

The Observability Gap in AI Agents

What to fix before agents scale

Madhulika Srikumar
Eric Mibuari
Claire Leibowicz
Partnership on AI

Borhane Blili-Hamelin
Independent

Kevin Klyman
Harvard University

Dan Leininger
Consumer Reports

Kyle Hall
Slalom Consulting

Xing Hang Lu
Mila Quebec AI Institute &
McGill University

Sean McGregor
Berkman Klein Center

Vinh Nguyen
Council on Foreign Relations

Amin Oueslati
The Future Society

Abhi Sanka
Microsoft

Jason Stanley
ServiceNow

Benedikt Stroebl
Center for Information
Technology Policy,
Princeton University

Sarah Tan
Salesforce

Selva Thandavarayan
JPMorganChase



```
conv.ai  
_tracer("ai-  
_current_span("i  
oute("gen_ai.syste  
("gen_ai.request.i  
pan.set_attribute  
request.max_token  
gen_ai.request+  
.chat.com  
",message  
_tokens=204  
perature=0.2  
_ai.usage.inp  
usage.prompt  
_tribute("gen_ai.  
usage.completion  
("gen_ai.respons  
choices[0].finish_
```

Contents

1. Introduction	3
2. Methodology	6
3. Monitoring Goals	7
What enterprises need to monitor when deploying agents	
Hypothetical use cases	
Six safety monitoring goals	
4. Processing Techniques	11
Emerging techniques for turning raw telemetry into monitoring signals	
Four telemetry processing techniques	
5. Telemetry Gaps	15
Gaps in telemetry generation and their impact on monitoring goals	
6. Recommendations	26
OpenTelemetry contributors	
Agent framework developers	
Model providers	
Enterprise deployers	
Standards bodies and regulators	
7. Conclusion and limitations	29
Appendix A: Telemetry emissions across four agent frameworks	30
Appendix B: Illustrative use cases	39
Bibliography	41

1. Introduction

Agents introduce failures that generative AI does not.¹ They inherit the unpredictability and reliability issues of their underlying Large Language Models (LLMs), but where a generative AI system produces a single output that a human acts on, an agent plans, selects tools, spawns sub-agents, executes actions, and revises its plan mid-run. A failure at any of these stages can shift the agent's course in ways that are hard to predict, with errors compounding as the process unfolds.² Even when the underlying cause lies in system design or training, the failure manifests and escalates during operation. So pre-deployment testing, while necessary, is not enough on its own to detect agent failures. No finite test suite can cover the full space of inputs an agent will encounter in deployment, so continuous monitoring of the conditions under which it actually operates becomes necessary to catch failures that testing can not anticipate. **Monitoring agents during operation is therefore a crucial safeguard as these systems are deployed in critical functions.**³

Real-time monitoring depends on observability: capturing what the system did, processing it into signals, and surfacing those to humans or automated oversight systems.⁴ Without extensive observability, an enterprise cannot reliably detect failures and intervene before harm occurs. As agents are entrusted with increasingly consequential tasks, failures that go undetected can expose sensitive data, trigger unauthorized actions, and create significant financial, legal, and operational risks. This applies whether agents are running internally or in customer-facing settings in industries with high stakes, like banking or healthcare. Agents fail in several ways. Close to half of the agent failures documented to date involve no adversary, arising instead from an agent performing its assigned task unreliably or from a misaligned agent pursuing goals its deployers did not intend.⁵ In a recent incident, misaligned agents in an OpenAI evaluation broke out of their test environment and breached Hugging Face's systems. The agents' behavior was not caught by monitoring during the evaluation and only surfaced after it caused a service outage.⁶ Agents can also be manipulated by bad actors, through hidden instructions or compromised inputs that cause unauthorized actions⁷ or exposure of sensitive data.⁸

Retrospective review also depends on a complete record of what the agent did as it ran. In regulated settings, an enterprise may need to reconstruct months later why an agent

Without extensive observability, an enterprise cannot reliably detect failures and intervene before harm occurs.

1 By agents, we mean LLM-based software systems that take actions in digital environments, not embodied or robotic systems. References to "agents" and their actions in this paper use anthropomorphized language to keep descriptions concise. This is not meant to blur the distinction between software systems and humans. For a breakdown of how LLM-based agents vary in their levels of influence on digital environments, see Figure 1 in Srikumar, Pratt et al., 2025.

2 Bengio et al., 2024.

3 Shavit et al., 2023; Chan et al., 2024; Srikumar, Pratt et al., 2025; Bengio et al., 2024; Boston Consulting Group, 2025; PwC, 2026.

4 Effective analysis of agent incidents depends on what information gets captured during operation. See Ezell, Roberts-Gaal & Chan, 2025.

5 Drawn from a subset of agent-related incidents in the AI Incident Database, compiled and categorized for this analysis; the categorization was partly automated. See AI Incident Database, 2026.

6 See METR, 2026.

7 See Chaikin & Sahib, 2025.

8 See French, 2025

All cited sources can be found in the Bibliography on page 41.

reached a particular decision, for instance to show an auditor that a declined loan or a closed investigation rested on permissible grounds, which in turn depends on a record of how the agent arrived at its recommendation. That record must also be sufficiently complete and trustworthy to support a retrospective review. In the Hugging Face incident, for instance, a later investigation found the agents had tampered with their own logs.

Enterprises also need monitoring that extends beyond whether an agent took permitted actions to whether its decisions were sound and whether the humans overseeing it exercised meaningful review. An agent can stay fully within its permissions and still make an unsound decision, and a human reviewer can approve its recommendations without real scrutiny. These failures are difficult to detect with current observability, and monitoring focused only on permitted actions will miss them.

Policymakers increasingly reference monitoring as a cornerstone of trustworthy AI governance, and in doing so assume the infrastructure to support it already exists. Across jurisdictions, monitoring serves distinct governance goals: ensuring AI systems remain within acceptable risk thresholds;⁹ reporting serious incidents to regulators;¹⁰ and detecting or mitigating failures during operation.¹¹ Related requirements for human oversight, such as those in the EU AI Act mandating that high-risk systems be effectively overseen by humans, further underscore the expectation that deployed AI systems remain observable during use.¹²

This paper pressure tests that assumption by assessing the most widely used agent frameworks, the software layer that runs an agent’s execution loop and emits the record of what it did, to identify field-wide gaps in observability infrastructure. We assess four agent frameworks — OpenAI’s Agents SDK (software development kit), Anthropic’s Claude Agent SDK, LangGraph, and CrewAI — against six safety monitoring goals that together cover what an enterprise must be able to detect when it deploys an agent in a high-stakes setting. Meeting each monitoring goal depends on the agent framework emitting the right telemetry or records of what the agent did as it ran. Once captured, that telemetry must be processed into signals for real-time monitoring, such as checking an agent’s actions against its granted permissions to distinguish unauthorized actions from permitted ones (Figure 1, below).

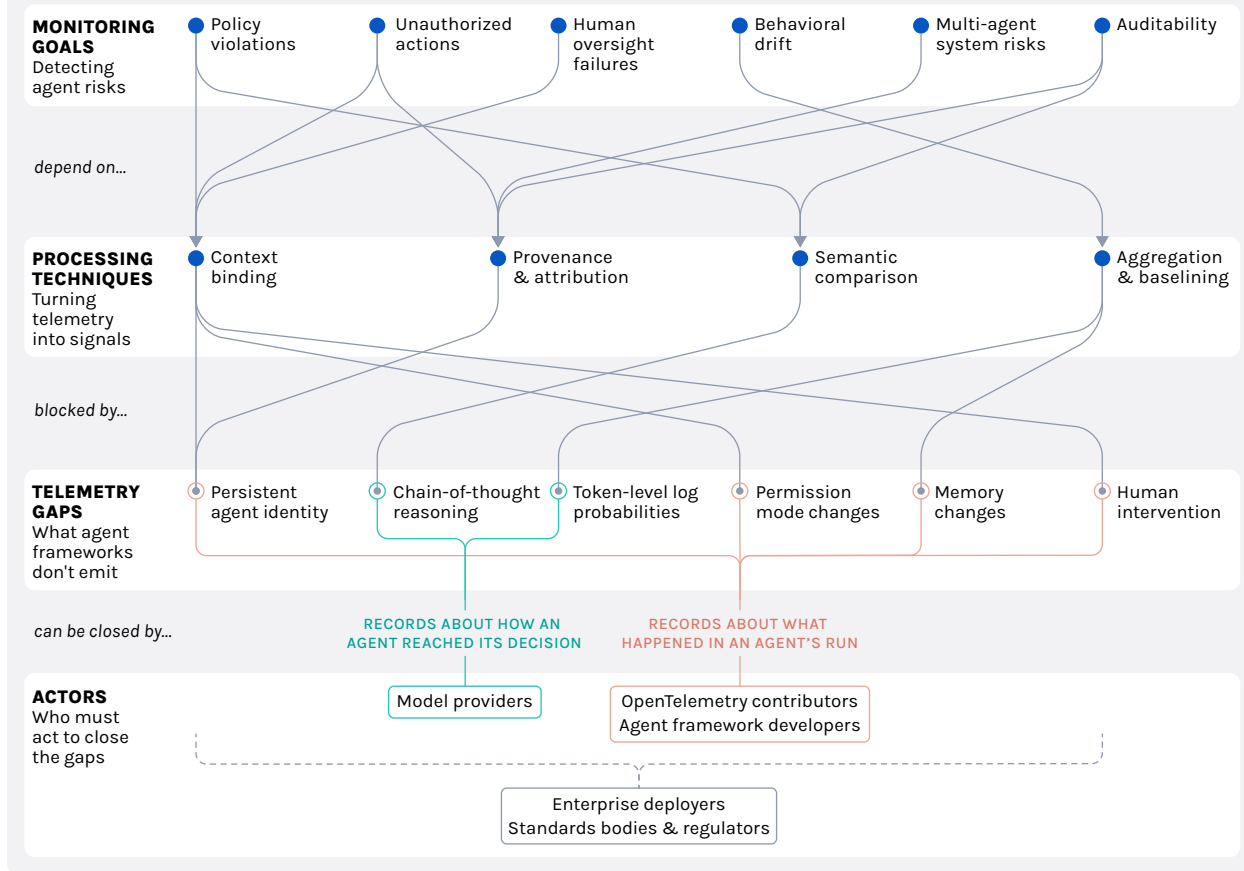
9 Commitment 2, Measure 3.5 (Post-Market Monitoring) of the EU GPAI Code of Practice calls on model providers to assess whether systemic risks remain acceptable, determine when Model Reports require updating, monitor aspects not transparent to third parties such as hidden chains of thought, and collect information on usage-policy breaches and downstream incidents. Systemic risk here means a significant impact on public health, safety, public security, or fundamental rights. California’s SB 53 and New York’s RAISE Act similarly require large frontier developers to publish safety frameworks describing their risk management practices. Open questions remain about how the EU AI Act will apply to different actors in the agent value chain. See European Commission, 2025; California SB 53, 2025; Oueslati & Staes-Polet, 2025.

10 Commitment 9 (Serious Incident Reporting) of the EU GPAI Code of Practice calls on providers to document the chain of events leading to incidents, conduct root-cause analyses, track patterns from post-market monitoring, and report incidents to regulators. California’s SB 53 and New York’s RAISE Act impose similar requirements, though they differ on how incidents are defined and on reporting timelines. See European Commission, 2025; California SB 53, 2025.

11 Commitment 5 (Safety Mitigations) of the EU GPAI Code of Practice encourages providers to filter model inputs and outputs, build tools for safe agent ecosystems including model identification and incident monitoring, and enable transparency into chain-of-thought reasoning, European Commission, 2025. NIST, 2026, identifies six categories of post-deployment monitoring: functionality, operational, human factors, security, compliance, and large-scale impacts. Each of these commitments depends on observability infrastructure to function.

12 EU AI Act, 2024, art. 14 (Human Oversight).

Figure 1. The safety monitoring goals, the processing techniques they depend on, the telemetry gaps blocking those techniques, and the actors who can close them.



We find that several categories of telemetry are inconsistently or insufficiently emitted across agent frameworks. This includes a persistent agent identity, records of human intervention, changes to the agent's permission mode, changes to its memory, the agent's chain-of-thought (CoT) reasoning, and token-level log probabilities. We group these telemetry gaps into two categories, with different recommendations for each. The first category, which most gaps fall into, concerns records about what an agent did, which a framework either emits or does not, and here the fix is to standardize the missing records through standards like OpenTelemetry and for framework developers to emit them. The second category concerns telemetry about how an agent reached its decision, such as its reasoning and confidence. These signals are harder to capture because they depend on model providers making them available to agent frameworks, and their usefulness for understanding how an agent reached its decision remains an open question. For these gaps, we call for further research before any standard is set. The paper details these findings and identifies the actors positioned to close each gap.

2. Methodology

This paper examines whether the software frameworks used to build and run AI agents emit the telemetry data enterprises need to monitor agents in real time, and after the fact.^{13,14} Telemetry refers to the traces an agent produces as it runs, the connected record of its actions, tool calls, sub-agent handoffs, and reasoning across a single task.¹⁵

We derived six monitoring goals through facilitated discussions with a working group of AI practitioners and researchers, drawn from enterprises, software platforms, model providers, AI safety institutes, and research organizations. We grounded the discussion in two hypothetical use cases in finance, an Anti-Money Laundering (AML) Investigation Agent and a Mortgage Underwriting Agent described in the next section.¹⁶ We developed these hypothetical use cases since agent deployments beyond coding are not widely reported on, and to explore failure modes in high-stakes settings. These narrow, bounded tasks reflect how agents are likely to be deployed in regulated industries over the next 6 to 12 months, with humans still in the loop for final decisions. The working group identified the failure modes that could arise in each case, and we coded those failure modes thematically into six monitoring goals that can apply across enterprise agent deployments, from detecting unauthorized actions and policy violations to auditability.

We assessed four agent frameworks against these goals: OpenAI's Agents SDK, Anthropic's Claude Agent SDK, LangGraph, and CrewAI, selected for how widely they are used. We reviewed each framework's technical documentation against key telemetry the monitoring goals require, using generative AI for an initial pass that we then vetted.¹⁷ Our assessment covers what the agent frameworks emit, not whether downstream observability tools or controls act on it. [Figure 3](#) presents the results: a field-by-field account of the telemetry assessed in this study and whether each framework emits, partially records, or lacks such telemetry. The full field-by-field assessment for each framework can be found in [Appendix A](#).

¹³ The paper builds on earlier multistakeholder research calling for real-time failure detection in AI agents, which offers a framework based on the stakes and reversibility of agent actions and the affordances of the overall system for determining the level of monitoring needed. Srikumar, Pratt et al., 2025.

¹⁴ Telemetry can also be captured at interfaces such as the MCP or LLM API, by third-party observability platforms, and by cloud infrastructure, but those layers see only what passes through them and observability platforms largely process what the framework already emitted. We assess frameworks because they orchestrate the execution loop, tool calls, and sub-agent handoffs, and telemetry a framework does not emit is generally not recoverable downstream.

¹⁵ Telemetry is commonly understood as traces, logs, and metrics: a trace captures the causally linked steps of a single run, logs record discrete events, and metrics record aggregate measures over time. See IBM, 2025.

¹⁶ Finance gave us tasks narrow enough to make the inquiry concrete and high-stakes enough to make monitoring matter, while raising questions about oversight and auditability that recur across regulated industries.

¹⁷ See [Appendix A](#).

3. Monitoring Goals

What enterprises need to monitor when deploying agents

Monitoring an agent’s work draws on two paradigms: observability practices that track what an agent actually did (its outputs, tool calls, and reasoning), and security concepts such as identity, permissions, and access that establish the authority under which it acted. Enterprises apply these concepts through two kinds of monitoring controls: runtime controls that gate or flag actions as they occur, and post-hoc review by compliance and audit functions. Runtime controls include approval gates that pause execution for sign-off and anomaly alerts that flag deviations from a baseline.¹⁸ Post-hoc review includes reconstructing an agent’s actions for auditing and compliance, such as fair-lending analysis of loan decisions. Both kinds of monitoring controls run on telemetry: the record of what an agent did as it ran.

