# Auditable Agents

**Yi Nian*** **Aojie Yuan*** **Haiyue Zhang*** **Jiate Li*** **Yue Zhao**
FORTIS Lab, University of Southern California
{yinian, aojieyua, haiyuez, jiateli, yue.z}@usc.edu

## Abstract

Knowledge-intensive LLM agents call tools, query databases, delegate tasks, and trigger external side effects. Once an agent system can act in the world, the question is no longer only whether harmful actions can be prevented—it is whether those actions remain *answerable* after deployment. We distinguish *accountability* (the ability to determine compliance and assign responsibility), *auditability* (the system property that makes accountability possible), and *auditing* (the process of reconstructing behavior from trustworthy evidence). Our claim is direct: **no agent system can be accountable without auditability**.

To make this operational, we define five dimensions of agent auditability—action recoverability, lifecycle coverage, policy checkability, responsibility attribution, and evidence integrity—and identify three mechanism classes (*detect*, *enforce*, *recover*) whose temporal information-and-intervention constraints explain why, in practice, no single approach suffices. We support the position with layered evidence rather than a single benchmark: lower-bound ecosystem measurements suggest that even basic security prerequisites for auditability are widely unmet (617 security findings across six prominent open-source projects); runtime feasibility results show that pre-execution mediation with tamper-evident records adds only 8.3 ms median overhead; and controlled recovery experiments show that responsibility-relevant information can be partially recovered even when conventional logs are missing. We propose an *Auditability Card* for agent systems and identify six open research problems organized by mechanism class.

## 1 Introduction

LLM agents do not only generate text. They delete files, send messages, issue payments, invoke third-party skills, and cross permission boundaries (Yao et al., 2023; Qin et al., 2024). Once an agent system can cause external side effects, its failures are no longer content problems alone. They are system problems that require a different safety guarantee than alignment or pre-deployment evaluation.

Consider a deployed enterprise agent. It reads customer records, queries an external API, drafts an email, and sends it on behalf of a human operator. The next day, the recipient reports that the email disclosed information it should not have. Three questions then become central:

> *What happened? Did the system comply with policy? Who or what was responsible?*

In most deployed agent systems today, these questions cannot be answered with confidence. Logs are partial or absent (Dong et al., 2024). Error paths, retries, fallbacks, approvals, and inter-agent handoffs are often missing or weakly represented. Skill provenance is shallow. Even when records exist, they rarely support mechanical policy checking and are seldom protected against silent modification.

Recent work has made real progress on alignment, adversarial evaluation, and runtime defenses (Zou et al., 2023; Qi et al., 2024; Mazeika et al., 2024; Xu et al., 2024; Inan et al., 2023). These efforts reduce the probability of harmful actions. But they do not answer the post-deployment question: once an agent system has acted, can its behavior be reconstructed, checked against policy, and attributed to a responsible component?

**Position. Agent systems should be auditable.** No agent system can be accountable without auditability. This paper makes that claim precise and argues that auditability should be treated as a first-class design and evaluation requirement for agents.

**Three levels.** We distinguish three related concepts. **Accountability** is the goal: an auditor can

* Equal contribution.

The 4th Workshop on Towards Knowledgeable Foundation Models (KnowFM), co-located with ACL 2026.

determine whether the system complied with policy and assign responsibility for violations. **Auditability** is the enabling system property: the system produces, preserves, and exposes enough trustworthy evidence to make accountability possible. **Auditing** is the process: reconstructing behavior, checking policy, and assigning responsibility from the available evidence. These three levels are distinct from observability, monitoring, and alignment, which address related but different questions. We return to these distinctions in Appx. B.

