Reliable Context

RELIABLE CONTEXT / RESEARCH & PRACTICAL GUIDANCE

Logging AI Agent Actions: Traces an Auditor Can Replay

> Reliable Context is a kernel of truth: a system of record for organizations that fuse private and public information and must defend every value they act on. One canonical record per real thing, a cited source and an observation date behind every value, unknowns declared rather than left blank, and any past state of the record reproducible.

Published by Reliable Context · 2026-10-29 · ai agent audit log · llm agent observability · agent trace logging · ai agent logging best practices · opentelemetry generative ai

Original article: https://reliablecontext.com/articles/logging-ai-agent-actions-replayable-traces-tool-calls



In this article

1. The question an agent log has to answer
2. Why ordinary logs fail for agents
3. An event model for agent runs
4. Referencing facts by identifier and version
5. OpenTelemetry GenAI conventions
6. Determinism and the limits of replay
7. Privacy and retention
8. Tamper evidence
9. A minimum viable schema
10. Checklist
11. Summary
12. Sources

The question an agent log has to answer

An AI agent is a program in which a language model chooses actions, such as calling a tool, querying a data source or handing work to another agent, and observes the results before choosing the next action. The pattern was named in the ReAct paper, which described language models generating "both reasoning traces and task-specific actions in an interleaved manner" (Yao et al., ReAct). When something goes wrong, the investigator's question is not "what did the server do?" but "what did the agent see, what did it decide, what did it do, and on which facts?"

Ordinary application logs answer the first question well and the second badly. This article defines a minimum event model for agent logs that lets an investigator reconstruct a run, explains where the OpenTelemetry generative AI conventions help and where they stop, and sets out the limits of replay. It is written for platform engineers and the compliance teams who will rely on their logs.

Why ordinary logs fail for agents

Three properties of agents break assumptions built into conventional logging.

1. **The decision is made from context, not code.** In a conventional service, the code path determines behaviour, and the code is versioned. In an agent, behaviour depends on the prompt, the retrieved documents, the tool results and the model version at that moment. A log that records "tool X called with arguments Y" without recording what the model had in front of it cannot explain why.
2. **Context is assembled from mutable sources.** A retrieval step returns the current version of a document. If the document is later corrected, a log that stored only its URL now points to text the agent never saw. The site's article on preserving evidence when a cited source changes treats this problem in detail for citations; agent logs inherit it.

3. **Runs branch and hand off.** Agents call sub-agents, retry, and transfer work. Without a shared identifier linking these steps, a run becomes a set of unrelated log lines. Handoffs are a known failure point; the site's agent handoff fault-injection test shows how facts get lost between agents.

The OWASP Logging Cheat Sheet's general rule still applies: application logs "must record 'when, where, who and what' for each event", and an "interaction identifier" links "all (relevant) events for a single user interaction" (OWASP Logging Cheat Sheet). For agents, "what" has to include the inputs to a decision, not only the action taken.

An event model for agent runs

The model below has seven event types. Each event carries a run identifier, a step number, a timestamp in UTC, and the identifier of the component that produced it.

Event	Records	Why an investigator needs it
`run.start`	Who or what initiated the run, the request, the agent name and version, the policy in force	Establishes authority and the task
`context.item`	Each item placed in the model's context: system instructions, retrieved passage, memory record, prior tool result. Recorded as identifier, version and content hash	Shows what the model could see
`model.call`	Model provider and name as requested and as returned, parameters, response identifier, token counts, finish reason	Ties the decision to a model version
`tool.call`	Tool name and version, call identifier, arguments (or a reference), authorisation scope used	Shows what the agent did and under which permission
`tool.result`	Status, error type, result (or a reference to it), duration	Shows what the agent observed
`decision`	The agent's chosen next step or final answer, with the context item identifiers it cites	Links output to evidence
`human.action`	Approval, rejection, edit or override, with the person, role and reason	Shows where human oversight intervened

Two design choices make this model work.

Context items are first-class events. Most observability setups record the prompt as a single blob. Recording each item separately, with its own identifier, lets an investigator ask which items were present at step 4 and whether any of them was later corrected.

Decisions cite context items. A `decision` event that lists the identifiers of the items it relied on is what turns a trace into evidence. Without it, an investigator can see that a document was in context but not whether the answer used it.