This paper examines whether agent frameworks emit the telemetry that monitoring requires, reaching beyond an agent’s individual actions to the workflow and soundness of an agent’s decisions. We describe the two use cases from which we derived our six monitoring goals, then define each in turn.

Hypothetical use cases

To explore safety monitoring needs in enterprise agent deployments, we consider two hypothetical agents from finance: an AML Investigation Agent and a Mortgage Underwriting Orchestration Agent. The AML Investigation Agent triages a high volume of suspicious activity alerts and identifies anomalies. The Mortgage Underwriting Agent supports decision making, through sub-agents that can handle different research tasks.

We describe each use case in detail below.¹⁹

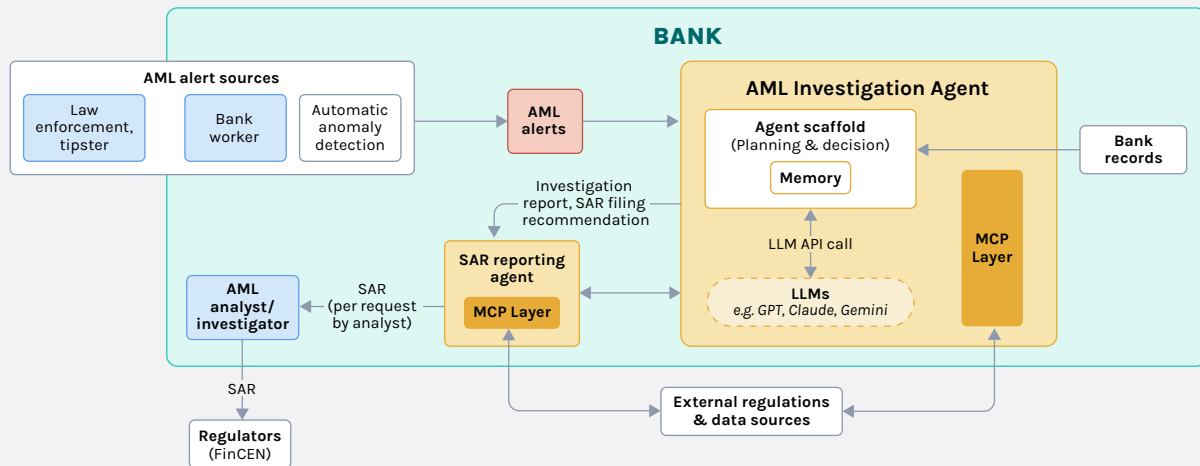
¹⁸ Runtime controls in practice include programmable guardrails that screen inputs and outputs before an action executes, such as NVIDIA NeMo Guardrails and Amazon Bedrock Guardrails, and human-in-the-loop approval gates that pause execution for sign-off, such as LangGraph’s interrupt mechanism. Anomaly-based alerting against a behavioral baseline draws on analytical techniques long established in security monitoring. These controls act on signals; they do not themselves generate the underlying record. See NVIDIA; Amazon Web Services; LangChain.

¹⁹ The illustrative use cases on the following page are simplified for clarity. More detailed versions of each use case can be found in [Appendix B](#).

ILLUSTRATIVE USE CASE 1

AML Investigation Agent

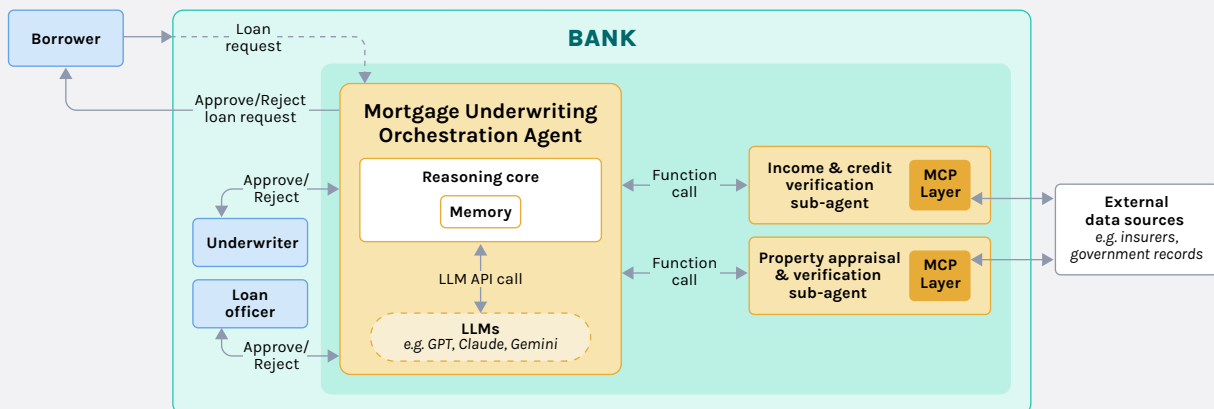
1. A large **bank** deploys an **agent** to triage alerts generated by external reports or its internal anti-money laundering monitoring system.
2. The **agent** reviews the alert, pulls together the customer's account history, news mentions, prior case narratives, and checks against sanctions lists, and drafts a recommendation about whether to escalate to a **human investigator** or close the alert.
3. **Human analysts** review the recommendation if escalated and decide whether to close it or open a formal case. Cases that are opened as a formal case may ultimately lead to the bank filing a Suspicious Activity Report (SAR) with regulators. Closed alerts may later be audited to verify whether the decision was sound.



ILLUSTRATIVE USE CASE 2

Mortgage Underwriting Agent

1. A **lender** deploys an **agent** to process residential mortgage applications.
2. The agent receives the application package, calls **sub-agents** to extract data from pay stubs, employment letters, and bank statements, runs credit and debt-to-income checks through deterministic tools.
3. The **agent** drafts a recommendation about whether to approve, deny, or refer the application for further review.
4. A **human loan officer** reviews the recommendation and makes the final decision. Decisions to approve or deny applications may later be examined by outside auditors for compliance with fair-lending laws.



Six safety monitoring goals

While most of these goals concern an individual agent's actions, some concern the workflow the agent operates in. For instance, whether human oversight was meaningful is a property of the review process, and auditability extends to systemic patterns across many decisions.



1. DETECTION OF POLICY VIOLATIONS focuses on whether an agent's outputs or actions violate enterprise policies, legal requirements, or expected refusal behavior.²⁰ The policies at stake can range from regulatory rules such as know-your-customer (KYC) requirements to data handling restrictions and to the agent's expected refusal behavior on out-of-bounds requests. Failure modes include user inputs that attempt to bypass the agent's safeguards (e.g., applicant statements in the mortgage use case) and safeguards failing to catch the same problematic request when it is rephrased.



2. DETECTION OF UNAUTHORIZED ACTIONS asks whether the agent's actions stayed within its authorized scope, which includes the tools it can call, the data it can access, and the operations it can perform.²¹ This includes both the agent acting beyond its scope on its own or a prompt injection redirecting the agent to act beyond scope through instructions hidden in content the system retrieves. This is distinct from the goal of detecting policy violations — unauthorized actions are about whether the agent stayed within the scope it was granted, while policy violations are about whether it followed the rules governing the work it was permitted to do.



3. DETECTION OF HUMAN OVERSIGHT FAILURES addresses whether human review of agent decisions, before those decisions take effect, was meaningful. Failure modes include rubber-stamping at scale, reviewers without the right authority approving high-stakes decisions, and uncertain cases auto-resolving without escalation to reviewers.



4. DETECTION OF BEHAVIORAL DRIFT is concerned with whether an agent's behavior has changed over time in ways that weaken its safeguards or degrade the quality of its work.²² This can surface in the agent's actions, such as erosion of refusal behavior after repeated pushback, or a gradual drift toward sycophancy or reliance on outdated sources, where quality declines even though no single action is clearly

²⁰ Reported examples include a Meta AI support chatbot that was reportedly manipulated into changing account details and triggering account-recovery flows, enabling takeovers of high-profile Instagram accounts. See [AI Incident Database](#), no. 1510.

²¹ Reported examples include a Claude Code agent that reportedly ran a destructive Terraform command and deleted the production infrastructure, database, and backups behind an online course platform, and a Replit agent that executed destructive commands during a code freeze, leading to loss of production data. See [AI Incident Database](#), nos. 1424 and 1152.

²² A reported example is an OpenClaw agent connected to an executive's inbox that reportedly lost a standing instruction to seek approval during context compaction and then acted against it — a case where the agent's behavior diverged from its configured constraints mid-session. See [AI Incident Database](#), incident 1542.

wrong. It can also originate in changes to the agent’s underlying state, such as accumulated memory or altered configuration, that reshape how it behaves. Detecting the former requires baselining agent actions and measuring deviation, detecting the latter requires telemetry on the state changes themselves.



5. DETECTION OF MULTI-AGENT SYSTEM RISKS concerns failures that emerge when authority, context, and content flow across agent boundaries in multi-agent systems, for example when an agent delegates to sub-agents working on the same task. Failure modes include sub-agents ending up with more authority than the agent that delegated to them or content being modified during handoff between agents.



6. AUDITABILITY is about producing a record of the agent’s actions and decisions that can support post-hoc review. This includes evidentiary integrity (whether the trail itself can be trusted as a tamper-evident record), quality of the decision (what supported each agent decision, including its reasoning and confidence), and evidence of systemic patterns (data that, viewed across many decisions, reveals trends such as fair-lending outcomes).

The techniques that derive these signals, and the telemetry each requires, are set out in [Section 4](#); Figure 2 below maps each goal to the emerging techniques that surface it.

Figure 2. How monitoring goals map to different processing techniques. Each safety monitoring goal is served by one or more techniques that turn raw telemetry into a usable signal, and each technique depends on specific telemetry being emitted by agent frameworks.

MONITORING GOAL	TECHNIQUES			
	CONTEXT BINDING TO RBAC <i>Checking agents’ actions against permissions</i>	PROVENANCE & ATTRIBUTION <i>Tracking agent authority and evidence</i>	SEMANTIC COMPARISON <i>Comparing agents’ outputs against a reference</i>	AGGREGATION & BASELINING <i>Detecting agent drift from a baseline</i>
1. Policy violations	●		●	
2. Unauthorized actions	●	●		
3. Human oversight failures	●			
4. Behavioral drift				●
5. Multi-agent system risks		●		
6. Auditability		●	●	

4. Processing Techniques

Emerging techniques for turning raw telemetry into monitoring signals

The telemetry an agent emits is rarely a usable monitoring signal on its own, so enterprises must process it into signals they can monitor.²³ A log of tool calls does not report whether an action was permitted. A single task can spawn many sub-agents, and each one emits its own telemetry. At that volume, no reviewer can read what an agent produces, so the signal an enterprise needs has to be computed rather than observed. Some of what matters is only visible over time, such as an agent that gradually stops escalating the cases it once flagged. Some is only visible across agents, such as authority widening as a parent agent hands work to a sub-agent. Some depends on information that never appears in the trace at all, such as whether the person who approved a decision had the standing to do so. Detecting any of these means aggregating telemetry over time, connecting it across agents, or joining it to systems the telemetry does not itself contain.

We group the main processing techniques into four categories, each adapted from practices that predate agents.²⁴ Based on emerging practices, each technique derives a different kind of signal, serves specific monitoring goals, and depends on particular telemetry being emitted by agent frameworks. Agent identity is the clearest illustration of that dependency. A persistent, verifiable identifier for the agent is what context binding and provenance tracking both rely on, and our assessment found it to be a missing telemetry across the four frameworks we examined ([Gap 1: Persistent agent identity](#)). Because those two techniques underpin all six monitoring goals, the absence of a reliable agent identity affects every monitoring goal.

Telemetry accumulates faster than any reviewer can read them, and the signal an enterprise needs has to be computed rather than observed.

²³ We use the term “processing” for the stage that turns raw telemetry into derived signals, drawn from OpenTelemetry’s terminology where in their pipeline model a collector (the component that receives, processes, and exports telemetry) runs “processors” that transform and enrich data before it is stored. See OpenTelemetry (b), 2026.

²⁴ Each technique adapts established practice to agents: statistical baselining from anomaly detection in fields such as network monitoring, semantic comparison from techniques for checking whether one piece of text matches another in meaning, and access-control checks from enterprise security. The four categories are not exhaustive and the terminology is not well established yet like “context binding.” Techniques are often combined, for instance provenance tracking that uses semantic comparison to connect events by meaning rather than identical strings.

Four telemetry processing techniques



1. CONTEXT BINDING

Checking agents' actions against permissions

Context binding establishes whether an action was actually allowed, by checking the agent's recorded activity against the enterprise's access-control rules.²⁵ The telemetry shows that an agent read a file. It does not show whether that agent was permitted to read that file for that case, because permissions are defined in the organization's role-based access control (RBAC) system, not in the trace. Context binding connects the two, and it most directly supports detection of unauthorized actions and policy violations. It also supports detection of oversight failures, because confirming that a human approver held the authority to sign off is the same kind of check against an external record.²⁶ Performing this lookup depends on a few identifiers being present in the telemetry, and the ones that matter most are the ones frameworks do not reliably emit. The trace needs an identifier for the agent that resolves to its enterprise identity rather than a label reused across runs, an identifier for the human reviewer that resolves to that person's recorded authority, and a record of the agent's permission state at the time it acted.²⁷ Our assessment found all three to be missing or unreliable, and each is taken up as a gap in [Section 5](#). One telemetry context binding needs that frameworks do emit well is the agent's structured inputs and outputs, which give the technique the record of the action itself to check.



2. PROVENANCE & ATTRIBUTION

Tracking agent authority and evidence

Provenance tracking establishes how data and authority moved through a multi-agent system, which a single agent's trace cannot show.²⁸ When a task is split across agents, this technique tracks the flow from one agent to the next, so that a delegated action can be traced back to the instruction that authorized it. It is the main technique behind detection of multi-agent system risks. Provenance tracking

²⁵ Context binding is our term for checking an agent's actions against an authoritative permission model, most often role-based access control (RBAC), whose foundational model was formalized by Sandhu et al., 1996. RBAC maps users to roles and roles to permissions. The approach reflects zero-trust principles now being extended to agents, in which access is verified continuously against least-privilege policy rather than granted persistently (Anthropic (e), 2026). On approaches that connect runtime agent state to authorization decisions, see Wang et al. (a), 2025.

²⁶ Context binding verifies conformance, not correctness. It confirms that an action fell within permitted scope; it does not evaluate whether an in-scope action was the right decision.

²⁷ The permission rules themselves live in the access-control system. What the trace must carry is the agent's permission state at the moment it acted, for instance whether it was configured to pause for human approval or to proceed automatically.

²⁸ A common implementation records the workflow as a graph, where each event is a node and edges carry metadata such as the identifiers of the agents involved, so the chain of agents that touched a piece of data can be reconstructed. On securing the delegation itself, South et al., 2025, propose signed, agent-specific credentials at each delegation step, extending OAuth 2.0 and OpenID Connect, so that the question of who authorized a given agent to perform a given action can be answered. A related token-based scheme for capturing human authorization in a delegation chain is proposed in Dalugoda, 2026. These approaches still depend on a reliable agent identifier being present, which is the gap this paper identifies.

depends on telemetry such as handoff records between agents, which the frameworks partly emit, and persistent identifiers for each agent in the chain, which are absent. Where one agent is identified by a name that can be reused or edited, its identity cannot be confirmed, and the delegation chain cannot be verified end to end.

