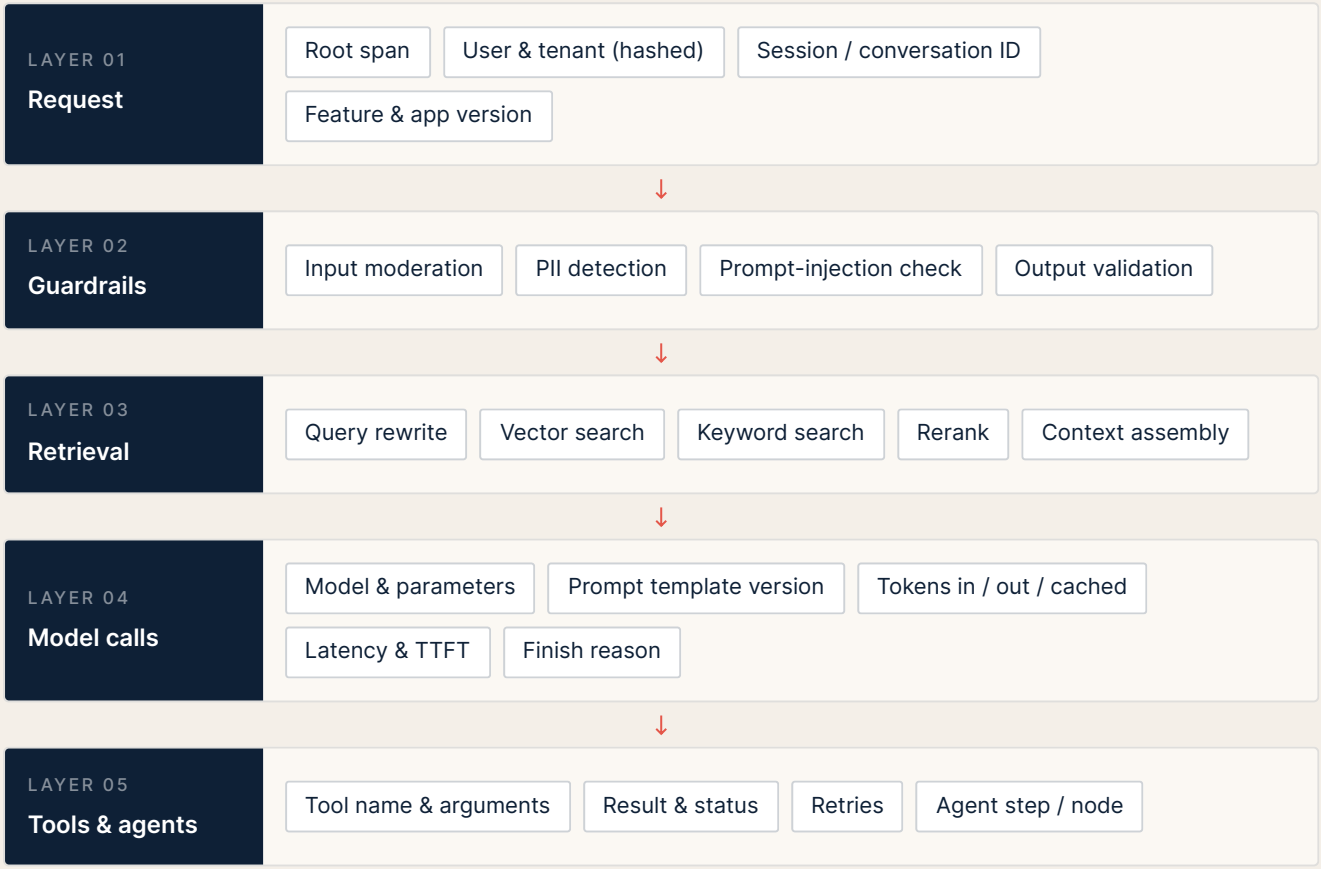
01 The span model for AI requests
02 What to log on every request
03 PII redaction and access to traces
04 Online evals and drift
05 Dashboards and alerts
06 From a bad answer to a root cause

01 / TRACING

The span model for AI requests

A single user request can involve a dozen model, retrieval and tool calls. Model each as a span in one trace so the whole path is visible and timed.

Anatomy of a traced request



Following OpenTelemetry conventions

CONCERN	CONVENTION	NOTES
Model calls	GenAI semantic conventions (`gen_ai.*` attributes)	Operation, provider, request and response model, token usage
Prompt and completion content	Recorded as events or attributes, opt-in	Often large and sensitive; capture behind a flag with redaction
Tool and agent steps	Child spans named per operation and tool	Keep tool name and status as attributes for filtering
Context propagation	W3C trace context and baggage	Carries trace, tenant and session across services and queues

NOTE

Conventions are still evolving

The OpenTelemetry GenAI conventions are newer than the HTTP and database ones and still changing. Pin the version you emit, and map attributes in one place so an upgrade is a single change.

02 / LOGGING

What to log on every request

The test of your logging is whether an engineer can reproduce a bad answer from the trace alone, without asking the user what they did.

INPUTS

Rendered prompt

The final prompt sent to the model, plus the template ID and version it came from.