**The paper's contribution.** This paper does not propose a single new algorithm. Instead, it advances a systems position: *agent auditability should be a first-class design and evaluation target*. We make the following contributions:

- **Five-Dimensional Auditability Framework.** We define five conditions that are jointly necessary for a defensible post-deployment audit: action recoverability, lifecycle coverage, policy checkability, responsibility attribution, and evidence integrity (§2).
- **Mechanism Classes.** We identify three classes of mechanism—*detect*, *enforce*, and *recover*—that operationalize the five dimensions across the system lifecycle, and argue that in practice no single temporal point can satisfy all five (§A).
- **Layered Evidence.** We support the position with three mutually reinforcing evidence blocks: public ecosystem measurements, runtime feasibility results, and missing-log recovery exp. (§3).
- **Auditability Card and Open Problems.** We propose an Auditability Card for agent systems—a compact reporting artifact analogous to model cards (Mitchell et al., 2019)—and identify six open research problems organized by mechanism class (Appx. C).

These dimensions are especially relevant for knowledge-intensive agents, where tool calls, retrieval steps, and knowledge-source provenance must remain traceable for faithful attribution. Sections on mechanism classes (Appx. A), related work (Appx. B), and the research agenda with Auditability Card (Appx. C) appear in the appendix; the formal execution model is in Appx. D.

## 2 Five Dimensions of Agent Auditability

The central conceptual contribution of this paper is a five-dimensional framework for agent auditability. Its purpose is not to add more terminology. It is to answer a precise question: *what, exactly, must be true of an agent system before a post-deployment audit can produce a defensible verdict?*

A defensible audit yields a verdict of the form

$$V = (\underbrace{s}_{\text{Action Recov.}}, \underbrace{c}_{\text{Lifecycle Cov.}}, \underbrace{\nu}_{\text{Policy Check.}}, \underbrace{r}_{\text{Attribution}}, \underbrace{\sigma}_{\text{Integrity}}), \quad (1)$$

where $s$ is the policy-relevant action under audit, $c$ is the execution context in which that action occurred, $\nu \in \{\text{comply}, \text{violate}\}$ is the policy verdict under a stated policy $\pi$, $r$ is the responsible component or responsibility chain, and $\sigma$ is the integrity guarantee protecting the supporting record. Each slot requires a distinct auditability condition, annotated beneath the corresponding element in Eq. (1). These five conditions—Action Recoverability, Lifecycle Coverage, Policy Checkability, Responsibility Attribution, and Evidence Integrity—are jointly necessary. Removing any one renders the verdict incomplete or untrustworthy.

Table 1 summarizes the framework. The five dimensions can be read as the five questions an auditor must answer: *what happened*, *in what context*, *whether it complied*, *who was responsible*, and *whether the evidence can be trusted*. Each dimension is paired with two metrics: one for existence (does the evidence appear at all?) and one for quality (is it strong enough to use?). The subsections below develop each dimension through a concrete scenario and define its metrics informally; the formal execution and definitions are in Appx. D.

### 2.1 Action Recoverability

The first question in any audit is whether the system left a usable record of the actions that matter. The relevant unit is not every latent model state. It is the set of *policy-relevant actions*: tool invocations, external requests, file operations, database queries, approvals, and delegation events. In practice, many existing agent systems and observability tools record some actions but omit the fields needed to reconstruct what actually happened (§B).

Consider an agent that calls a database API. The audit log records that a call was made but omits the query arguments and the returned data. The action is *covered* but not *recoverable*: a shallow log scores well on coverage while scoring poorly on fidelity. We formalize this distinction through two metrics: *Action Coverage Rate (ACR)*, which measures whether policy-relevant actions appear in the record at all, and *Record Fidelity (RF)*, which measures