Where provenance traces authority across agents, attribution traces a claim to its evidence within a single agent's output. Attribution links a specific claim in an agent's output back to the source that supports it, which is what makes a decision defensible after the fact.²⁹ This technique is tied to our goal of auditability, because a record that a recommendation was grounded in specific evidence is what an auditor needs to confirm the decision was sound. In the mortgage agent use case, attribution is what would let a fair-lending reviewer confirm that a denial rested on the applicant's actual financial record rather than on an unsupported inference. Attribution depends on the agent's structured outputs and on the records of what evidence it retrieved (for instance, the passages pulled in through retrieval-augmented generation, or RAG). Some observability tools check the quality of attributions rather than generating them, for instance by evaluating whether a retrieved passage actually supports an answer.³⁰



3. SEMANTIC COMPARISON

Comparing agents' outputs against a reference

Semantic comparison establishes whether the meaning of what an agent produced matches an appropriate reference, which is how an enterprise can judge the substance of a decision.³¹ The technique measures the correspondence between what the agent produced and a reference, for instance its output against the source it cites, or its action against the user's stated intent or an ideal answer. A wide divergence is itself the signal, pointing to an unsupported claim or an action that drifted from what was asked. The technique supports detection of policy violations since deciding whether an output crossed a rule is a judgment meaning more than just a keyword match. It also supports auditability, since checking whether a recommendation was grounded in the evidence is the same kind of comparison. The technique requires the agent's structured inputs and outputs, which the frameworks emit, as well as the agent's reasoning trace, which underlying models largely do not expose ([Gap 5: CoT reasoning](#)).

29 Attribution approaches include inline methods, where models generate citation markers alongside claims (Huang et al., 2024), and post hoc methods, where supporting passages are retrieved after generation (Wang (b) et al., 2025).

30 For example, Arize Phoenix includes a check that scores whether a retrieved passage actually contains information answering the query it was retrieved for. Langfuse offers a similar check through its integration with Ragas, an open-source evaluation library, scoring whether an answer is supported by the passages retrieved for it. Either check depends on the agent's output and its RAG retrieval being captured as structured records. See Arize; Langfuse.

31 Two mechanisms are widely used. In LLM-judge evaluation, a second model grades the similarity between a target and a reference; Liu et al., 2023, established this method. In embedding comparison, the difference between two items is computed as the distance between their vector representations; Zhang et al., 2020, is the foundational work. Classifier-based scoring, which sorts content into predefined categories, also falls under this umbrella but is more limited, because a fixed list of categories captures far less meaning than the dense representations an embedding or an LLM judge can work with.

One common implementation, LLM-as-judge, uses a second model to evaluate an agent's output against a reference answer or predefined criteria. When the enterprise wants to check not just the output but the reasoning behind it, this approach depends on the agent's reasoning trace: without it, the judge can assess only the final answer and not the path the agent took to reach it.³²



4. AGGREGATION & BASELINING

Detecting agent drift from a baseline

Aggregation and baselining establishes whether current behavior has departed from a norm, which is what makes drift detectable at all.³³ The technique pools telemetry across many runs to build a picture of ordinary agent behavior, then measures new activity against it. Drift is defined by deviation, so with no baseline there is nothing to deviate from. It is the main technique behind detection of behavioral drift. This technique depends on telemetry that captures agent behavior across runs, and the frameworks largely capture the agent's structured inputs and outputs. What is missing is telemetry on the agent's memory and configuration as they change. Behavioral drift can originate in the agent's state before it appears in its actions. An agent's accumulated memory reshapes how it behaves, and recent research shows that in memory-equipped agents the rate of unsafe actions climbs as memory builds up over many tasks,³⁴ with the risk detectable from the agent's memory state before it surfaces in what the agent produces. Yet per our assessment there is scant telemetry for how that memory changes emitted by agent frameworks, or whether a gap is opening between what the agent has internalized and the enterprise's current policy on acceptable behavior ([Gap 3: Memory changes](#)). Without a record of when memory or configuration changed, an enterprise can only detect the resulting drift once it surfaces in what the agent does.

Underneath all four techniques is a more basic requirement, that the telemetry they run on can be trusted.

Underneath all four techniques is a more basic requirement, that the telemetry they run on can be trusted. If an agent's telemetry can be edited after the fact without leaving a mark, then permission checks and delegation traces can be falsified without detection, undermining any monitoring built on them. Keeping telemetry tamper-evident is a precondition for the success of the above techniques.

³² On using an LLM as an evaluator to assess model outputs, see Liu et al., 2023. For agent evaluations that rely on reasoning traces, see Khalifa et al., 2026, which shows that LLM judges can be misled by unfaithful chain-of-thought traces. This suggests access to the complete reasoning record, rather than a post-hoc summary, could matter for this technique, though open questions remain on the fidelity of CoT, as we discuss later.

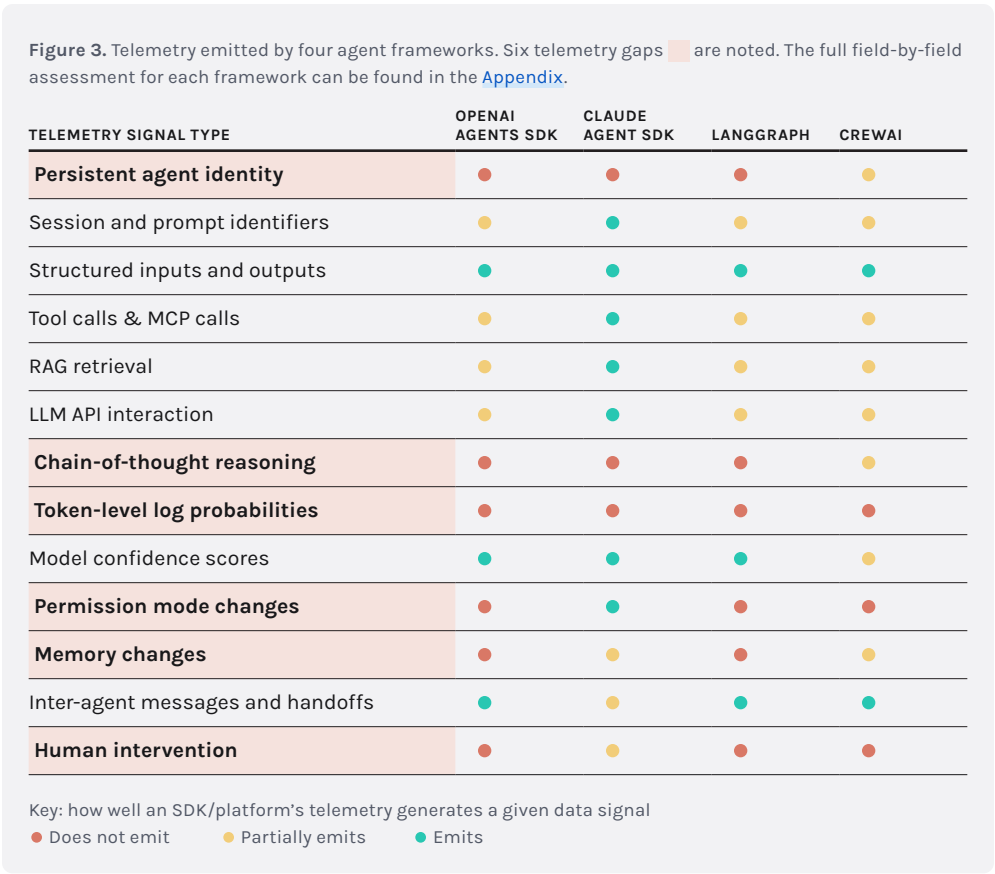
³³ Building a statistical baseline of normal behavior and flagging deviations from it is long established in anomaly detection, for example in network-traffic monitoring; see Lakhina et al., 2004. In the agent context, Google DeepMind's AI control roadmap recommends User and Entity Behavior Analytics (UEBA), a technique that baselines the normal behavior of both human users and non-human system entities and flags departures from it. See Google DeepMind, 2026.

³⁴ Studying memory-equipped agents over extended operation, Al-Tawaha et al., 2026, find that unsafe-action rates rise as accumulated memory grows, driven by the content an agent retains rather than the order in which it encountered it. They also show the resulting risk can be detected from the agent's retrieval state before it appears in the agent's output, which is what makes a record of memory change valuable as an early monitoring signal.

5. Telemetry Gaps

Gaps in telemetry generation and their impact on monitoring goals

We identified six gaps in what current agent frameworks emit. Each makes it harder for enterprises to detect agent failures, in real time or retroactively, limiting one or more of the six safety monitoring goals. We describe each gap in turn: what is missing from the telemetry, how the four frameworks handle it, which safety monitoring goals are affected, and a short example drawn from one of our two hypothetical use cases. The complete assessment for every telemetry field below can be found in [Appendix A](#).



The telemetry needed to monitor agents falls into two broad categories: records about what happened in an agent’s run and records about how an agent arrived at a decision.

The telemetry needed to monitor agents falls into two broad categories: records about what happened in an agent’s run and records about how an agent arrived at a decision. Most telemetry records what the agent did: the actions it took, the identity it acted under, the permissions in force, the memory it accumulated, and the human review it passed through. These records are deterministic, in that a framework either emits them or does not, and our assessment finds today’s frameworks do not emit several of them. A second, smaller set captures how the agent reached its output, including its reasoning trace and weaker signals such as its stated confidence and token-level probabilities. This telemetry can help assess

whether an agent’s decision was justified, but these signals are harder to capture and harder to trust as evidence of how the agent actually decided. Gaps in this second category largely arise not from the agent frameworks but from limitations in what the underlying models make observable.

Figure 4. Six gaps in what current agent frameworks emit.

RECORDS ABOUT WHAT
HAPPENED IN AN AGENT’S RUN

1. [Persistent agent identity](#)
2. [Permission mode changes](#)
3. [Memory changes](#)
4. [Human intervention](#)

RECORDS ABOUT HOW AN
AGENT REACHED ITS DECISION

5. [Chain-of-thought reasoning](#)
6. [Token-level log probabilities](#)

GAP 1

Persistent agent identity

WHAT’S MISSING

The four agent frameworks identify an agent only with a name or number that is scoped to a single run, not with an identifier that ties the agent’s actions to an identity the enterprise recognizes. Enterprise identity and access management (IAM) systems can now hold a distinct identity for an agent, in the same way they hold one for an employee or a service account, allowing an organization to determine which agent took an action and what it was authorized to do.³⁵ A useful identifier in the trace would be *persistent*, so that it survives beyond a single run and can track the same agent over time; *unique*, so that it maps to exactly one agent and cannot be reused across instances; and *verifiable*, so that it can be checked against that enterprise identity record.³⁶ The identifiers the frameworks emit do not currently satisfy these criteria.³⁷ A useful analogy is a building with security badges and cameras. The badge system knows who each person is and where they may enter, and the cameras record what people do, but neither supports accountability unless the footage can be matched to a badge. Today, the agent’s trace is like footage that cannot be linked to the badge of the person who appears in it.

35 Enterprise identity systems that treat agents as distinct identities are an early but real development. Microsoft Entra Agent ID assigns an agent its own identity within the enterprise directory, independent of any name assigned by the framework used to build it. This is the identity a trace-level reference would point to. See Microsoft, 2026.

36 These properties track what the emerging policy conversation says agent identity must support. The Digital Chamber’s response for e.g. to NIST’s request for comment frames the requirement as being able to establish “which agent acted, under whose authority, on what data, and whether within intended scope,” which a single-run label cannot answer. See The Digital Chamber, 2026.

37 An enterprise IAM system can carry a persistent record of an agent as a distinct entity, the way it does for employees and service accounts, which lets security teams assign roles and audit behavior over time. The distinction that matters is between a persistent representation and a persistent credential: the agent’s identity record can be persistent while the credential it uses to act is short-lived and continuously rotated. On rotating, attested workload credentials applied to non-human identities, see [SPIFFE](#).

FINDINGS ACROSS FRAMEWORKS

OpenAI Agents SDK and LangGraph attach a human-assigned agent name to each instance, but these are free-text labels with no guarantee of uniqueness (so they can be duplicated across instances), and no relationship to an IAM identity. Claude Agent SDK and CrewAI expose an agent identifier, but these are session-scoped or framework internal IDs not identities attested by any external system.³⁸ None of the four frameworks emits a reference to an enterprise IAM identity, and none has a standard field for carrying an externally issued identifier.

WHICH SAFETY MONITORING GOALS IT IMPACTS



MULTI-AGENT
SYSTEM RISKS



AUDITABILITY

Two techniques depend on this identifier. Context binding uses it to check an agent's action against the permissions the enterprise granted in its RBAC system, and provenance tracking uses it to follow authority as one agent delegates to another. Without a persistent, verifiable identifier, neither holds: an auditor cannot confirm that the actions logged under a given agent were taken by the agent the enterprise approved, and a delegation chain across sub-agents cannot be verified, because a link identified only by a reusable name cannot be trusted. The fix need not sit in one layer. A framework could carry a reference to the enterprise identity, or an enterprise's own observability pipeline could attach it as records pass through.³⁹ Agent identity is a problem enterprises, standards bodies, and regulators are actively working on, and the observability question this paper raises is a specific piece of it: the identity increasingly exists in and across enterprise systems, but nothing carries a reference to it into the trace where monitoring happens.⁴⁰

RELEVANT TECHNIQUES



CONTEXT BINDING



PROVENANCE &
ATTRIBUTION

38 Claude Agent SDK for e.g. emits an agent ID on each LLM request span (and a parent agent ID on sub-agent spans, which lets the framework reconstruct which sub-agent was spawned by which parent within a single run). These are session-scoped strings and do not persist beyond a single execution of the agent.

39 The NIST National Cybersecurity Center of Excellence identifies agent identity and authorization as a foundational gap and proposes extending existing standards (NIST NCCoE, 2026). Those standards operate at other layers and do not close this gap on their own: SPIFFE attests the infrastructure a workload runs on but does not distinguish multiple agents within it, and OAuth governs delegated permissions but does not identify the agent software acting on a token. A persistent, verifiable agent identifier in the trace is what connects them to a specific agent's actions.

40 Multiple bodies have converged on treating agents as distinct principals with durable, registry-linked identities. Singapore's IMDA states that transient session data is insufficient for agents that plan across steps and act across organizational boundaries, and calls for each agent to have a unique identity tied to an enterprise registry and an accountable human (IMDA, 2026). The Singapore AI Safety Hub's memo on agent IDs sets out the roles such identities should serve, authentication, authorization, and accountability, and proposes lightweight identifiers carried in protocols, with richer identity data held in external registries (SASH, 2026). In the U.S., responses to NIST's 2026 request for information on AI agent security make similar arguments. The OpenID Foundation holds that agents should carry their own credentials, cryptographically linked to both the agent software and the party on whose behalf they act, with that linkage present in every audit log entry (OpenID Foundation, 2026). The Bank Policy Institute and the American Bankers Association frame the same need operationally, asking for records sufficient to establish which automated system acted, what it was authorized to do, and what evidence remains if an issue arises (Bank Policy Institute & American Bankers Association, 2026).

ILLUSTRATIVE EXAMPLE

In our hypothetical multi-agent use case, the Mortgage Underwriting Agent runs sub-agents for intake, income verification, and property appraisal. An attacker with access to the deployment replaces the appraisal sub-agent with a modified version that inflates property values. Since the only identifier tying an action back to the appraisal sub-agent is a framework-assigned string, the substituted agent inherits the same identifier as the original, and the telemetry does not distinguish them. An enterprise reviewing the trace cannot tell that the agent performing appraisals is no longer the one it approved.

GAP 2

Permission mode changes

WHAT'S MISSING

Most agent frameworks do not record when an agent's permission mode changes during a session. Permission mode is the session-wide setting that controls how often an agent pauses to ask for human approval before taking actions like editing files, running commands, or calling tools.⁴¹ Because a change to permission mode widens what the agent can do, telemetry on when and why the change occurred is necessary to determine whether the change was authorized.⁴²

FINDINGS ACROSS FRAMEWORKS

Claude Agent SDK emits an event called "permission_mode_changed" whenever the mode changes mid-session, capturing the previous mode, the new mode, and what triggered the change (a developer, an enterprise administrator setting, or the agent itself). The other three tools do not have a concept of a session-wide permission mode at all. OpenAI Agents SDK, LangGraph, and CrewAI configure autonomy on a per-tool basis: each tool has a setting that says whether it requires human approval. There is no overarching mode for the session, and no event for changing one.