INPUTS

Retrieved context

Chunk IDs, scores and document versions — not just the text — so you can check what the model saw.

CONFIG

Versions

Model, provider, parameters, prompt, index and application version on every span.

OUTPUTS

Raw response

The unparsed output, finish reason, parse errors and any guardrail action taken.

COST

Usage

Input, cached and output tokens, computed cost and latency per call.

FEEDBACK

Outcome signals

Thumbs, edits, retries, escalations and task completion, joined back to the trace ID.

LOGS THAT DO NOT HELP
“LLM call failed” with no prompt
Final answer only, no retrieval
No prompt or model version
Feedback stored in a separate tool with no trace ID

LOGS THAT DO
Full span tree with inputs and outputs
Retrieved chunk IDs and scores
Every version pinned on the span
Feedback linked to the exact trace

✦ 03 / PRIVACY

PII redaction and access to traces

Traces of AI systems contain exactly what users typed and what documents said. Treat the trace store as a production data store with its own controls.

01 **Redact at the source**

Run PII detection and masking in the service before spans are exported, not later in the pipeline.

02 **Tokenize, don’t delete, identifiers**

Replace emails, phone numbers and IDs with stable hashes or tokens so traces can still be grouped and joined.

03 **Separate content from metadata**

Store prompt and completion text in a restricted store with shorter retention; keep metrics and metadata broadly available.

04 **Control and audit access**

Role-based access to raw content, scoped by tenant; every view of raw content is logged.

05 **Apply retention and deletion**

Retention periods per data class, and user or tenant deletion requests propagate to traces.

SECURITY

Secrets end up in prompts

Users paste API keys, passwords and tokens into chat. Add secret-pattern detection to the redaction step, not just personal data detection.

04 / QUALITY

Online evals and drift

Offline evals tell you a release was good on the questions you thought of. Online evals tell you whether production is still good on the questions users actually ask.

01 Sample production traces → 02 Redact & enrich → 03 Score with judges & checks → 04 Aggregate by segment → 05 Alert & queue for review

SIGNAL	HOW TO MEASURE	WHAT IT CATCHES
Groundedness	LLM judge compares answer with retrieved context	Hallucinations and unsupported claims
Retrieval quality	Top score distribution; empty or low-score retrievals	Index gaps, stale content, query drift
Format validity	Schema and parser checks on structured output	Prompt or model regressions
Refusal rate	Share of “I don’t know” or declined answers	Over-cautious prompts or missing content
User feedback	Thumbs, edits, retries per segment	Dissatisfaction not visible to automated checks
Input drift	Topic clusters, language and length over time	New use cases the system was not built for

05 / OPERATIONS

Dashboards and alerts

Build dashboards around questions on-call engineers and product owners actually ask, and alert only on what someone must act on.

01

Health

Request volume, error rate by provider and tool, timeouts, p50 and p95 latency, time to first token.

02

Quality

Online eval scores, refusal rate and negative feedback, sliced by feature, tenant and model version.

03

Cost

Tokens and spend by feature, model and tenant; cost per successful outcome; cache hit rates.

04

Safety

Guardrail triggers, blocked outputs, PII detections and prompt-injection flags over time.

Alerts worth paging on

ALERT	TYPICAL TRIGGER	FIRST CHECK
Provider errors	Error rate above baseline for several minutes	Provider status; failover route
Latency regression	p95 above budget after a deploy	Prompt size, model change, retrieval stage
Quality drop	Online eval score falls below threshold	Recent prompt, model or index changes
Spend anomaly	Hourly spend well above trailing average	Loops, retries, a single tenant or key

✦ 06 / DEBUGGING

From a bad answer to a root cause

Most AI incidents start as one user reporting one wrong answer. A repeatable workflow keeps the investigation short.

01 **Find the trace**

Look up by trace ID from the feedback event, or by user, session and time.

02 **Check retrieval first**

Was the right source retrieved and ranked high? If not, the fix is in ingestion or retrieval, not the prompt.

03 **Check the rendered prompt**

Did the model see the right instructions and context, within the token budget?

04 **Replay with versions pinned**

Re-run the exact request against the same model, prompt and index version to confirm reproducibility.

05 **Add it to the eval set**

Every confirmed failure becomes a regression case so the fix stays fixed.

Observability readiness

- ☐ Every model, retrieval and tool call emits a span in one trace
- ☐ Prompt, model, index and app versions recorded on spans
- ☐ PII and secret redaction applied before export
- ☐ Online evals running on sampled traffic with alerts
- ☐ User feedback joined to trace IDs
- ☐ Runbook for the bad-answer debugging workflow

GET IN TOUCH

Let's Talk

We instrument AI systems end to end — traces, redaction, online evals and the dashboards on-call teams actually use. Bring us a system you cannot see inside.

- Span model and OpenTelemetry setup
- PII redaction and trace access controls
- Online evals, drift detection and alerting

BOOK AN AI STRATEGY SESSION →

EMAIL
hello@splendenslabs.com

WEB
splendenslabs.com

OFFICE
Sona Incubation Foundation, Salem