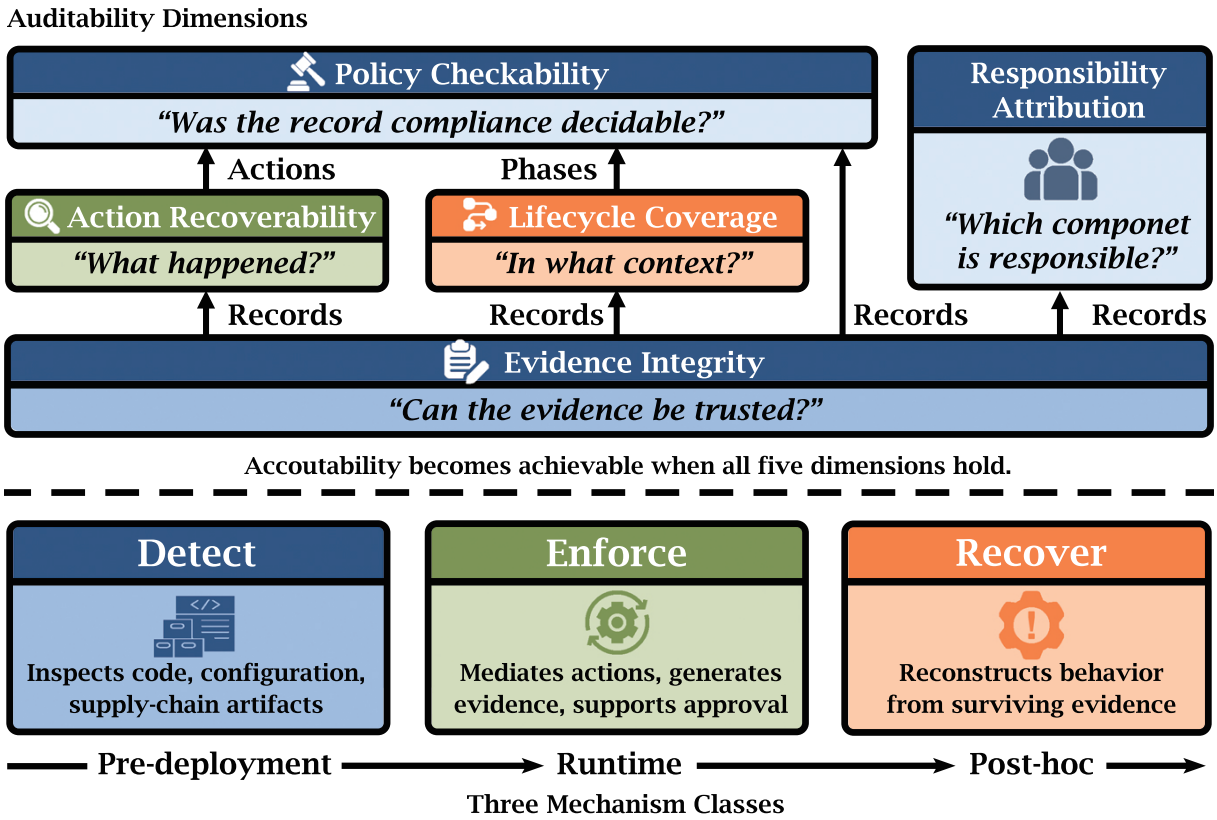


Figure 1: The auditability framework. An agent system's actions must be answerable after deployment. Auditability requires five dimensions (center), each corresponding to a necessary condition for a defensible audit verdict (Eq. 1). Evidence Integrity is foundational; Action Recoverability and Lifecycle Coverage jointly enable Policy Checkability; Responsibility Attribution is independently necessary. Three mechanism classes (bottom)—detect, enforce, recover—operating at successive temporal vantage points realize these dimensions in practice. No single mechanism class suffices (Table 1).

whether enough fields survive for each recorded action to support reconstruction (Appx. D.2).

### 2.2 Lifecycle Coverage

Recording individual actions is necessary but not sufficient. Even a fully recorded action may be unauditable if the auditor cannot reconstruct the execution context in which it occurred. Lifecycle Coverage operates over execution *phases* rather than individual steps and asks whether the record covers the full execution structure. This is among the most neglected dimensions in existing agent tools: retries, fallbacks, approvals, and delegation handoffs are rarely represented as identifiable phases in current traces (Appx. B).

Return to the enterprise agent from §1. Suppose the agent sends an email after querying customer records. The auditor can see both actions. But the record omits that the agent first attempted to send a different email, was blocked by a policy check, retried with modified content, and received human approval on the second attempt. Without that lifecycle context, the auditor cannot determine whether the final action was a clean execution or a policy-circumventing retry. *Lifecycle Phase Coverage (LPC)* measures the fraction of execution phases that are observed, and *Gap Burden (GB)* measures how much lifecycle structure is missing (Appx. D.3).

