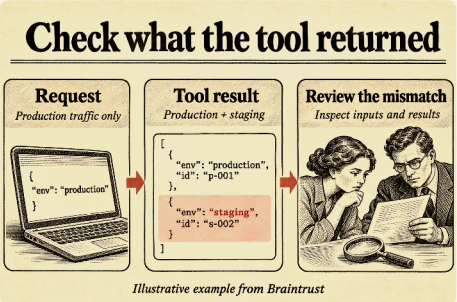
tidytraces

*Daily AI observability news by **Soumendra Kumar Sahoo**.*

• TOP STORY

NEWS · 2026-11-05 · BRAINTRUST › · PERMALINK

Braintrust compares MCP tool requests with the results agents received



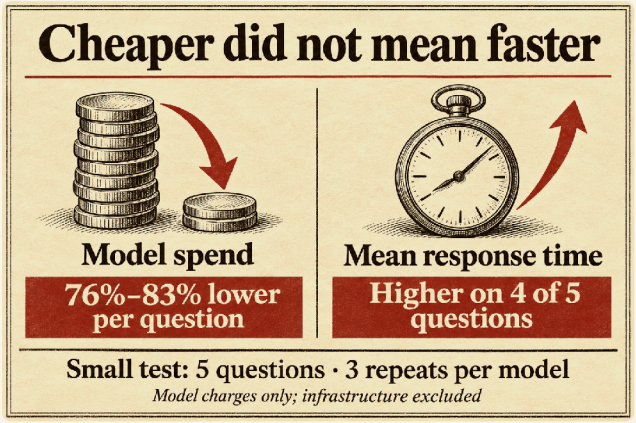
Braintrust is an AI observability and evaluation platform that records agent activity for inspection. Its new server tracing for Model Context Protocol, the interface agents use to call tools, addresses a visibility gap: a server operator may see a successful response without knowing that the tool returned the wrong data. The operator may also lack the agent's full conversation.

First, enable tracing with the Braintrust JavaScript or Python SDK; the announcement covers official MCP implementations and FastMCP. Each call records its inputs, results, timing and errors. Then open the call in Logs and compare the request with the response. The article illustrates this with a production-only query that incorrectly returns staging records.

Operators can optionally add telemetry.intent to the tool schema and a send_feedback tool to collect an agent's stated purpose and reported difficulties. These additions are not prerequisites for tracing. Patterns can surface repeated behavior with supporting calls. After a repair, the article recommends repeating the request, adding a test and reviewing later traces for recurrence.

NEWS · 2026-11-05 · ELASTIC OBSERVABILITY LABS › · PERMALINK

Elastic's agent traces show a cheaper model can still answer more slowly



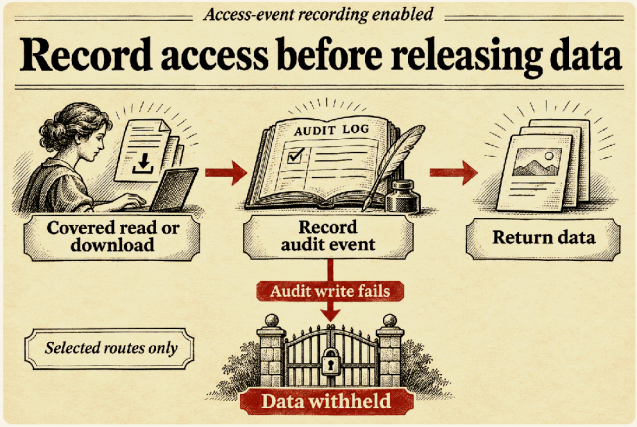
Elastic Observability collects application telemetry for investigating behavior, latency and cost. Its walkthrough instruments a LangChain shopping assistant with OpenTelemetry's generative-AI conventions, shared names for model and agent records. The problem is that total request time alone cannot identify which model call made the customer wait or consumed the budget.

First, the example connects agent, tool and model operations as spans, each recording one operation and its timing. A trace shows three model calls taking about 92% of one request. The application separately collects provider-reported model charges and puts their sum on the request's root record; this excludes infrastructure and retrieval charges. Parent and child durations must not be added together.

Then the author replaces only the product-comparison model and tests five questions three times per model, checking answers as well as cost and latency. Model spend falls by 76% to 83% per question, but mean response time rises on four questions. This is a small sequential test, not a production guarantee. The article recommends broader evaluation and testing whether returning the tool's answer directly can avoid the final model call.