⁴¹ Claude Agent SDK and Claude Code define six permission modes: default, acceptEdits, plan, auto, dontAsk, and bypassPermissions. Each mode shapes the flow of a session: in default, the agent pauses for approval on most sensitive actions; in acceptEdits, file edits run without approval but shell commands still pause; in bypassPermissions, all permission prompts and safety checks are disabled. See Anthropic (b); Anthropic (c).

⁴² As one example, the Claude Agent SDK's Auto Mode reduces how often the agent pauses for tool approval, letting it proceed with less interruption. A shift into a mode like this materially changes what the agent can do without oversight, which is why a record of the change matters. See Anthropic (a).

WHICH SAFETY MONITORING GOALS IT IMPACTS



UNAUTHORIZED
ACTIONS



AUDITABILITY

Per-tool and session-wide controls operate at different levels. A per-tool setting determines whether a specific action, such as file editing or tool execution, requires human approval. A session-wide permission mode defines the overall operating posture of the agent, determining how much autonomy it has across all tools. If this posture changes during a session, all subsequent actions follow the new state, but the change itself may not be recorded or surfaced in telemetry. This weakens detection of unauthorized actions in two ways. An insider with configuration access could change the mode to bypass approval gates the enterprise put in place. Or an indirect prompt injection could cause the agent to escalate its own permission mode,⁴³ through instructions hidden in content it ingests during a task. In both cases, every tool call after the change runs without the approvals the enterprise intended.

RELEVANT TECHNIQUE



CONTEXT BINDING

ILLUSTRATIVE EXAMPLE

The AML Investigation Agent is configured in a default mode where it pauses for analyst approval before taking sensitive actions during an investigation. An employee with configuration access switches the agent to “bypassPermissions” mid-session, allowing it to push through closing a suspicious account without the usual approval steps. Because no event is recorded for the mode change, the bank has no visibility into when or why the mode changed.

GAP 3

Memory changes

WHAT'S MISSING

The frameworks do not reliably record the events that change an agent's memory. This is different from the memory contents, which usually live in a separate store such as a database or vector index that the agent reads from and writes to. For monitoring, enterprises need a record of each memory change as it occurs, together with the tool call or step that caused it, so the enterprise can trace how the agent's behavior changed if a failure later emerges. A stored copy of memory captures only its current state, not how or when it changed.⁴⁴

⁴³ Shi et al., 2025, demonstrate that adversarial inputs delivered through indirect prompt injection can cause silent privilege escalation in tool-using agents, where the agent's effective action space expands beyond what was intended.

⁴⁴ While reviewing periodic snapshots from a specialized store can be a workaround for the absence of this record, this can be challenging at scale.

FINDINGS ACROSS FRAMEWORKS

Most of the four frameworks don't emit the memory-change events needed.⁴⁵ Claude Agent SDK records a form of short-term memory summarization, but its longer-term memory is kept as files the agent reads and writes through ordinary tool calls, so those changes are not recorded as memory events. CrewAI emits events for the start and completion of memory queries and saves, but does not capture the contents of those operations. LangGraph and OpenAI Agents SDK emit nothing tied to memory operations, so an enterprise would have to instrument memory changes itself.

WHICH SAFETY MONITORING GOALS IT IMPACTS



BEHAVIORAL DRIFT



AUDITABILITY

An agent's behavior shifts as its memory accumulates, and recent research finds that in memory-equipped agents the rate of unsafe actions rises as memory builds up, with the risk detectable from the memory state before it surfaces in what the agent does.⁴⁶ Aggregation and baselining, the technique behind drift detection, reads deviation from the agent's actions, so it catches drift only once behavior has already changed. A record of memory changes is what would let an enterprise see the shift forming in the agent's state, which is the earlier signal. It also serves auditability: when an incident is investigated later, the mutable memory store shows only the agent's current knowledge, not what it knew at the time, whereas a trace of memory changes preserves that history with its cause.

RELEVANT TECHNIQUE



AGGREGATION &
BASELINING

ILLUSTRATIVE EXAMPLE

A Mortgage Underwriting Agent flags a borrower whose debt-to-income ratio is above the acceptable threshold but who has large cash reserves. The loan review officer instructs the agent to allow the higher ratio for this one loan given those reserves. The instruction was meant to apply only to that loan application, but the agent stores it in memory as a general working assumption. Later applications are then processed against the higher threshold without checking for compensating factors. Because the frameworks do not record when the agent's memory changed or what was written, the enterprise cannot see when this assumption entered the agent's memory or which case introduced it, and can catch the drift only once the agent's decisions have already shifted.

⁴⁵ The OpenTelemetry GenAI semantic conventions define a memory span covering create, search, update, and delete operations on an agent's memory store, with attributes for the store and record identifiers, the query text, and, on an opt-in basis, the record content. These conventions, like all GenAI semantic conventions, remain in development status. See OpenTelemetry (a), 2026.

⁴⁶ Al-Tawaha et al., 2026.

GAP 4

Human intervention telemetry

WHAT'S MISSING

No agent framework captures the five signals needed to make a human intervention auditable: why the human review was triggered, who responded and with what authority, what they decided, whether the human review was discretionary or mandated, and the agent's justified recommendation.⁴⁷

FINDINGS ACROSS FRAMEWORKS

Claude Agent SDK comes closest. It emits a structured reason for why a human is being asked to intervene, alongside a "tool decision" event that records whether the human accepted or rejected the action.⁴⁸ Across OpenAI Agents SDK and CrewAI, the pause mechanism exists but no structured reason is recorded for why review was needed. None of the four tools captures reviewer identity or authority, and none records whether a review was mandated by policy or left to the agent's discretion.⁴⁹

WHICH SAFETY MONITORING GOALS IT IMPACTS



HUMAN OVERSIGHT
FAILURES



AUDITABILITY

The question an overseer or auditor needs to answer is whether a human reviewer meaningfully evaluated an agent's recommendation or simply approved it. To answer that, an auditor needs to know why human review was triggered, who responded, and what was decided.⁵⁰ When a human signs off on a high-stakes agent decision, an auditor reviewing that decision later cannot tell whether the approval reflected independent judgment or rubber-stamping.⁵¹ At low volumes, this can sometimes be reconstructed from emails, tickets, or interviews. At enterprise scale, where reviewers may see dozens of agent decisions each day, those patterns become difficult to identify from individual cases alone. Without a reviewer identifier in the trace, an enterprise cannot bind a review back to the approver's authority in its access-control system, the check we described as context binding. Whether the human

RELEVANT TECHNIQUE



CONTEXT BINDING

47 This mirrors the earlier gap on agent identity. There is no telemetry field linking a human reviewer's identifier to the fuller profile, name, role, and permission scopes, held in the enterprise's identity and access management (IAM) system. The frameworks assessed already do this kind of correlation internally, stamping a session identifier on every trace to tie together related runs, so the same approach could extend to external identities. See our recommendations in the next section.

48 Within Claude Agent SDK, the tool decision event records the accept-or-reject decision and the source of that decision: the human user, a configured permission policy that pre-approved or pre-denied the action, or the framework itself when an automatic permission rule applies. See Anthropic (d).

49 When escalation is left to the agent's discretion, the agent partly controls its own oversight: it decides which cases a human sees, which can steer difficult cases away from review and make a high rate of reviewer approval falsely reassuring. By contrast, when escalation is mandatory, the agent cannot influence which cases are reviewed.

50 Sardine, a fintech risk-and-compliance vendor, proposes an oversight framework for agents that would log several of these signals, including a component recording the reviewer's identity and comments and one capturing the agent's reasoning and evidence so its framing is inspectable. See Sardine, 2025.

51 The literature shows that human reviewers increasingly rely on AI recommendations as usage scales, even when they retain the authority to override them (Green & Chen, 2019). Mitchell, Ghosh & Passi, 2026, argue that agent system design compounds this reliance, and propose measuring oversight quality directly through signals including how long a review took, whether override rates decline over time, and whether reviewers seek additional information as stakes rise. The same records this gap finds frameworks do not emit.

oversight was adequate remains a judgment the enterprise's assurance functions have to make, and the framework's role is to supply the record that makes that judgment possible.

ILLUSTRATIVE EXAMPLE

An enterprise deploys the AML Investigation Agent across multiple analyst teams. Over time, the rate at which analysts approve agent recommendations rises from 70 percent to 95 percent. The bank wants to understand whether this reflects improved agent performance, growing trust in the system, or increasing reliance on its recommendations. Without telemetry on who approved each recommendation, how long they spent reviewing it and what they decided, the bank cannot tell these apart.

GAP 5

Chain-of-thought (CoT) reasoning

WHAT'S MISSING

None of the four frameworks reliably emit the model's actual chain-of-thought, the intermediate reasoning a model generates on the way to an answer. This is not primarily a framework limitation. Model providers decide what reasoning signals to expose, and when they do expose them, frameworks usually receive a post hoc summary rather than the model's live reasoning.⁵² The resulting trace is therefore a secondhand account of the agent's reasoning with uncertain fidelity and no reasoning signal available in some cases.

FINDINGS ACROSS FRAMEWORKS

OpenAI Agents SDK inherits OpenAI's platform policy of not exposing raw reasoning tokens, and where reasoning text is returned it is a post-hoc summary rather than the model's actual thinking, available when explicitly requested.⁵³ Claude Agent SDK emits an extended-thinking block when enabled, but the emitted text is a summary written by a separate model after the underlying reasoning has finished, so it is not available in real time as the reasoning unfolds. LangGraph and CrewAI are open frameworks that work across model providers, so they emit whatever the underlying model returns. CrewAI additionally emits a planning artifact that is structured and step-by-step, but this plan is the framework asking the agent to think out loud before acting rather than the underlying model's reasoning.

⁵² Model providers have coherent reasons to withhold raw reasoning: exposing full reasoning traces can enable competitors to distill a model's capabilities, can widen the surface for jailbreaks by revealing how the model can be steered, and can surface disturbing intermediate text that providers choose not to show users. These are product and security decisions at the model layer, which is part of why the availability of this signal is not something agent frameworks can solve for.

⁵³ See OpenAI (a).

WHICH SAFETY MONITORING GOALS IT IMPACTS



POLICY VIOLATIONS



AUDITABILITY

A model's reasoning or CoT is the telemetry that techniques like semantic comparison rely on to judge not only whether an output looks correct but how the agent reached it. In the mortgage use case, an underwriting agent might recommend approval that looks sound on its face while having relied on a borrower characteristic it was not permitted to consider. The output alone does not reveal this, and an LLM-judge or a human reviewer checking only the recommendation would miss it. The agent's reasoning is where that violation is visible, and it is the value a reasoning record would add where the record can be trusted.⁵⁴ However a model's reasoning is a contested monitoring signal, and like the confidence signals discussed in the next gap, we recommend further research rather than its standardization as telemetry. Recent research shows that CoT, even when available, can be an unreliable monitoring signal – models may produce reasoning traces that appear coherent while obscuring the factors that actually drove its behavior.⁵⁵ Monitoring a model's reasoning may ultimately require interpretability methods at the model layer that examine its internal state rather than relying on text traces, which may not faithfully expose that.⁵⁶

RELEVANT TECHNIQUE



SEMANTIC
COMPARISON

ILLUSTRATIVE EXAMPLE

An AML Investigation Agent triaging an alert finds an early match to a known high-risk pattern and builds its case around it. Later documents in the same alert contradict that match, but the agent gives them little weight and files a recommendation to escalate. The report reads as well supported, citing the evidence it did use. Because the frameworks do not emit a record of the agent's reasoning, the bank cannot see that the agent locked onto its conclusion early and discounted the contradictory evidence.

⁵⁴ See Chen et al., 2025, which finds that models frequently rely on information they do not verbalize in their stated reasoning. Where a provider additionally exposes only a post-hoc summary rather than the raw reasoning, the gap between the visible trace and the actual reasoning widens further.

⁵⁵ Korbak et al., 2025, a joint statement across several frontier labs and researchers, argues that CoT offers a valuable but fragile safety signal – valuable because models often must reason in language to complete hard tasks, fragile because that property is not guaranteed to persist. Baker et al., 2025, show that applying optimization pressure directly to the reasoning trace can teach a model to hide misbehavior in its reasoning while still misbehaving, degrading the trace as a monitoring signal.

⁵⁶ This points to interpretability research on reading a model's internal activations rather than its emitted text. That work sits at the model provider layer, outside the scope of an assessment of telemetry emitted by agent frameworks but it may be where reasoning-level monitoring proves most durable as emitted reasoning grows less reliable.

Token-level log probabilities

WHAT'S MISSING

None of the four frameworks emit token-level log probabilities as a structured telemetry field. Log probabilities are the confidence scores a model assigns to each token as it generates its output.⁵⁷ All the frameworks capture the model's verbalized confidence score, a value the model states about its own answer, but only when the model is explicitly prompted to produce one.⁵⁸ Both reflect the model's confidence, but whether a framework can emit them depends on what the underlying model exposes.

FINDINGS ACROSS FRAMEWORKS

The availability of log probabilities varies across model providers, but none of the four frameworks expose them as structured telemetry. OpenAI's API can return log probabilities when explicitly requested, but the Agents SDK does not surface them by default. Anthropic does not expose token-level log probabilities.⁵⁹ In open frameworks like LangGraph and CrewAI, log probabilities may be present in raw model responses, but they are not treated as structured fields and are not directly accessible to observability tools. For verbalized confidence, OpenAI and Claude agent SDKs, and LangGraph can each emit a score when the output is requested in a structured format, and CrewAI can as well but suppresses it by default, so an enterprise has to opt in.

WHICH SAFETY GOAL IT IMPACTS



AUDITABILITY

A confidence signal, whether a log probability or a verbalized confidence score, can help assess whether an agent's output was justified. An auditor reopening a closed AML case cannot tell from the record whether the agent's recommendation rested on a confident read of the evidence or on uncertainty it never surfaced, and a confidently worded claim and a well-grounded one look identical in the trace. As numerical values confidence signals can feed aggregation and baselining, which establishes what normal confidence looks like for a task type and flags outputs that fall below it. Neither record, log probabilities nor verbalized confidence scores, are a reliable measure of whether an agent output was correct. Log probabilities record how predictable each token was, not whether it was right, and reinforcement

RELEVANT TECHNIQUE



AGGREGATION & BASELINING

⁵⁷ Token-level log probabilities, often abbreviated as logprobs, are the natural logarithms of the per-token probabilities the model assigns to each word in its output. A high log probability means the model was confident about that token. A low one means the model was choosing among multiple near-equally-plausible alternatives. See OpenAI (b), 2026.

⁵⁸ A verbalized confidence score is a value the model is prompted to state about its own answer, as distinct from the token-level probabilities underlying its generation. See Tian et al., 2023.

⁵⁹ The Anthropic API's native response schema does not include a logprobs field, and Anthropic has not historically exposed token-level log probabilities even when developers request them. See Willows, 2024, for a developer's account of this constraint, and of the monitoring mechanisms it challenges for e.g. flagging low confidence results for review, or surfacing the model's alternative candidates. This places log probabilities outside what enterprise observability can recover for Claude-based deployments today, and makes the gap a model-provider question, not just a framework one.

learning from human feedback degrades their calibration.⁶⁰ Model verbalized scores are better calibrated than log probabilities but tend to cluster in a narrow band near full confidence, because they are produced by the same process that generated the answer.⁶¹ Since the reliability of these signals remains an open question, in the next section we recommend further research rather than standardization.