Note that LPC measures coverage of phases that actually occurred. It does not by itself distinguish "phase $p$ did not occur" from "phase $p$ occurred but was not recorded." Resolving that ambiguity may require explicit phase-entry and phase-exit markers. We treat this as an open measurement problem.

### 2.3 Policy Checkability

A complete record of actions and lifecycle phases is still not auditable if it cannot answer the policy question that motivated the audit. Policy Checkability asks whether the record contains enough information for compliance to be *mechanically*

Table 1: Five dimensions of agent auditability. Each dimension corresponds to a distinct slot in the audit verdict (Eq. 1) and a distinct auditor question. Accountability is not a sixth dimension; it is the derived property that emerges only when all five dimensions are sufficiently satisfied (§2.6).

| Dimension | Slot | Auditor question | Representative metrics |
|---|---|---|---|
| Action Recoverability | $s$ | What happened? | *ACR*: fraction of policy-relevant actions in the record.<br>*RF*: fraction of required fields recoverable per recorded action. |
| Lifecycle Coverage | $c$ | In what context? | *LPC*: fraction of observed lifecycle segments.<br>*GB*: total unobserved lifecycle content. |
| Policy Checkability | $\nu$ | Compliance decidable? | *SPDR*: fraction of policies decidable from the record.<br>*ADL*: time from violation to determination. |
| Responsibility Attribution | $r$ | Who was responsible? | *AC*: fraction of actions with full responsibility chain.<br>*ACD*: average recovered chain length. |
| Evidence Integrity | $\sigma$ | Evidence trustworthy? | *IS*: ordinal scale (none / append-only / hash-chained / signed).<br>*VC*: time to verify integrity. |

*determined*. Existing runtime enforcement tools can gate actions at execution time, but they do not typically support post-hoc compliance verification from recorded evidence (§B). The distinction matters: runtime blocking prevents harm in real time, while policy checkability enables accountability after the fact.

We focus on *structural policies*: machine-checkable rules such as "tool $X$ requires prior user approval" or "no external network call may follow access to data class $Y$ without sanitization." A structural policy $\pi$ evaluated against the audit record has three possible outcomes: $\mathrm{comply}$, $\mathrm{violate}$, or $\perp$ (undecidable from the record). This third outcome—undecidable, not merely unknown—is the reason the dimension exists. A policy is not always either satisfied or violated. It can be *impossible to decide* because the record omits a required field.

*Structural Policy Decidability Rate (SPDR)* measures the fraction of policies that are decidable from the record, and *Audit Detection Latency (ADL)* measures the delay from violation to determination (Appx. D.4).

**Proposition 1 (Record schema determines policy decidability).**
*Let $\pi$ be a structural policy with required field set $F_\pi$, and let $\mathcal{S}_\pi \subseteq \mathcal{S}$ be the set of execution steps whose recorded fields could contribute to deciding $\pi$ (including, e.g., approval steps for an approval-required policy). Suppose $\mathcal{S}_\pi \neq \emptyset$. If there exists $f \in F_\pi$ such that $f \notin \widehat{F}(s_i; \mathcal{L})$ for all $s_i \in \mathcal{S}_\pi$, then $\pi(\mathcal{L}) = \perp$.*

This is not a deep result, but it is a consequential one: a single field omitted from the record schema can render an entire class of policies uncheckable, regardless of how many events are logged. (Proof in Appx. D.4.)

### 2.4 Responsibility Attribution

Once the action is visible, the context is recovered, and the policy verdict is determined, the remaining question is responsibility. In simple systems, responsibility may appear local. In real deployments, it is often a chain: user $\rightarrow$ agent $\rightarrow$ skill $\rightarrow$ tool $\rightarrow$ service, possibly with a human approval step in between. While recent work on authenticated delegation addresses who is *permitted* to act, tracing who *actually* caused a given outcome across multi-agent delegation chains remains open (§B).