2026-11-05 · LANGFUSE › · PERMALINK

Langfuse records more trace reads, exports and downloads in access audits

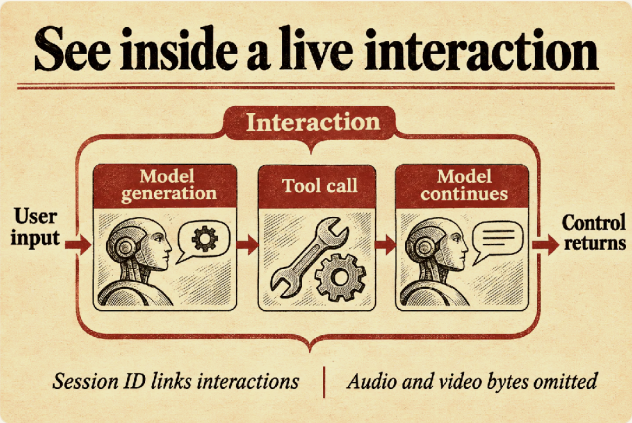


Langfuse is an AI observability platform for inspecting application traces and evaluation scores. Those records can contain sensitive application data, so reviewing who accessed them is a separate operational need from collecting them. The release expands audit access events for selected reads of trace, observation, session and score data, and for exports and downloads. An observation records an operation within a trace.

First, an organization must have Enterprise access-event recording enabled, either through its organization setting or instance enforcement. Covered reads declare the resource and action beside their handler. The implementation records validated request parameters before returning data; a staging-write failure prevents a successful read from releasing it. Selected refused reads also receive denied events.

Exports record before their jobs enter the queue, while covered downloads record before sending bytes. A failed audit write prevents those operations from proceeding. Coverage remains selective: aggregate-only reads and several other routes are excluded. Scheduled exports are outside the export change, batch-export row counts are unavailable at request time, and refused downloads are not recorded as denied. These boundaries come from the linked implementation notes.

OpenInference traces Gemini live interactions and their tool calls



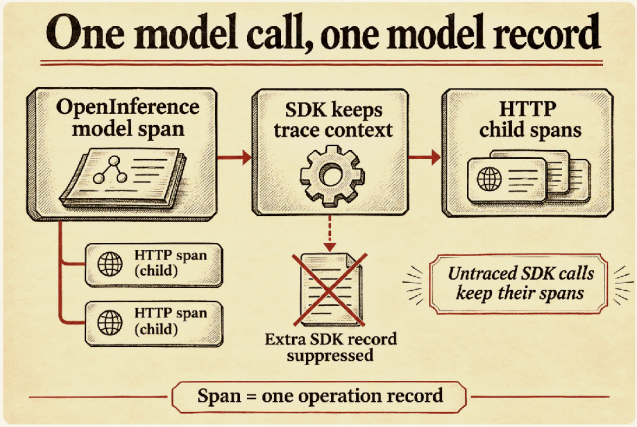
OpenInference provides instrumentation that records AI application activity as traces. Its Google GenAI integration now covers live sessions, which exchange data through a persistent WebSocket connection. Previously, those sessions bypassed the methods the instrumentor wrapped and produced no spans. A span is a record of one operation within a trace.

The new implementation first creates an interaction record from user input until the model returns control. Within it, separate records follow model generations and tool calls. A tool request ends the current generation; another generation begins when the model continues. Session identifiers connect successive interactions. Cancelled tools receive a cancellation event, and nonblocking calls can outlast their starting interaction.

The implementation omits audio and video bytes but can record enabled transcriptions and remote media references. Images remain subject to masking settings, and the session-resumption handle is omitted. The contributor reports a real API smoke test with text and tool turns, but no real microphone or Vertex backend test. Tracing also wraps private SDK functions; their removal can reduce coverage, and streamed tool arguments may be incomplete.

2026-11-05 · ARIZE AI / OPENINFERENCE › · PERMALINK

OpenInference removes duplicate Anthropic SDK spans from traced calls



OpenInference is an instrumentation library that records AI model operations in traces. Its JavaScript Anthropic integration fixes duplicate records introduced when the Anthropic SDK began creating its own span for every API call. A span describes one operation within a trace. Instrumented calls consequently exported both an OpenInference model span and an SDK span for the same request.

The release identifies message creation, message streaming and token counting as affected, across stable and beta interfaces. Streaming had an additional hierarchy problem: its SDK span was not beneath the OpenInference model span. The patch changes the context the integration hands to the SDK when a call is already traced.

First, OpenInference supplies a non-recording span carrying the model span's context. The SDK then retains that context without exporting a second record. The request continues to carry trace context, and HTTP instrumentation records sit beneath the model operation. Calls that OpenInference does not trace keep the SDK's own spans. This is a targeted correction to record duplication and nesting; the release does not report a measured reduction in telemetry cost or changes to model behavior.