ILLUSTRATIVE EXAMPLE

A regulator or financial supervisor (such as FinCEN in the U.S. or the European Banking Authority in the EU) opens an audit of an AML investigation that closed months earlier with a recommendation for no further action. The auditor needs to determine whether the agent's conclusion reflected high confidence in a difficult case or uncertainty that was not surfaced in the report. Without any confidence signal in the trace, the audit has even less to go on, even if the signals are an imperfect guide. The agent's output appears identical in both cases.

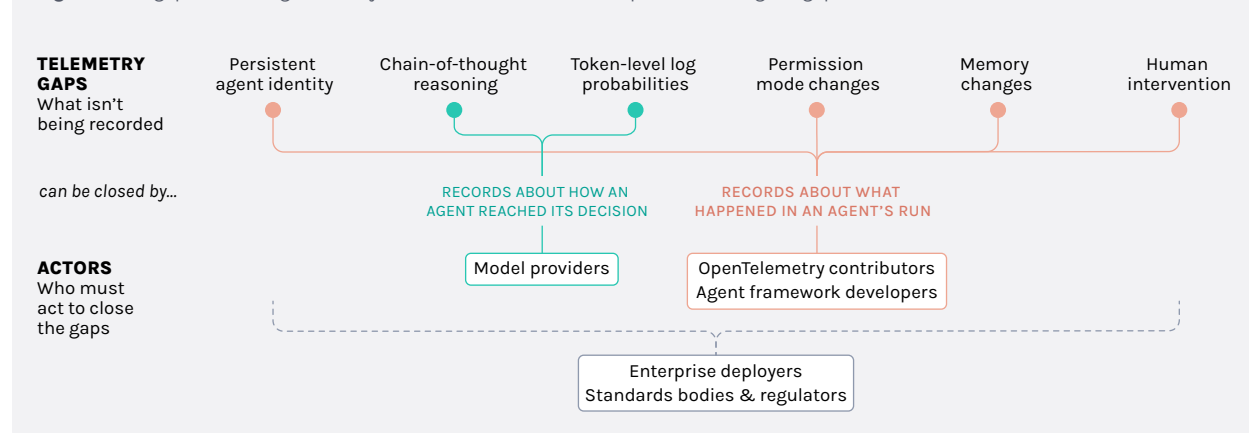
⁶⁰ Kadavath et al., 2022, find that log probabilities are well calibrated only when a model chooses among explicit answer options; calibration degrades on open-ended questions, which better resemble the multi-step evidential reasoning in our two use cases.

⁶¹ Tian et al., 2023, find that for models fine-tuned with human feedback, verbalized scores are typically better calibrated than the model's conditional probabilities. Xiong et al., 2024, find that those scores nonetheless cluster in a narrow high band, predominantly between 80 and 100 percent.

6. Recommendations

Most of the telemetry gaps this paper identifies can be closed by extending OpenTelemetry, the leading open standard for collecting and sharing telemetry data across software systems, and by framework developers adopting those extensions.⁶² The exception is telemetry about how an agent reached its output, including reasoning traces and confidence signals. Since those signals are not yet reliable enough to standardize, we recommend further research before defining telemetry standards. Separately, deriving monitoring signals from emitted telemetry requires coordination between enterprises and observability vendors. The greatest leverage sits with standards bodies and regulators, since no individual enterprise can close a field-wide gap on its own.

Figure 5. The gaps in existing telemetry standards, and the actors capable of closing the gaps.



OpenTelemetry contributors

Most of the deterministic telemetry gaps, those covering what an agent did, could be closed by adding new semantic conventions to OpenTelemetry's generative AI specification.⁶³ The most consequential additions would be a persistent, verifiable reference to an agent's enterprise identity; a session-level permission mode with an event emitted when it changes; and a set of human-intervention fields covering why a review was triggered, who responded, that reviewer's authority, and what they decided. Not every field needs to be populated by the agent framework itself. The reference to an agent identity for example can be carried by the agent framework or attached downstream, provided the trace ends up connecting an agent's actions to the enterprise identity that governs its permissions. The standard therefore needs to define the field, not require that any particular layer populate it.

RELEVANT GAPS

- Persistent agent identity
- Permission mode changes
- Memory changes
- Human intervention

⁶² OpenTelemetry is an open-source observability framework, hosted by the Cloud Native Computing Foundation, that provides standard formats and conventions for telemetry so that data from different systems can be collected and read consistently. Its conventions are developed in the open by contributors from across industry.

⁶³ The relevant venue is OpenTelemetry's semantic conventions for generative AI, a set of standards defining how telemetry from AI systems and agents should be structured. A standard of this kind defines what a field means and how it is represented; it does not by itself compel any framework to emit that field, which is why the recommendations to contributors and to framework developers are complementary. See OpenTelemetry (c).

Agent framework developers

Framework developers still need to adopt OpenTelemetry conventions and emit the corresponding telemetry. Some gaps we explore in this paper may be adoption gaps rather than gaps in the standard. For instance, OpenTelemetry's GenAI conventions now define memory change events, but three of the four frameworks we assessed do not yet emit them. Even before standardization through OTel, framework developers can emit deterministic fields already within their reach, including a reference to the agent's enterprise identity, a session level permission mode and change event, memory change events, and human intervention fields. Claude Agent SDK already emits permission mode change and tool decision events (whether a human accepted or rejected an action), showing that these fields are technically feasible. Reasoning and confidence signals are different because frameworks can only surface what the underlying model provides. Gaps in those signals therefore primarily concern model providers.

RELEVANT GAPS

- [Persistent agent identity](#)
- [Permission mode changes](#)
- [Memory changes](#)
- [Human intervention](#)

Model providers

Reasoning and confidence signals like token-level log probabilities originate in the underlying model rather than the agent framework, so these gaps cannot be closed through framework changes alone. For chain-of-thought reasoning, the open questions are whether an emitted reasoning summary faithfully reflects the model's actual reasoning, and what access arrangement would let an enterprise verify reasoning without requiring providers to expose raw traces they may withhold for competitive and safety reasons. Until these questions are resolved, a telemetry standard for these records could be premature. Progress here depends primarily on model providers working with enterprise deployers and technical researchers.

RELEVANT GAPS

- [Chain-of-thought reasoning](#)
- [Token-level log probabilities](#)

Enterprise deployers

Enterprises in regulated industries can signal collective demand for these telemetry gaps to be addressed when procuring agent frameworks or observability tools, creating buyer pressure where standards do not yet exist. That demand should extend to telemetry covering both agent actions and its decision quality. Independent evaluation of whether observability tools or frameworks emit and process key signals would give buyers a basis for comparison across vendors. Enterprises also need to configure the intent layer that observability tooling cannot supply on its own, including policies, permissions, roles, and deny lists, against which telemetry is judged, since even complete telemetry cannot be analyzed without this organizational context. Finally, telemetry only supplies the evidence and enterprises still need controls that act on it, both runtime controls such as approval gates and anomaly alerts and post-hoc review by compliance and audit functions, calibrated to the stakes and reversibility

RELEVANT GAPS

- [Persistent agent identity](#)
- [Permission mode changes](#)
- [Memory changes](#)
- [Human intervention](#)
- [Chain-of-thought reasoning](#)
- [Token-level log probabilities](#)

of an agent's actions and the affordances of the system.⁶⁴ Capturing this telemetry is however not without costs. Emitting and retaining the full set of signals recommended in our paper, from reasoning traces, memory changes to reviewer identity, increases trace volume, storage cost, and potentially latency, and it concentrates sensitive data at rest. Enterprises can tier telemetry capture to the stakes of the agent's actions: full capture for high-stakes, irreversible actions, and sampling or shorter retention for routine ones.⁶⁵ Retained telemetry should be governed by the same access policies that apply to other sensitive records.

Standards bodies and regulators

For high-risk deployments such as financial services, healthcare, and defense, regulators can specify what observability signals must be produced without prescribing the technical method. This can force convergence among agent framework developers, observability vendors and enterprises while leaving implementation open. This approach can be integrated into current monitoring requirements. In the United States, the Center for AI Standards and Innovation (CAISI) at NIST has explicitly committed to producing voluntary guidelines through its AI Agent Standards Initiative.⁶⁶ The U.S. AI Action Plan further directs NIST and CAISI to develop federal AI incident response frameworks, which depend on the kind of structured, tamper-evident telemetry this paper argues is currently missing. Internationally, Singapore's IMDA released the Model AI Governance Framework for Agentic AI in January 2026 and is actively soliciting case studies and refinements. The framework's emphasis on technical controls and human accountability creates a natural home for observability-specific guidance.⁶⁷

RELEVANT GAPS

- [Persistent agent identity](#)
- [Permission mode changes](#)
- [Memory changes](#)
- [Human intervention](#)
- [Chain-of-thought reasoning](#)
- [Token-level log probabilities](#)

⁶⁴ On calibrating the level of monitoring and control to the stakes and reversibility of an agent's actions and the affordances of the surrounding system, see Srikumar, Pratt et al., 2025.

⁶⁵ Srikumar & Nguyen, 2026

⁶⁶ CAISI's strategic pillars include facilitating industry-led standards, fostering community-led protocols, and fundamental research into agent authentication and identity. Active workstreams include the RFI on AI Agent Security and the NCCoE concept paper on software and AI agent identity and authorization. See CAISI, 2026.

⁶⁷ Singapore IMDA, Model AI Governance Framework for Agentic AI (IMDA, 2026).

7. Conclusion and Limitations

This paper offers an early, field-wide assessment of whether the telemetry available to enterprises today is enough to detect AI agent failures, in real time and after the fact. By assessing four widely used agent frameworks against six concrete monitoring goals, we identify what agent telemetry enterprises need are missing and which actors are best positioned to close them. The telemetry gaps divide along a line worth restating: the records of what an agent did are mostly a matter of standardization and adoption, while the records of how the agent arrived at a decision remains an open research problem.

This paper has two main limitations. The six monitoring goals were derived by going deep into two hypothetical use cases and generalizing the failure modes that emerged. Both cases reflect a near-term pattern the working group pointed to: enterprises in regulated industries deploying narrow-purpose agents with human reviewers in the loop, often constraining the agent to deterministic tool calls even when the underlying model is not. As agents move toward greater autonomy, monitoring needs may shift in ways these two cases did not surface, and future work should pressure-test the goals against more autonomous deployments, such as an agent that plans long horizon tasks without a human in the loop. Second, the telemetry signals we assessed are not exhaustive. We focused on those that map most directly to the six goals. Enterprises preparing to deploy agents in high-stakes settings should work backward from the failure modes they most need to detect. The gaps identified in this paper can serve as a starting point for a coordinated effort to close field-wide observability gaps in AI agents before they are deployed across industries.

The gaps identified in this paper can serve as a starting point for a coordinated effort to close field-wide observability gaps in AI agents before they are deployed across industries.

APPENDIX A

Telemetry Emissions across Four Agent Frameworks

Definitions of data fields

The definitions below focus on the data structures that are used to capture different raw telemetry signals. Although some of these data field names are derived from the name of an actual process (e.g., RAG retrieval) or a step in the AI agent lifecycle (e.g., LLM API request, MCP call), the definitions used here are not precise definitions of those processes, but rather highlight the data elements that should be included in the corresponding telemetry logs and traces. The data field names are primarily inspired by terms used in the OpenTelemetry standard and the Claude Agent software development kit (SDK).

Two commonly used terms in reference to the AI agent telemetry process are “trace” and “span.” A trace is a structured, chronological record of every single action, decision, and sub-process an AI agent performs to fulfill a user prompt, while a span is the fundamental unit of a trace in an agent process — it represents a single, timed operation with a defined start and end.

CATEGORY	DATA FIELD	DEFINITION
Agent identity	Agent identity	A unique, persistent identifier that distinguishes one agent from another. Ideally, the agent ID should enable authentication to securely identify the principal (human or system) that initiates the agent action, and authorization to scope the agent's exact permissions and facilitate the agent's interactions with external services. It should also help identify the runtime instance an agent is executing on, so that handoffs can be targeted and state resumed.
Session and prompt identifiers	Session ID	A unique identifier for a single agent session used to correlate all events produced within it.
	Prompt ID	A unique identifier correlating a user prompt to all subsequent events.
Structured inputs and outputs	User prompt	The full prompt text content and its attributes such as its length.
	Structured input-output log	A data structure that contains structured chat history sent to the model and structured model responses, for example using a JSON schema or key-value array.
Tool calls and MCP calls	Tool result	A data structure that captures the details of a tool invocation. It contains the name and unique identifier of the tool called, tool input, whether it succeeded or failed, execution time, errors and decision types, sizes of tool input and tool results, and tool parameters.
	MCP call	A data structure that contains MCP connection details including the connection status, the transport protocol, server configuration scope, connection attempt duration, configured server name, and an error code and full error message when the connection fails.
RAG retrieval	RAG retrieval	A data structure that contains the Retrieval Augmented Generation (RAG) query text, the provider's name of the vendor or engine serving the retrieval, a data source ID uniquely identifying the specific store or knowledge base being queried, the conversation ID, the retrieval documents and any retrieval errors.

LLM API interaction	LLM API request	A data structure that contains request details such as a unique API request ID, an identifier of the model called, estimated cost, request duration, number of input and output tokens, query source subsystem and effort level.
	LLM API error	A data structure that contains error details when an LLM API request returns an error. The details include the LLM API request details and the error message.
	LLM API response body	A data structure that contains an API's response parameters and details like the body length, the identifier of model called and subsystems that issued the request.
Chain-of-thought reasoning	Chain-of-thought reasoning	A structured, step-by-step breakdown of the problem-solving process before reaching the final answer that is emitted before the agent's final output.
Token-level log probabilities	Token-level log probabilities	Also called <i>logprobs</i> . Natural logarithms of per-token probabilities over the model's vocabulary distribution for the generated output.
Model confidence scores	Model confidence scores	A verbalized confidence score (e.g., a number between 0 – 1 or a high/medium/low label) based on a self-assessment or reflection generated when a model is prompted to emit its confidence estimate alongside the answer. The prompt for confidence estimation may be generated by the agent framework or harness or may be part of the user's initial prompt.
Permission mode changes	Permission mode changes	A data field, currently only implemented by Claude Agent SDK, that shows modifications in the agent's degree of autonomy which determines how often an agent pauses for user approval before running actions.
Memory changes	Memory changes	Any change that can be detected in an agent's memory, including its short-term working context (conversation history) and long-term persistent storage of knowledge and context.
Inter-agent messages and handoffs	Inter-agent message content	Actual content of the communication between agents.
	Orchestration and handoffs metadata	Records about control flow across agents such as IDs of what agent handed off what task to what other agent.
Human intervention	HitL (Human in the loop) trigger reason	A description of the condition that caused the pause for a human to intervene within an agent process.
	HitL reviewer identity	The structured identifier data about the human who took an action during agent execution, such as a user ID or employee identifier.
	Tool decision	A data structure capturing a human's response to a tool use request. It contains the name of the tool called, a unique identifier for the tool invocation, the "accept" or "reject" decision, and the source of the invocation.
	Review escalation mode	A record of whether the human review is discretionary (the agent or a heuristic is able to choose whether to escalate) or mandated (the process design guarantees a human review).
	Review request rationale	The agent's recommendation plus rationale based on the underlying evidence used by the agent.

Grading for telemetry generation assessment

When assessing the raw telemetry generation capabilities enabled by different AI agent SDKs and frameworks, we used a three-level grading: “Emits,” “Partially Emits,” and “Does Not Emit,” described below.

STATUS	SCORE
<div><div></div> Emits</div> <div>If the SDK/framework’s telemetry is able to fully capture and record the data item. For example, all four SDKs/frameworks assessed fully capture and record structured user inputs.</div>	2
<div><div></div> Partially Emits</div> <div>If an incomplete or insufficient version of the data field is emitted by the SDK/framework’s telemetry.</div>	1
<div><div></div> Does Not Emit</div> <div>If the SDK/framework’s telemetry does not capture the data item.</div>	0

“Emits” is assigned a score of 2, “Partially Emits” is assigned a score of 1, and “Does Not Emit” is assigned a score of 0. Related data fields are grouped together into categories as shown in Table 1. Using these grades, we compute, for each SDK/framework, the average score for all the data fields in each category. For example, the OpenAI Agents SDK has a score of 1 for the “Session ID” data field and a score of 0 for the “Prompt ID” data field. So the aggregated “Session and prompt identifiers” category in the OpenAI Agents SDK column gets an average score of 0.5.