Consider a multi-agent system where Agent A delegates a task to Agent B, which invokes a third-party skill that calls an external API. The API returns sensitive data. The audit log records Agent B's API call, but not that Agent A initiated the task or that the skill was dynamically selected. The immediate executor is visible, but the responsibility chain is broken. *Attribution Completeness (AC)* measures the fraction of actions with a fully recovered chain, and *Attribution Chain Depth (ACD)* measures average recovered depth. When outcomes arise from joint behavior rather than a single delegation sequence, attribution generalizes from chain recovery to subgraph recovery over the interaction topology (Appx. D.5).

### 2.5 Evidence Integrity

The previous four dimensions all depend on the audit record being trustworthy. Yet integrity protection is nearly absent in existing agent systems: among the approaches surveyed in §B, only two provide any form of tamper-evident or cryptographically protected records. Without integrity guarantees, a record that appears to satisfy Action Re-

coverability, Lifecycle Coverage, Policy Checkability, and Responsibility Attribution may have been silently modified after the fact. Evidence Integrity is therefore foundational.

We define *Integrity Strength (IS)* on an ordinal scale:

- **Level 0 (none):** no verification mechanism; entries can be modified without detection.
- **Level 1 (append-only):** entries cannot be deleted or reordered, but individual entries are not cryptographically bound.
- **Level 2 (hash-chained):** entries are linked by collision-resistant hashes, so modifying one entry invalidates subsequent links.
- **Level 3 (signed):** entries or batches are digitally signed, enabling third-party verification without relying on the original system.

We pair this with *Verification Cost (VC)*: the time and compute required to verify integrity over the full record. A mutable database table with no append-only or cryptographic protection may appear operationally convenient while still providing weak audit evidence (Appx. D.6).

### 2.6 From Dimensions to Auditability

The five dimensions above are not an arbitrary list. They are derived from the verdict structure in Eq. (1): each slot requires exactly one dimension, and no slot can be filled without its corresponding condition. In §B, we confirm empirically that no existing approach covers all five dimensions jointly, with Evidence Integrity and Lifecycle Coverage as the most neglected. This subsection shows why no fewer than five suffice, how the dimensions depend on each other, and how they combine into a formal definition of auditability.

**Why these five, and not six?** Each dimension can fail independently while the others hold. A system can log many events and still fail Policy Checkability because approval fields or data-flow markers are absent. It can record individual actions and still fail Lifecycle Coverage because retries, fallbacks, or escalations are not represented as identifiable phases. It can support policy checks and still fail Responsibility Attribution because delegated skill calls cannot be linked back to their source. And it can satisfy the first four in appearance while still failing Evidence Integrity if the record is mutable.

Possible candidates for a sixth dimension generally fall into one of two categories. Some are already captured by the existing five. For example, *timeliness* is captured by Audit Detection Latency under Policy Checkability, and *interpretability* is largely captured by Record Fidelity under Action Recoverability. Others are not dimensions of auditability itself but constraints on audit design. For example, *privacy* constrains how evidence can be collected, retained, or redacted, but it does not define an additional constituent of auditability. Any property that might be proposed as a sixth dimension is thus either reducible to one of the five or is a design constraint than a constituent of auditability.

**Dependency structure.** The five dimensions are not independent. Evidence Integrity is foundational: without it, the other four cannot be trusted. Action Recoverability and Lifecycle Coverage jointly enable Policy Checkability, which cannot operate on actions or phases absent from the record. Responsibility Attribution is parallel to Policy Checkability but independently necessary to complete the verdict. The dependency is therefore: Evidence Integrity underpins the other four; Action Recoverability and Lifecycle Coverage enable Policy Checkability; Responsibility Attribution is separately required for accountability.

**Existence vs. diagnostic metrics.** The auditability predicate thresholds only the existence-and-sufficiency metrics: ACR, RF, LPC, GB, SPDR, AC, and IS. The remaining metrics—ADL, ACD, and VC—are diagnostic. They characterize audit quality and operational burden after auditability is established, but they do not determine whether auditing is possible in principle. A system can be auditable while still having high detection latency, deep responsibility chains, or high verification cost.