Referencing facts by identifier and version

The cheapest log is one that stores everything. It is also the most dangerous: it copies personal data, credentials and confidential documents into a system usually designed for wide engineering access. OWASP lists data that should "usually not be recorded directly in the logs", including "access tokens", "sensitive personal data" and "data of a higher security classification than the logging system is allowed to store" (OWASP).

The alternative is to log a reference that resolves to the exact content, held in a store with its own access controls and retention. A reference is useful only if it is:

- **Specific:** an identifier for the record, not the system it lives in.
- **Versioned:** the version or effective time the agent saw, so later edits do not change what the log points to.
- **Verifiable:** a content hash, so a reviewer can confirm that the retrieved content is byte-identical to what was used.

The OpenTelemetry GenAI conventions now support this pattern explicitly. When content is uploaded to external storage, instrumentation "SHOULD use `gen_ai.system_instructions_ref`, `gen_ai.input.messages_ref`, `gen_ai.output.messages_ref`, or `gen_ai.tool.definitions_ref`", and the convention "does not prescribe the URI scheme, object layout, serialization format, retention policy, or access mechanism" (OpenTelemetry GenAI spans). It also permits "content-derived identifiers to deduplicate repeated content".

The same principle applies to facts the agent retrieves from a system of record. If an agent reads "Supplier status: approved" from a canonical record, the log should hold the record identifier, the field, the version, and the value's own source, not just the text "approved". A related test on this site, the cached-answer reuse validity test, shows why: an answer is only valid while the facts it rested on are unchanged, and the log is how that is checked.

OpenTelemetry GenAI conventions

OpenTelemetry is the open standard for traces, metrics and logs. Its generative AI semantic conventions define names for spans and attributes so that different frameworks emit comparable telemetry. Two facts matter before relying on them:

- The documentation page at opentelemetry.io now shows a notice that the GenAI conventions "have moved to the OpenTelemetry GenAI semantic conventions repository" (opentelemetry.io). The current text is in the semantic-conventions-genai repository.
- Their status is "Development" (GenAI README). Attribute names may change.

The conventions read on 28 October 2026 define, among others, these agent operations and attributes (agent spans; model spans):

Event in the model above	OpenTelemetry construct	Notes
`run.start`	`invoke_agent` or `invoke_workflow` span; `gen_ai.agent.name`, `gen_ai.agent.id`, `gen_ai.conversation.id`	Conversation id SHOULD NOT be invented: "a new UUID, a trace identifier, or a hash of request content SHOULD NOT be used as a fallback value"
`context.item`	`gen_ai.input.messages`, `gen_ai.system_instructions`, `gen_ai.retrieval.documents`, `gen_ai.data_source.id`, or the `_ref` variants	Content attributes are Opt-In
`model.call`	Inference span; `gen_ai.request.model`, `gen_ai.response.model`, `gen_ai.response.id`, token usage	Records the model actually returned, not just requested
`tool.call` / `tool.result`	`execute_tool {gen_ai.tool.name}` span; `gen_ai.tool.call.id`, `gen_ai.tool.type`, `gen_ai.tool.call.arguments`, `gen_ai.tool.call.result`, `error.type`	Arguments and result are Opt-In and flagged "may contain sensitive information"
Handoff	"Agent as a tool" refinement for tool calls that transfer work to another agent	Must not be used for transfers that do not happen through a tool call
`decision`	Not defined	Application-specific
`human.action`	Not defined	Application-specific

Content capture is off by default. The configuration `OTEL_INSTRUMENTATION_GENAI_CAPTURE_CONTENT` supports `NO_CONTENT` (the default), `SPAN_ONLY`, `EVENT_ONLY` and `SPAN_AND_EVENT` (model spans). An organisation that turns on full content capture for debugging has also turned its trace store into a store of prompts, retrieved documents and tool outputs, with whatever personal data they contain.

The gaps are the two events that matter most for accountability: the decision with its cited evidence, and the human action. Both have to be added by the application, as custom spans or log records linked by trace and span identifiers.

Determinism and the limits of replay

"Replay" is used loosely. It helps to separate three things:

Term	Meaning	Feasible for LLM agents?
Reconstruction	Showing exactly what the agent saw, called and returned at each step, from the log	Yes, if context items and results are retained or resolvable by reference
Re-execution	Running the same steps again with the same inputs	Partly; tools and data sources may have changed, and model output may differ between calls
Reproduction	Obtaining the same outputs on re-execution	Not guaranteed; depends on the provider, model version and parameters

Auditors generally need reconstruction. The aim of the event model is that every model call and tool call in a past run can be laid out with its inputs and outputs as they were, without calling anything again. Re-execution is useful for debugging and for testing whether a fix changes behaviour, but it answers a different question: what would the agent do now?