In the associated color coding, ● red indicates weak telemetry generation capabilities in the framework or SDK being assessed, ● yellow indicates medium capabilities, and ● green indicates strong capabilities. Even though category averages are continuous, we round each to the nearest of the three color bands for ease of gap visibility in the aggregate table in the main paper.

DATA FIELD	OPENAI AGENTS SDK	CLAUDE AGENT SDK	LANGGRAPH	CREWAI
AGENT IDENTITY				
Agent identity	<ul style="list-style-type: none"> Agents have a free-text name attribute on agent span data, but there is no persistent identity. The SDK does not provision an identity credential for each agent instance. Authentication data about the initiating principal is not reflected in tracing and authorization for external services is not tied to any agent identifier. 	<ul style="list-style-type: none"> Each LLM request span carries agent ID and parent agent ID attributes. These are run-scoped instance identifiers, not persistent agent identities. There's no per-agent identity for authorization scoping against access to services; permissioning is handled by the tool-permission system using allow and deny lists. The SDK generates a unique user ID that identifies the human or organization principal, but it's not tied to the agent ID. 	<ul style="list-style-type: none"> An agent name attribute exists in the OpenInference convention but it is not a persistent unique identity. LangGraph does not provision per-agent unique identity credentials. LangGraph Platform's authentication layer authenticates and authorizes the human end user, not the agent. 	<ul style="list-style-type: none"> Agent spans expose an agent ID and an agent role but these are framework-internal unique IDs, not attested digital identities. Each agent automatically receives a unique fingerprint when created, which cannot be manually overridden and it is explicitly persistent and remains unchanged throughout an agent's lifecycle. The fingerprint appears in exported telemetry as a distinct span attribute. But this agent ID fingerprint does not serve authentication or authorization purposes.
SESSION AND PROMPT IDENTIFIERS				
Session ID	<ul style="list-style-type: none"> Emits a group ID on the trace that links related runs in a conversation, and a unique trace ID per workflow. But there is no dedicated session ID attribute on every span. The developer must pass a group ID themselves to correlate sessions. 	<ul style="list-style-type: none"> Every span, metric, and event carries a session ID as a standard attribute using OpenTelemetry's instrumentation. It is also stamped on every trace span automatically. 	<ul style="list-style-type: none"> LangGraph uses LangSmith as its native tracing layer and LangSmith is seamlessly attached whenever a developer builds an agent or multi-agent workflow using LangGraph. LangSmith supports threading runs by session ID, thread ID or conversation ID and groups traces under that, but the developer needs to supply it. There is no automatic, framework-level session ID stamped on every event. 	<ul style="list-style-type: none"> The CrewAI framework implements built-in instrumentation emitting telemetry using OpenTelemetry semantic conventions. CrewAI built-in tracing groups runs under a 'trace batch' tied to a process kickoff. Session IDs are not indicated as being attached to events in the public documentation.
Prompt ID	<ul style="list-style-type: none"> The SDK ties events together via trace ID and parent span ID, not a dedicated unique ID that correlates one user prompt to all downstream events. Spans correlate through the span hierarchy only. 	<ul style="list-style-type: none"> Every event carries a prompt ID — a universally unique ID that links the user prompt event, all API requests, tool results, and tool decision events produced while processing that prompt. 	<ul style="list-style-type: none"> No native prompt ID attribute generated. LangSmith uses run trace IDs and parent IDs to correlate prompts and subsequent events. A developer may attach a custom metadata key for the same purpose, but it is not emitted automatically by LangGraph itself. 	<ul style="list-style-type: none"> CrewAI does not emit a unique ID that uniquely ties one user prompt to every downstream event.

STRUCTURED INPUTS AND OUTPUTS

User prompt	<ul style="list-style-type: none"> ● The full input is captured by various spans including the agent span, the generation and the response spans. 	<ul style="list-style-type: none"> ● Claude Agent SDK has a dedicated user prompt event that carries the prompt details including the length. The full prompt text is captured when the corresponding underlying OpenTelemetry configuration is set to allow it. The interaction span also carries the user prompt. 	<ul style="list-style-type: none"> ● LangSmith captures inputs and outputs for every run by default, including the human message that started the graph. It's visible as the run input on the kick-off root run. 	<ul style="list-style-type: none"> ● CrewAI's LLM call event and the LLM trace spans capture the prompt sent to the model. CrewAI Agent Management Platform (AMP) shows all prompts sent to the LLM.
Structured input-output log	<ul style="list-style-type: none"> ● The SDK's generation span has data for input and output as a sequence of messages, and the response span has data for the full response object. System instructions sit inside the model configuration input. 	<ul style="list-style-type: none"> ● Claude Agent SDK's API request body and API response body events emit JSON-serialized messages, including the full conversation, tools, and system prompt. 	<ul style="list-style-type: none"> ● LangSmith captures structured messages on every run – inputs (system, human, AI, tool messages) and outputs are stored as typed message objects. 	<ul style="list-style-type: none"> ● LLM call events and OpenInference LLM spans capture the full messages array including system instructions, user messages, assistant responses and tool calls.

TOOL CALLS AND MCP CALLS

Tool result	<ul style="list-style-type: none"> ● The SDK's function span data includes the tool's name, input and output. Other tool use aspects such as errors, decision type and execution time are not built in but some can be inferred from the span's start and end timestamps. Errors come through span error state. 	<ul style="list-style-type: none"> ● Claude Agent SDK has a tool result event that includes tool name, tool use ID, success, duration, error type, decision type, decision source, MCP server, scope and tool parameters. The tool input is also included when the corresponding underlying OpenTelemetry configuration is set to allow it. 	<ul style="list-style-type: none"> ● Tool runs record the tool's name, inputs, outputs, latency, status, and errors. But tool decision type, and explicit success/fail booleans are not standard fields captured. When a tool call fails an error status is returned. 	<ul style="list-style-type: none"> ● Every tool span captures the tool's name, parameters and result and every call results in log events for when tool use starts, when tool use ends and when there's a tool use error. Decision types and explicit success booleans are not documented as being captured.
MCP call	<ul style="list-style-type: none"> ● The SDK uses MCP list tools span to capture the MCP server name and the associated list of tools. But other details like connection or disconnection status, the transport type, scope, duration, and error code are not captured. 	<ul style="list-style-type: none"> ● Claude Agent SDK uses an MCP server connection event to capture the connection status, transport type, server scope, duration, error code, and server name. 	<ul style="list-style-type: none"> ● LangGraph itself does not run MCP servers natively in the way agent SDKs from Claude and OpenAI do. Thus there's no documented MCP connection telemetry event. MCP connection details are instead wrapped inside tool calls, and attributes such as input parameters and response payloads are captured on tool spans. 	<ul style="list-style-type: none"> ● CrewAI logs a number of events (such as at MCP connection start, at completion, and when the connection fails) with MCP server connection details including server name, URL, transport type, timeout, duration, error message, and error type.

RAG RETRIEVAL				
RAG retrieval	<ul style="list-style-type: none"> OpenAI SDK performs RAG using a file search tool. The file-search query, the returned retrieval documents, and the source IDs are captured in a response object which is stored in memory but are not emitted by the default exporter as discrete fields. 	<ul style="list-style-type: none"> Claude Agent SDK implements RAG retrieval as a tool call, and tool telemetry captures the retrieval signals. Using the OpenTelemetry (OTEL) standard, it's able to export these fields. 	<ul style="list-style-type: none"> LangSmith/ LangChain has a built-in retriever run type, and OpenInference's LangChain instrumentation emits retrieval documents and retrieval query text on retriever spans. 	<ul style="list-style-type: none"> CrewAI has dedicated knowledge-retrieval events that emit some of the retrieval details such as the query text, the retrieval documents and retrieval errors. The source ID and provider is not emitted.
LLM API INTERACTION				
LLM API request	<ul style="list-style-type: none"> The key LLM API request details such as input, output, model called, and model configuration are captured by the generation span. But other attributes such as estimated cost, query source subsystem, and effort level are not captured. 	<ul style="list-style-type: none"> API request details such as the model called, cost, duration, number of input and output tokens, request id, speed, query source and effort are captured through an API request event. 	<ul style="list-style-type: none"> LLM runs record model, latency, tokens, and inputs and outputs. Cost is computed downstream from the model pricing map; effort level and query source subsystem are not native LangGraph attributes. 	<ul style="list-style-type: none"> LLM call events capture model, prompt, response, latency and tokens; cost is shown via the AMP dashboard or integration of the OpenLIT open source tool for cost tracking. Effort level and a dedicated query source subsystem are not captured.
LLM API error	<ul style="list-style-type: none"> The generation span ends with error status when the model call fails, and the exception is recorded. But other error details like number of attempts, query source, and effort level are not captured. 	<ul style="list-style-type: none"> API request error details such as the model called, error, status code, duration, number of attempts, request ID, speed, query source and effort are logged in an API error event. 	<ul style="list-style-type: none"> LLM runs record an error field and status when the call fails. The number of attempts, query source subsystem, and effort level are not captured. 	<ul style="list-style-type: none"> When there's an API call error an LLM call failure event is fired and it logs the error message string and error type. OpenTelemetry spans record exception status. But the number of attempts, query source subsystem and effort level are not recorded.
LLM API response body	<ul style="list-style-type: none"> The generation span's output and the response span capture the model's response. The response's body length and calling subsystem are not captured. 	<ul style="list-style-type: none"> The API response body event carries the JSON-serialized messages API response, body length, model, query source and request ID. 	<ul style="list-style-type: none"> LLM run outputs are stored verbatim, including content blocks and finish reason. Request's body length and calling subsystem are not captured. 	<ul style="list-style-type: none"> CrewAI uses an LLM call completed event and OpenInference LLM spans to capture the API response. Response body length and calling subsystem are not captured.
CHAIN-OF-THOUGHT REASONING				
Chain-of-thought reasoning	<ul style="list-style-type: none"> OpenAI Agents SDK's reasoning item is constrained by OpenAI's platform policy, which the SDK inherits and cannot override. OpenAI does not expose the raw reasoning tokens emitted by the model and the SDK only emits what the model provider returns. 	<ul style="list-style-type: none"> The SDK does not expose raw reasoning. When Claude's extended thinking is turned on, the SDK emits dedicated reasoning text in an explicitly separated content block called a "thinking block." But the reasoning text that is emitted to the user before the final answer is not the model thinking live before taking actions – it's a post hoc summarized description of an already completed thinking process and is done by a different model. 	<ul style="list-style-type: none"> LangGraph emits only what the underlying model returns and does not manufacture raw CoT where the API withholds it. Reasoning output must be enabled on the model, and if the provider only returns a summary that's what LangGraph emits. 	<ul style="list-style-type: none"> CrewAI has a planning feature orchestrated by the framework that, when turned on, allows agents to reflect on a task and create a plan before execution. This plan is structured and is programmatically retrievable as an agent reasoning event emitted when an agent finishes its reasoning process. When a thinking chunk is received from an underlying model, CrewAI logs it.

TOKEN-LEVEL LOG PROBABILITIES				
Token-level log probabilities	<ul style="list-style-type: none"> ● Token-level log probabilities are a model-API feature that can be requested via model configuration, but the SDK's tracing does not have a dedicated attribute for them. Whatever the model returns can sit inside the response span data, but the SDK does not capture this distinctly. 	<ul style="list-style-type: none"> ● Anthropic's messages API does not return per-token log probabilities and the Claude Agent SDK telemetry does not capture it. 	<ul style="list-style-type: none"> ● LangSmith telemetry does not surface per-token log probabilities as a field. But they may appear inside the raw model output when the provider returns them. 	<ul style="list-style-type: none"> ● CrewAI does not define a log probabilities attribute. But the log probabilities may sit inside the raw response when the underlying provider returns them.
MODEL CONFIDENCE SCORES				
Model confidence scores	<ul style="list-style-type: none"> ● OpenAI Agents SDK emits a verbalized confidence value between 0 and 1 alongside the model's output when there's a prompt to generate the output in a structured format instead of a regular plain text response. 	<ul style="list-style-type: none"> ● Claude Agent SDK generates a verbalized confidence value accompanying an output as long as the prompt specifies that it's required as part of the structured output schema. 	<ul style="list-style-type: none"> ● LangGraph, through its structured response capabilities, enables the generation of a numeric verbalized confidence rating between 0 – 5 in its trace. 	<ul style="list-style-type: none"> ● CrewAI supports the generation of a verbalized confidence score through its structured output capability. But CrewAI deliberately suppresses outputs by default, so one would need to opt in to have the telemetry export it.
PERMISSION MODE CHANGES				
Permission mode changes	<ul style="list-style-type: none"> ● OpenAI Agents SDK has no concept of global session-wide permission modes and thus no telemetry event for switching between them. Autonomy is configured per tool via a flag for whether approval is needed. 	<ul style="list-style-type: none"> ● Claude Agent SDK uses a permission mode changed event with the previous mode, the current mode, and the trigger for the change. 	<ul style="list-style-type: none"> ● LangGraph has no permission mode concept and thus no telemetry event for mode changes. LangGraph defines an autonomy level using the interrupt policy for human intervention but it is configured per tool and not session-wide. The policy is configured when the agent is created, is static and cannot be modified at runtime. 	<ul style="list-style-type: none"> ● CrewAI has no permission mode concept and thus no telemetry event for mode changes. CrewAI has no session-wide, mutable permission mode. It has a per-task flag that is defined statically and indicates whether human input is needed. To integrate human input into agent execution, the human input flag is set in the task definition, and when enabled, the agent prompts the user for input before delivering the final answer.
MEMORY CHANGES				
Memory changes	<ul style="list-style-type: none"> ● OpenAI Agents SDK does not emit any data associated with memory operations. There's no span associated with memory. 	<ul style="list-style-type: none"> ● Claude Agent SDK captures one memory-change operation using a "compaction" event when a conversation summarization completes, indicating whether it was automatically or manually triggered, whether the compaction succeeded, and an approximate token count before and after completion. It emits no other data associated with memory operations. 	<ul style="list-style-type: none"> ● LangGraph does not have direct capture or emission of any memory operations. One would need to instrument memory operations as tool calls. 	<ul style="list-style-type: none"> ● Though CrewAI emits memory events such as start and completion of memory queries and saves, it does not capture the actual contents of these memory operations.

INTER-AGENT MESSAGES AND HANDOFFS				
Inter-agent message content	<ul style="list-style-type: none"> ● The SDK has handoff span data that captures the from-agent and the to-agent. The actual message passed between agents is captured on the destination agent's input span. 	<ul style="list-style-type: none"> ● Subagents spawned through the task tool record their own LLM request and tool spans nested under the parent's tool span. The text passed to the subagent is captured on its first user prompt event or interaction span, but it is not a dedicated inter-agent-message field. 	<ul style="list-style-type: none"> ● Handoffs in LangGraph are implemented as tool calls that return a command. The tool call arguments which include the message and the updated state are captured on the tool run and on the next agent node's input. 	<ul style="list-style-type: none"> ● CrewAI uses the agent-to-agent (A2A) protocol message sent event and the A2A response received event to capture the message content, turn number, context ID, message ID, and final-response flag for agent-to-agent communication.
Orchestration and handoffs metadata	<ul style="list-style-type: none"> ● The SDK's trace's span hierarchy records who handed off what to whom. Specifically, agent handoffs (from and to) are captured in the handoff span data. Additionally agent span data also lists handoffs and tools. 	<ul style="list-style-type: none"> ● Subagent activity nests under the parent tool span and LLM request spans carry agent ID and parent agent ID capturing the parent-child relationship. There is no dedicated 'handoff' event between named agents. 	<ul style="list-style-type: none"> ● Each graph node is a run, and a command handoff between agents produces nested runs that show who routed control where, with the routing state visible on the run. Graph node ID, graph node name and graph node parent ID attributes formalize this. 	<ul style="list-style-type: none"> ● CrewAI uses the A2A delegation start and completion events to record the endpoint, task description, agent ID, context ID, multi-turn flag, turn number, and result. It also uses the A2A parallel delegation event to record fanout across multiple agents.
HUMAN INTERVENTION				
HitL (Human in the loop) trigger reason	<ul style="list-style-type: none"> ● The HitL mechanism pauses when a tool's approval rule fires but the interruption reason is not captured. The triggers are caused by a simple boolean (true/false) flag that indicates whether human approval is needed, but no reason is provided. 	<ul style="list-style-type: none"> ● Claude Agent SDK generates a structured reason for the HitL trigger. It generates the trigger reason, attaches it to a HitL-specific event, and emits it as telemetry. It has a dedicated permission request event for HitL notification. Before tools are called, an event containing a permission decision reason is logged. 	<ul style="list-style-type: none"> ● LangGraph provides an interrupt function that pauses graph execution and surfaces a value to the client. Though the trigger context is captured, the trigger reason is developer-defined, so there's no standardized reason mechanism and no structured reason field. 	<ul style="list-style-type: none"> ● CrewAI provides a flag for whether human input is needed which generates a pause for review, but the pause is hard-wired to the task-completion checkpoint and no reason for the pause is captured.
HitL reviewer identity	<ul style="list-style-type: none"> ● Does not emit a reviewer identity. Though HitL interruptions are enabled, there's no parameter identifying who the approver is. 	<ul style="list-style-type: none"> ● There's no field for reviewer identity. 	<ul style="list-style-type: none"> ● LangGraph includes no reviewer identity field. Requests are not scoped to individual users. 	<ul style="list-style-type: none"> ● CrewAI does not emit a reviewer identity.
Tool decision	<ul style="list-style-type: none"> ● The SDK's function tools execute without a built-in approval or decision telemetry event. Human-in-the-loop tool approval is supported, but the accept or reject decision and source are not exposed as a tracing event. 	<ul style="list-style-type: none"> ● The tool name, tool use ID, accept or reject decision, and source of the tool call are captured in a tool decision event. 	<ul style="list-style-type: none"> ● LangGraph supports human-in-the-loop approval for tool use, but there is no telemetry captured for the tool use id, accept or reject decision. 	<ul style="list-style-type: none"> ● CrewAI has no documented event for tool use accept or reject decision. It emits events for starting and finishing tool use and associated errors.
Review escalation mode	<ul style="list-style-type: none"> ● OpenAI Agents SDK does not record whether a human review request was discretionary or mandated. 	<ul style="list-style-type: none"> ● Claude Agent SDK does not record whether a human review was discretionary or mandated. 	<ul style="list-style-type: none"> ● LangGraph does not record whether a human review was discretionary or mandated. 	<ul style="list-style-type: none"> ● CrewAI does not record whether a human review was discretionary or mandated.

Review request rationale	<p>● OpenAI Agents SDK provides a runtime review mechanism but has no telemetry to capture the review content.</p>	<p>● For tool use, Claude Agent SDK presents an approve/reject decision to a reviewer and captures the decision, but no preferred recommendation or rationale is presented or captured.</p>	<p>● LangGraph captures a 'value' field showing a detailed reviewer payload. But it's up to the deployer to decide what goes in the payload, so the value can contain an agent's recommendation and rationale.</p>	<p>● CrewAI provides a review mechanism through a console prompt but has no telemetry to capture the review content.</p>
---------------------------------	--	---	--	--

Technical documentation consulted for each framework

OpenAI Agents SDK

OpenAI (a), "Tracing" Agents SDK — Tracing documentation <https://openai.github.io/openai-agents-python/tracing/>. Accessed May 13, 2026.

OpenAI (b), "Span data" OpenAI Agents SDK — Span data API reference https://openai.github.io/openai-agents-python/ref/tracing/span_data/. Accessed May 13, 2026.

OpenAI (c), "Creating traces/spans" OpenAI Agents SDK — API reference <https://openai.github.io/openai-agents-python/ref/tracing/create/>. Accessed May 13, 2026.

OpenAI (d), OpenAI Agents SDK — "Guardrails" <https://openai.github.io/openai-agents-python/guardrails/>. Accessed May 13, 2026.

OpenAI (e), "OpenAI Agents SDK" <https://openai.github.io/openai-agents-python/>. Accessed May 13, 2026.

Claude Agent SDK

Anthropic (a), Claude Agent SDK — Observability with OpenTelemetry <https://code.claude.com/docs/en/agent-sdk/observability>. Accessed May 18, 2026.

Anthropic (b), Claude Code Monitoring reference <https://code.claude.com/docs/en/monitoring-usage>. Accessed May 18, 2026.
Claude Agent SDK emits the same data.

Anthropic (c), Claude Agent SDK — Hooks documentation <https://code.claude.com/docs/en/agent-sdk/hooks>. Accessed May 18, 2026.

Anthropic (d), Claude Agent SDK — Permissions <https://code.claude.com/docs/en/agent-sdk/permissions>. Accessed May 18, 2026.

Anthropic (e), Claude Agent SDK — MCP integration <https://code.claude.com/docs/en/agent-sdk/mcp>. Accessed May 18, 2026.

LangGraph

LangChain (a), "Trace LangGraph applications" <https://docs.langchain.com/langsmith/trace-with-langgraph>. Accessed May 18, 2026.

LangChain (b), March 26, 2025. "Introducing End-to-End OpenTelemetry Support in LangSmith" <https://blog.langchain.com/end-to-end-opentelemetry-langsmith/>. Accessed May 18, 2026.

LangChain (c), "Know what your agents are really doing" <https://www.langchain.com/langsmith/observability>. Accessed May 18, 2026.

Arize AI (a), "Semantic Conventions" OpenInference — Semantic Conventions used by LangChain/LangGraph instrumentation https://github.com/Arize-ai/openinference/blob/main/spec/semantic_conventions.md. Accessed May 18, 2026.

LangChain (d), "Handoffs" LangGraph — Multi-agent handoffs documentation <https://docs.langchain.com/oss/python/langchain/multi-agent/handoffs>. Accessed May 18, 2026.

CrewAI

CrewAI (a), CrewAI — Built-in Tracing documentation <https://docs.crewai.com/en/observability/tracing>. Accessed May 18, 2026.

CrewAI (b), CrewAI — Event Listeners (CrewAIEventsBus) documentation <https://docs.crewai.com/en/concepts/event-listener>. Accessed May 18, 2026.

CrewAI (c), CrewAI — OpenLIT integration (OTel one-line setup) <https://docs.crewai.com/en/observability/openlit>. Accessed May 18, 2026.

CrewAI (d), CrewAI — Observability overview / platform integrations <https://docs.crewai.com/en/observability/overview>. Accessed May 18, 2026.

Arize AI (b), OpenInference — CrewAI instrumentation <https://github.com/Arize-ai/openinference>. Accessed May 18, 2026.

APPENDIX B

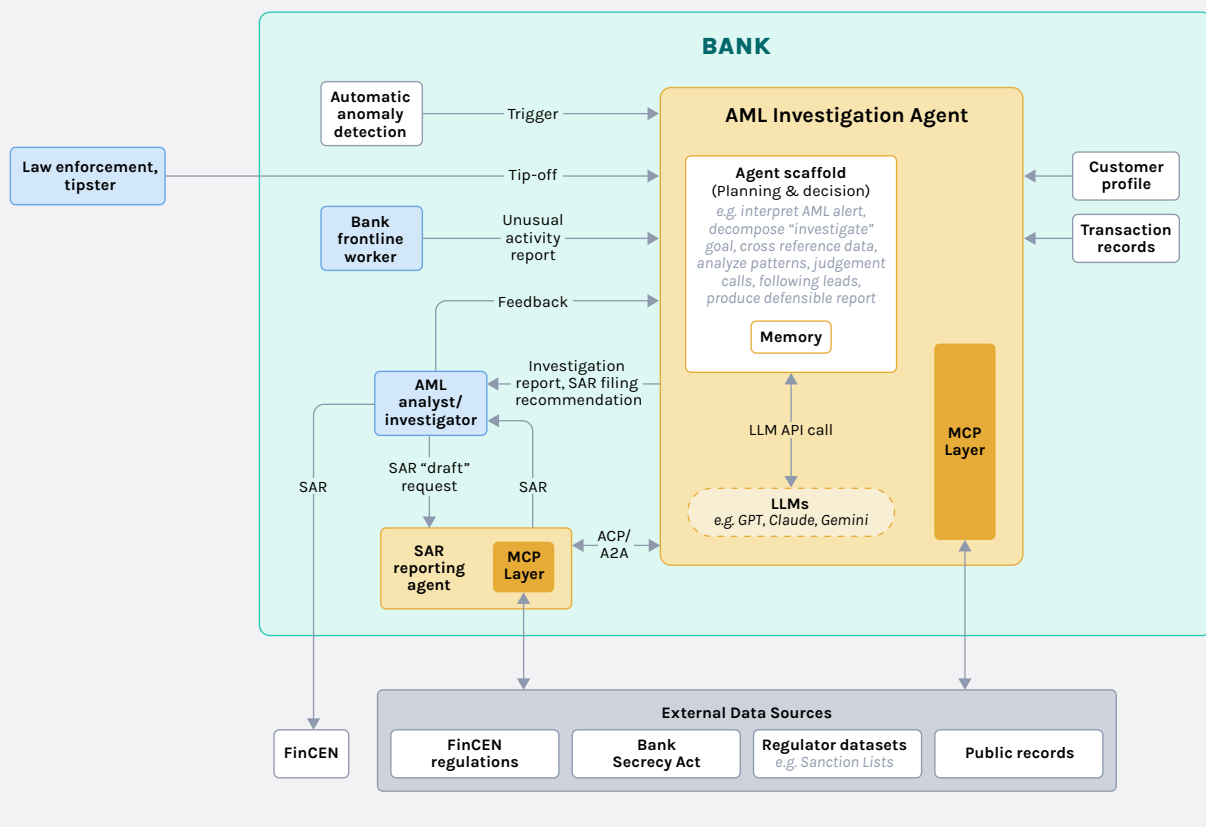
Illustrative use cases

The following diagrams provide more details about the illustrative use cases introduced in [Section 3: Monitoring Goals](#). They include more granular depictions of the bank's workflow and external data sources; the agentic processes are unchanged.

ILLUSTRATIVE USE CASE 1

AML Investigation Agent

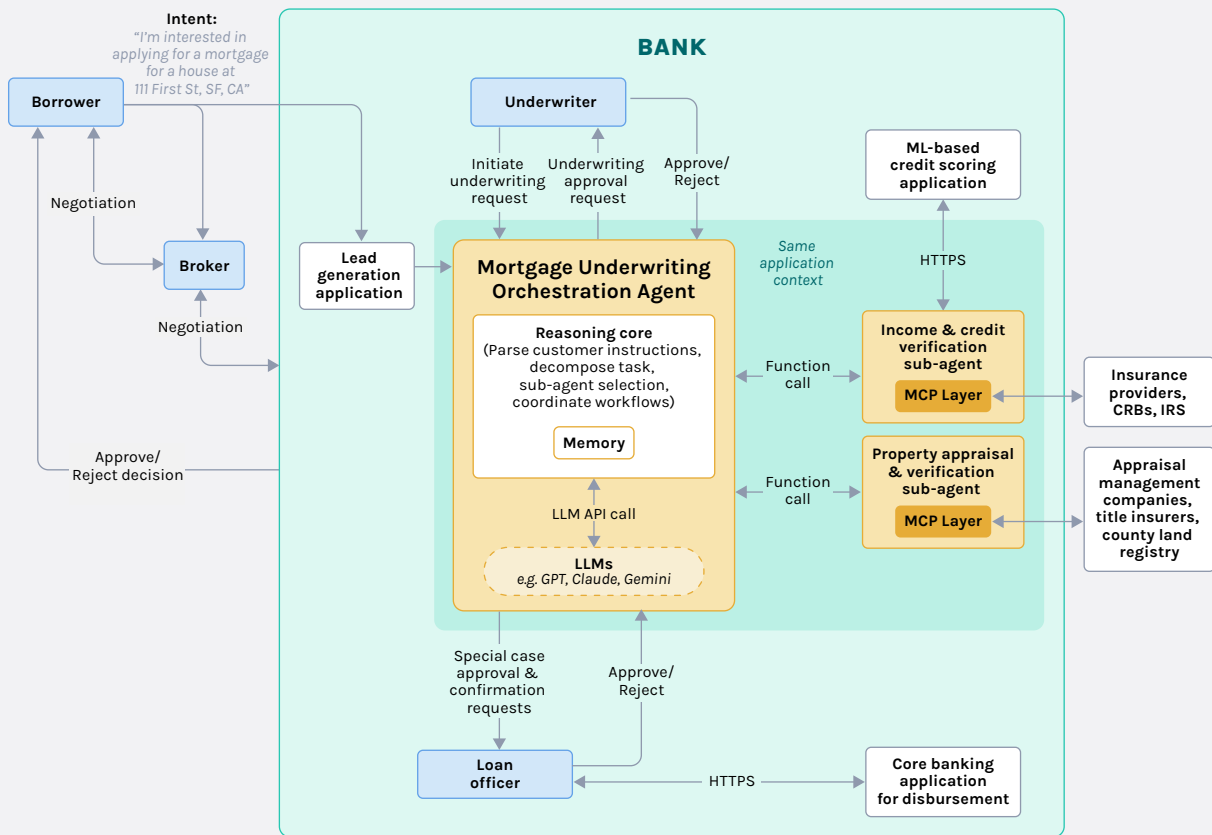
1. A large **bank** deploys an **agent** to triage alerts generated by external reports or its internal anti-money laundering monitoring system.
2. The **agent** reviews the alert, pulls together the customer's account history, news mentions, prior case narratives, and checks against sanctions lists, and drafts a recommendation about whether to escalate to a **human investigator** or close the alert.
3. **Human analysts** review the recommendation if escalated and decide whether to close it or open a formal case. Cases that are opened as a formal case may ultimately lead to the bank filing a Suspicious Activity Report (SAR) with regulators. Closed alerts may later be audited to verify whether the decision was sound.



ILLUSTRATIVE USE CASE 2

Mortgage Underwriting Agent

1. A **lender** deploys an **agent** to process residential mortgage applications.
2. The agent receives the application package, calls **sub-agents** to extract data from pay stubs, employment letters, and bank statements, runs credit and debt-to-income checks through deterministic tools.
3. The **agent** drafts a recommendation about whether to approve, deny, or refer the application for further review.
4. A **human loan officer** reviews the recommendation and makes the final decision. Decisions to approve or deny applications may later be examined by outside auditors for compliance with fair-lending laws.