Two practical consequences follow. First, record the model identifier returned by the provider (`gen_ai.response.model` in OpenTelemetry), because the conventions distinguish the model requested (`gen_ai.request.model`) from the model that generated the response, and the two need not match. Second, record tool results, or references to retained results, because the tool's answer tomorrow may differ from its answer at the time.

Privacy and retention

Agent logs sit across two sets of rules that pull in opposite directions: keep enough to explain, and keep no more than necessary. Three design rules reduce the conflict.

1. **Separate the trace from the content.** Keep the trace (identifiers, timings, tool names, status) in the observability system, and keep content in a controlled store referenced by `_ref` attributes or equivalent.
2. **Set retention by event type.** A `tool.call` event with an authorisation scope may need to be kept as long as access records are kept; full prompt content may not. Retention should be a property of the record, as in a retention schedule, not of the log index.
3. **Log access to the logs.** OWASP: "All access to the logs must be recorded and monitored" (OWASP).

Where an agent is part of a high-risk AI system under the EU AI Act, the system must "technically allow for the automatic recording of events (logs) over the lifetime of the system", and logging must enable recording of events relevant to identifying risk situations, post-market monitoring and deployer monitoring (AI Act, Article 12). Providers and deployers keep those logs, to the extent under their control, for a period appropriate to the intended purpose "of at least six months" unless other law provides otherwise (Article 19; Article 26(6)). Following the Digital Omnibus on AI, these high-risk requirements apply to Annex III systems from 2 December 2027 (Article 113). Article 12 does not list agent-specific fields; the event model above is one way to meet its purpose for an agent. This is a description of the text, not legal advice.

Tamper evidence

A log that can be edited without trace is weak evidence. OWASP recommends to "build in tamper detection so you know if a record has been modified or deleted" and to "store or copy log data to read-only media as soon as possible" (OWASP). NIST's SP 800-92, published in September 2006, remains the general reference for building log management infrastructure and processes.

For agent runs, a simple pattern is to hash each event together with the hash of the previous event in the run, and to write the run's final hash to an append-only log separate from the trace store. A reviewer can then detect a deleted or altered step by recomputing the chain.

A minimum viable schema

The following example shows three events from one run of a fictional procurement agent at Halden Therapeutics AG (a fictional company). The agent was asked whether a supplier may receive a purchase order.


```
{"run_id":"run_7Q2K","step":3,"type":"context.item","ts":"2026-10-12T09:14:03.220Z",
 "item_id":"supplier:SUP-00412#field=status","version":"v17","valid_from":"2026-09-30",
 "source":"record://vendor-master/SUP-00412","sha256":"9f2c...e1","content_ref":"vault://ctx/run_7Q2K/3"}
{"run_id":"run_7Q2K","step":4,"type":"tool.call","ts":"2026-10-12T09:14:03.910Z",
 "tool":"sanctions_screen","tool_version":"2.3.1","call_id":"call_88a1",
 "args_ref":"vault://args/call_88a1","auth_scope":"screening:read","agent":"procurement-agent@1.4.0"}
{"run_id":"run_7Q2K","step":6,"type":"decision","ts":"2026-10-12T09:14:05.004Z",
 "model":"provider-model-2026-07-15","response_id":"resp_51f0",
 "output_ref":"vault://out/run_7Q2K/6","cites":["supplier:SUP-00412#field=status@v17","call_88a1"],
 "prev_hash":"4b7d...0c","hash":"a03e...77"}
```


Six weeks later, the supplier's status is corrected from "approved" to "suspended", effective 25 September. An investigator can establish from this log that the agent saw version 17 of the status field, valid from 30 September, that the correction post-dates the run, and that the decision cited both the status and the screening call. Without step 3, all that would be known is that the agent approved a suspended supplier.

Checklist

1. Every run has an identifier propagated through sub-agents, tools and handoffs.
2. Each context item is logged separately, with identifier, version and content hash.
3. The model identifier returned by the provider is recorded, not only the alias requested.
4. Tool calls record tool version, call identifier and authorisation scope; arguments and results are stored or referenced.
5. Final outputs cite the context items they rely on.
6. Human approvals, edits and overrides are events with person, role and reason.
7. Content is referenced from a controlled store; trace and content have separate retention.
8. Events are hash-chained, and run hashes are written to append-only storage.
9. Access to logs is itself logged.

Summary

An agent log is useful as evidence when it lets someone reconstruct what the agent saw and did, step by step, and link each output to the facts it used. OpenTelemetry's GenAI conventions cover model calls, tool calls and content references, but leave decisions and human actions to the application. Facts referenced by identifier, version and hash, rather than copied, keep logs both explainable and proportionate. Reliable Context provides identifiers and versions of that kind for the organisational facts agents read.

Sources

1. Yao et al., *ReAct: Synergizing Reasoning and Acting in Language Models*, arXiv:2210.03629. https://arxiv.org/abs/2210.03629
2. OWASP, *Logging Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html
3. OpenTelemetry, *Moved: Generative AI semantic conventions* (notice page). https://opentelemetry.io/docs/specs/semconv/gen-ai/
4. OpenTelemetry, *Semantic conventions for generative AI systems* (README, status Development). https://github.com/open-telemetry/semantic-conventions-genai/blob/main/docs/gen-ai/README.md
5. OpenTelemetry, *Semantic Conventions for GenAI model spans* (including execute tool span, content capture and upload). https://github.com/open-telemetry/semantic-conventions-genai/blob/main/docs/gen-ai/gen-ai-spans.md
6. OpenTelemetry, *Semantic Conventions for GenAI agent and framework spans*. https://github.com/open-telemetry/semantic-conventions-genai/blob/main/docs/gen-ai/gen-ai-agent-spans.md
7. NIST, *SP 800-92, Guide to Computer Security Log Management*, September 2006. https://csrc.nist.gov/pubs/sp/800/92/final
8. European Commission, AI Act Service Desk, consolidated text as at 27 July 2026: Articles 12, 19, 26 and 113. https://ai-act-service-desk.ec.europa.eu/en/ai-act/article-12
9. Regulation (EU) 2024/1689 (Artificial Intelligence Act), EUR-Lex. https://eur-lex.europa.eu/eli/reg/2024/1689/oj

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. https://arxiv.org/abs/2210.03629
2. https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html
3. https://github.com/open-telemetry/semantic-conventions-genai/blob/main/docs/gen-ai/gen-ai-spans.md
4. https://opentelemetry.io/docs/specs/semconv/gen-ai
5. https://github.com/open-telemetry/semantic-conventions-genai
6. https://github.com/open-telemetry/semantic-conventions-genai/blob/main/docs/gen-ai/README.md
7. https://github.com/open-telemetry/semantic-conventions-genai/blob/main/docs/gen-ai/gen-ai-agent-spans.md
8. https://ai-act-service-desk.ec.europa.eu/en/ai-act/article-12
9. https://ai-act-service-desk.ec.europa.eu/en/ai-act/article-19
10. https://ai-act-service-desk.ec.europa.eu/en/ai-act/article-26
11. https://ai-act-service-desk.ec.europa.eu/en/ai-act/article-113
12. https://csrc.nist.gov/pubs/sp/800/92/final
13. https://eur-lex.europa.eu/eli/reg/2024/1689/oj

THE PUBLISHER

About Reliable Context

Reliable Context is a system of record for organizations that fuse private and public information and must defend every value they act on.

The guarantees

One canonical record per real thing: duplicates resolve by rule, ambiguity is shown rather than hidden, and related organizations relate without ever being merged. A verbatim, cited source and an observation date sit behind every value. Unknowns are declared as not found, blocked or not applicable with a reason, never left as an empty cell. Any past state of the record is reproducible, together with what changed since.

Where it runs

It runs inside the customer's perimeter, one database per organization. Permitted sources, model routing, reader clearance and retention are set as the customer's own policies, and every value is exportable with its chain of custody for an auditor or regulator.

What it is not

It is not a vector database or a RAG framework, not a data catalog, MDM or data-quality suite, not a CRM or data-enrichment vendor, and not a chatbot or AI assistant. Where a reader needs one of those, the honest answer is to say so and point them there.

Visit

See Reliable Context.

Reliable Context is in production use in two sectors; more than six thousand of its identity decisions were made by rule and none has been reversed. Articles published here explain general practice, cite their primary and academic sources, and name the jurisdiction of any regulatory rule they describe. They are not legal, regulatory or compliance advice for a specific organization, and requirements change.

Website: https://reliablecontext.com

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.