Bibliography

- AI Incident Database. [Filtered search results for 27 agent-related incidents]. Responsible AI Collaborative. Accessed August 2026. <https://shorturl.at/3vVog>
- AI Incident Database. "Incident 1510." Responsible AI Collaborative. Accessed June 2026. <https://incidentdatabase.ai/cite/1510>.
- AI Incident Database. "Incident 1424." Responsible AI Collaborative. Accessed June 2026. <https://incidentdatabase.ai/cite/1424>.
- AI Incident Database. "Incident 1152." Responsible AI Collaborative. Accessed June 2026. <https://incidentdatabase.ai/cite/1152>.
- AI Incident Database. "Incident 1542." Responsible AI Collaborative. Accessed June 2026. <https://incidentdatabase.ai/cite/1542>.
- Al-Tawaha, Ahmad, Shangding Gu, Peizhi Niu, Ruoxi Jia, and Ming Jin. "Remembering More, Risking More: Longitudinal Safety Risks in Memory-Equipped LLM Agents." arXiv preprint arXiv:2605.17830, May 2026. <https://arxiv.org/abs/2605.17830>.
- Amazon Web Services. "Amazon Bedrock Guardrails." AWS Documentation. Accessed June 2026. <https://aws.amazon.com/bedrock/guardrails/>.
- Anthropic (a). "Auto Mode." Claude Code Documentation. Accessed June 2026. <https://code.claude.com/docs/en/auto-mode-config>.
- Anthropic (b). "Permission Modes." Claude Code Documentation. Accessed June 2026. <https://code.claude.com/docs/en/permission-modes>.
- Anthropic (c). "Permissions." Claude Agent SDK Documentation. Accessed June 2026. <https://platform.claude.com/docs/en/agent-sdk/permissions>.
- Anthropic (d). "Tool Decision Event." Claude Code Documentation. Accessed June 2026. <https://code.claude.com/docs/en/monitoring-usage#tool-decision-event>.
- Anthropic (e). "Zero Trust for AI Agents." May 27, 2026. <https://claude.com/blog/zero-trust-for-ai-agents>.
- Arize. "Retrieval (RAG) Relevance." Phoenix Documentation. Accessed June 2026. <https://arize.com/docs/phoenix/evaluation/pre-built-metrics/retrieval-rag-relevance>.
- Baker, Bowen, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y. Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. "Monitoring Reasoning Models for Misbehavior and the Risks of Promoting Obfuscation." arXiv preprint arXiv:2503.11926, March 2025. <https://arxiv.org/abs/2503.11926>.
- Bank Policy Institute and American Bankers Association. Comment on NIST Request for Information Regarding Security Considerations for Artificial Intelligence Agents, Docket No. NIST-2025-0035. Submitted by BITS, the technology policy division of the Bank Policy Institute, March 9, 2026. <https://bpi.com/wp-content/uploads/2026/03/BPI-ABA-Comment-NIST-3.9.2026.pdf>.
- Bengio, Yoshua, et al. International Scientific Report on the Safety of Advanced AI: Interim Report. Department for Science, Innovation and Technology, May 2024. <https://www.gov.uk/government/publications/international-scientific-report-on-the-safety-of-advanced-ai>
- Boston Consulting Group. "Making AI Agents Safe for the World." 2025. <https://www.bcg.com/publications/2025/making-ai-agents-safe-for-world>.
- CAISI (NIST Center for AI Standards and Innovation). "AI Agent Standards Initiative." National Institute of Standards and Technology, February 2026. <https://www.nist.gov/artificial-intelligence/ai-agent-standards-initiative>.
- California Senate Bill 53. Transparency in Frontier Artificial Intelligence Act. Cal. Stats. 2025, ch. 271. https://leginfo.ca.gov/faces/billTextClient.xhtml?bill_id=202520260SB53.
- Chaikin, Artem and Sahib, Shivan Kaul. Agentic Browser Security: Indirect Prompt Injection in Perplexity Comet. Brave, August 2025. <https://brave.com/blog/comet-prompt-injection/>.
- Chan, Alan, Carson Ezell, Max Kaufmann, Kevin Wei, Lewis Hammond, Herbie Bradley, Emma Bluemke, et al. "Visibility into AI Agents." arXiv preprint arXiv:2401.13138, January 2024. <https://arxiv.org/abs/2401.13138>.
- Chen, Yanda, Joe Benton, Ansh Radhakrishnan, Jonathan Uesato, Carson Denison, John Schulman, Arushi Somani, et al. "Reasoning Models Don't Always Say What They Think." arXiv preprint arXiv:2505.05410, 2025. <https://arxiv.org/abs/2505.05410>.
- Dalugoda, Asiri. "HDP: A Lightweight Cryptographic Protocol for Human Delegation Provenance in Agentic AI Systems." arXiv preprint arXiv:2604.04522, April 2026. <https://arxiv.org/abs/2604.04522>.
- European Commission. General-Purpose AI Code of Practice. 2025. <https://digital-strategy.ec.europa.eu/en/policies/contents-code-gpai>.
- EU AI Act. Regulation (EU) 2024/1689 of the European Parliament and of the Council Laying Down Harmonised Rules on Artificial Intelligence. 2024. <https://eur-lex.europa.eu/eli/reg/2024/1689/oj>.
- Ezell, Carson, Xavier Roberts-Gaal, and Alan Chan. "Incident Analysis for AI Agents." arXiv preprint, August 2025. <https://arxiv.org/abs/2508.14231>.
- French, Laura. Microsoft 365 Copilot 'zero-click' vulnerability enabled data exfiltration. SC Media, June 2025. <https://www.scworld.com/news/microsoft-365-copilot-zero-click-vulnerability-enabled-data-exfiltration>.
- Google DeepMind. "Securing the Future of AI Agents." June 18, 2026. <https://deepmind.google/blog/securing-the-future-of-ai-agents/>.
- Green, Ben, and Yiling Chen. "The Principles and Limits of Algorithm-in-the-Loop Decision Making." Proceedings of the ACM on Human-Computer Interaction 3, CSCW (2019). <https://scholar.harvard.edu/files/bgreen/files/19-cscw.pdf>.

- Huang, Lei, Xiaocheng Feng, Weitao Ma, Yuxuan Gu, Weihong Zhong, Xiachong Feng, Weijiang Yu, et al. "Learning Fine-Grained Grounded Citations for Attributed Large Language Models." In Findings of the Association for Computational Linguistics: ACL 2024, 14095–14113. Bangkok: Association for Computational Linguistics, 2024. <https://arxiv.org/abs/2408.04568>.
- IBM. "What Is Telemetry?" IBM Think, October 13, 2025. <https://www.ibm.com/think/topics/telemetry>.
- IMDA (Infocomm Media Development Authority, Singapore). Model AI Governance Framework for Agentic AI. January 2026. <https://www.imda.gov.sg/resources/press-releases-factsheets-and-speeches/press-releases/2026/new-model-ai-governance-framework-for-agentic-ai>.
- Kadavath, Saurav, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, et al. "Language Models (Mostly) Know What They Know." arXiv preprint arXiv:2207.05221, July 2022. <https://arxiv.org/abs/2207.05221>.
- Khalifa, Muhammad, Lajanugen Logeswaran, Jaekyeom Kim, Sungryull Sohn, Yunxiang Zhang, Moontae Lee, Hao Peng, Lu Wang, and Honglak Lee. "Gaming the Judge: Unfaithful Chain-of-Thought Can Undermine Agent Evaluation." arXiv preprint arXiv:2601.14691, January 2026. <https://arxiv.org/abs/2601.14691>.
- Korbak, Tomek, Mikita Balesni, Elizabeth Barnes, Yoshua Bengio, Joe Benton, Joseph Bloom, Mark Chen, et al. "Chain of Thought Monitorability: A New and Fragile Opportunity for AI Safety." arXiv preprint arXiv:2507.11473, July 2025. <https://arxiv.org/abs/2507.11473>.
- Lakhina, Anukool, Mark Crovella, and Christophe Diot. "Diagnosing Network-Wide Traffic Anomalies." In Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '04), 219–230. New York: Association for Computing Machinery, 2004. <https://doi.org/10.1145/1030194.1015492>.
- LangChain. "Human-in-the-Loop." LangGraph Documentation. Accessed June 2026. <https://langchain-ai.github.io/langgraph/>.
- Langfuse. "Model-Based Evaluations." Langfuse Documentation. Accessed June 2026. <https://langfuse.com/docs/scores/model-based-evals>.
- Liu, Yang, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. "G-Eval: NLG Evaluation Using GPT-4 with Better Human Alignment." In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, 2511–2522. Singapore: Association for Computational Linguistics, 2023. <https://aclanthology.org/2023.emnlp-main.153>.
- METR. Independent Investigation of the OpenAI/Hugging Face Incident. August 2026. <https://metr.org/hugging-face-incident-report-aug-2026.pdf>.
- Microsoft. "What Is Microsoft Entra Agent ID?" Microsoft Learn. Last updated June 24, 2026. <https://learn.microsoft.com/en-us/entra/agent-id/what-is-microsoft-entra-agent-id>.
- Mitchell, Margaret, Avijit Ghosh, and Samir Passi. "AI Agents Push Humans Out of the Loop." arXiv preprint arXiv:2608.23642, August 2026. <https://arxiv.org/abs/2608.23642>.
- National Institute of Standards and Technology. Challenges to the Monitoring of Deployed AI Systems. NIST AI 800-4. March 2026. <https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.800-4.pdf>.
- NIST National Cybersecurity Center of Excellence. Accelerating the Adoption of Software and AI Agent Identity and Authorization. Concept paper, February 5, 2026. <https://www.nccoe.nist.gov/projects/software-and-ai-agent-identity-and-authorization>.
- NVIDIA. "NeMo Guardrails." NVIDIA Documentation. Accessed June 2026. <https://docs.nvidia.com/nemo/guardrails/>.
- OpenAI (a). "Reasoning Models." OpenAI API Documentation. Accessed June 2026. <https://developers.openai.com/api/docs/guides/reasoning>.
- OpenAI (b). "Using Logprobs." OpenAI Cookbook. Accessed June 2026. https://cookbook.openai.com/examples/using_logprobs.
- OpenID Foundation. Response to NIST Request for Information Regarding Security Considerations for Artificial Intelligence Agents (NIST-2025-0035). AI Identity Management Community Group, Threat Modeling Subgroup, March 6, 2026. https://openid.net/wp-content/uploads/2026/03/Attachment1_NIST-2025-0035-0001.pdf.
- OpenTelemetry (a). "Add GenAI Memory Operation Conventions." Pull Request #140, open-telemetry/semantic-conventions-genai, 2026. <https://github.com/open-telemetry/semantic-conventions-genai/pull/140>.
- OpenTelemetry (b). "Processors." OpenTelemetry Collector Documentation. Last modified March 16, 2026. <https://opentelemetry.io/docs/collector/components/processor/>.
- OpenTelemetry (c). "Semantic Conventions for Generative AI." open-telemetry/semantic-conventions-genai. Accessed July 2026. <https://github.com/open-telemetry/semantic-conventions-genai>.
- Oueslati, Amin, and Robin Staes-Polet. Ahead of the Curve: Governing AI Agents under the EU AI Act. The Future Society, June 2025. <https://thefuturesociety.org/aiagentsintheeu/>.
- PwC. "AI Observability." Accessed June 2026. <https://www.pwc.com/us/en/tech-effect/ai-analytics/ai-observability.html>.
- Sandhu, Ravi S., Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. "Role-Based Access Control Models." IEEE Computer 29, no. 2 (February 1996): 38–47. <https://csrc.nist.gov/csrc/media/projects/role-based-access-control/documents/sandhu96.pdf>.
- Sardine. The Agentic AI Oversight Framework. 2025. <https://www.sardine.ai/whitepapers/the-agentic-ai-oversight-framework>.
- SASH (Singapore AI Safety Hub). Designing Agent IDs. Policy memo, March 2026. <https://agent-id.org/docs/agent-ids-policy-memo-sash-2026-03-31.pdf>.

- Shavit, Yonadav, Sandhini Agarwal, Miles Brundage, Steven Adler, Cullen O’Keefe, Rosie Campbell, Teddy Lee, et al. Practices for Governing Agentic AI Systems. OpenAI, December 2023. <https://cdn.openai.com/papers/practices-for-governing-agentic-ai-systems.pdf>.
- Shi, Tianneng, Jingxuan He, Zhun Wang, Linyu Wu, Hongwei Li, Wenbo Guo, and Dawn Song. “Progent: Programmable Privilege Control for LLM Agents.” arXiv preprint arXiv:2504.11703, April 2025. <https://arxiv.org/abs/2504.11703>.
- South, Tobin, Samuele Marro, Thomas Hardjono, Robert Mahari, Cedric Deslandes Whitney, Dazza Greenwood, Alan Chan, and Alex Pentland. “Authenticated Delegation and Authorized AI Agents.” In Proceedings of the 42nd International Conference on Machine Learning (ICML), 2025. <https://arxiv.org/abs/2501.09674>.
- SPIFFE. “SPIFFE Overview.” Accessed June 2026. <https://spiffe.io/docs/latest/spiffe-about/overview/>
- Srikumar, Madhulika, Jacob Pratt, Kasia Chmielinski, Carolyn Ashurst, Chloé Bakalar, William Bartholomew, Rishi Bommasani, et al. Prioritizing Real-Time Failure Detection in AI Agents. Partnership on AI, September 2025. <https://partnershiponai.org/wp-content/uploads/2025/09/agents-real-time-failure-detection.pdf>.
- Srikumar, Madhulika, and Vinh X. Nguyen. “We Can’t Monitor AI Agents at Scale. Here’s What It Will Take.” Tech Policy Press, July 16, 2026. <https://www.techpolicy.press/we-cant-monitor-ai-agents-at-scale-heres-what-it-will-take/>.
- The Digital Chamber. Response to NIST RFI on Software and AI Agent Identity and Authentication. April 2026. https://digitalchamber.wpenginepowered.com/wp-content/uploads/2026/04/TDC-NIST-RFI-Response_-_Software-and-AI-Agent-Identity-4.3.26.pdf.
- Tian, Katherine, Eric Mitchell, Allan Zhou, Archit Sharma, Rafael Rafailov, Huaxiu Yao, Chelsea Finn, and Christopher D. Manning. “Just Ask for Calibration: Strategies for Eliciting Calibrated Confidence Scores from Language Models Fine-Tuned with Human Feedback.” In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, 5433–5442. Singapore: Association for Computational Linguistics, 2023. <https://arxiv.org/abs/2305.14975>.
- Wang (a), Charles L., Trisha Singhal, Ameya Kelkar, and Jason Tuo. “MI9: An Integrated Runtime Governance Framework for Agentic AI.” arXiv preprint arXiv:2508.03858, 2025. <https://arxiv.org/abs/2508.03858>.
- Wang (b), Yanting, Wei Zou, Runpeng Geng, and Jinyuan Jia. “TracLLM: A Generic Framework for Attributing Long Context LLMs.” In Proceedings of the 34th USENIX Security Symposium, 3845–3864. Seattle: USENIX Association, 2025. <https://www.usenix.org/conference/usenixsecurity25/presentation/wang-yanting>.
- Willows, Sophia. “Leveraging Logprobs to Build Better Generative AI Systems.” November 11, 2024. <https://sophiabits.com/blog/leveraging-logprobs>.
- Xiong, Miao, Zhiyuan Hu, Xinyang Lu, Yifei Li, Jie Fu, Junxian He, and Bryan Hooi. “Can LLMs Express Their Uncertainty? An Empirical Evaluation of Confidence Elicitation in LLMs.” In Proceedings of the 12th International Conference on Learning Representations (ICLR), 2024. <https://arxiv.org/abs/2306.13063>.
- Zhang, Tianyi, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. “BERTScore: Evaluating Text Generation with BERT.” In Proceedings of the 8th International Conference on Learning Representations (ICLR), 2020. <https://arxiv.org/abs/1904.09675>.

Cite as: Srikumar, M., Mibuari, E., Leibowicz, C., Blili-Hamelin, B., Klyman, K., Leininger, D., Hall, K., Lu, X. H., McGregor, S., Nguyen, V., Oueslati, A., Sanka, A., Stanley, J., Stroebel, B., Tan, S., & Thandavarayan, S. (2026, September). *The observability gap in AI agents: What to fix before agents scale*. Partnership on AI.

Acknowledgments

We thank the following individuals for their valuable input on drafts of this paper and for their participation in a working group that informed this research: Stephanie Bell, Alan Chan, Deborah Eng, Liz Green, Maia Hamin, Elliot Jones, Jeffrey Lim, Simon Morrison, Vineeth Sai Narajala, JD Rosensweig, Helen Toner, Tyler Tracy, and Vanessa Wilfred.