**Definition 1 (Auditability).**
*Let $\Pi$ be a structural policy set and let $\theta = (\tau_{\mathrm{ACR}}, \tau_{\mathrm{RF}}, \tau_{\mathrm{LPC}}, \tau_{\mathrm{GB}}, \tau_{\mathrm{SPDR}}, \tau_{\mathrm{AC}}, \tau_{\mathrm{IS}})$ be a deployment-specific threshold vector. Execution $X$ is* auditable *with respect to $\Pi$, record $\mathcal{L}$, and $\theta$ if all existence metrics meet or exceed their thresholds (with $\mathrm{GB} \leq \tau_{\mathrm{GB}}$). The full formal statement with explicit inequalities is in Appx. D.7.*

Note that the auditability predicate is monotone: if every thresholded metric improves (or stays the same) and GB does not increase, auditability is preserved.

Accountability becomes achievable when au-

Table 2: Layered evidence. Each block validates one mechanism class from §A.

| Block | Question | Evidence |
|---|---|---|
| Ecosystem | Public default audit-ready? | Security scan of 6 agent projects; 617 findings in 8/10 OWASP categories. |
| Runtime | Control points practical? | Blocked 48/48 attacks; 1.2% FP on 500 benign calls; 8.3 ms overhead. |
| Recovery | Logs missing—accountability lost? | Approx. 0.93 IoU, 0.95 token acc., near 0.96 topology under degraded logging. |

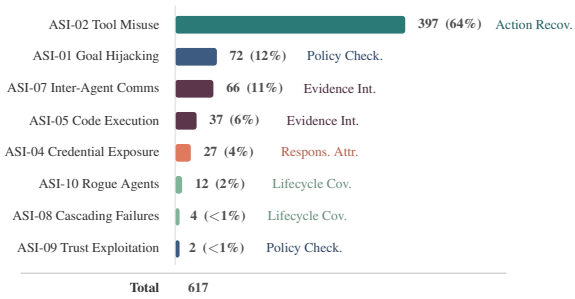


Figure 2: Security findings across 6 open-source agent projects by OWASP Agentic category (617 total), color-coded by the auditability dimension each category most directly undermines (§2). Tool Misuse alone accounts for 64% of all findings.

ditability holds and an auditor can, from $\mathcal{L}$ alone, determine $\pi_j(\mathcal{L}) \in \{\text{comply}, \text{violate}\}$ for each relevant policy $\pi_j \in \Pi$, and recover the relevant responsibility chain or interaction subgraph for each violation. Accountability is therefore not a sixth dimension. It is the derived property that becomes possible only when all five dimensions hold.

## 3 Evidence for the Auditability Gap

A position paper needs evidence, but not necessarily a single monolithic benchmark. The right question is not whether one system already solves auditing in every setting. It is whether there is enough evidence to make the position difficult to dismiss. In this section, the evidence is intentionally heterogeneous because it addresses three different objections, one for each mechanism class identified in §A. First, is the auditability gap real in the public ecosystem (*detect*)? Second, are auditable control points practical on the runtime execution path (*enforce*)? Third, does accountability collapse when logs are incomplete, redacted, or detached (*recover*)? We answer these questions with three mutually reinforcing evidence blocks: an ecosystem lower bound, a runtime feasibility layer, and a recovery frontier.

### 3.1 Ecosystem lower bound: the public default is audit-blind

**Question. Do publicly visible artifacts already show that minimal conditions for accountability are often absent?**

The first evidence block is intentionally conservative. We do not claim to measure end-to-end auditability from public artifacts alone. Instead, we ask a weaker and more defensible question: if basic *security* gaps are already visible in public agent code, can we at least conclude that the more demanding requirements of post-deployment auditing are unlikely to be met? Security is a prerequisite for auditability: a system with unvalidated tool inputs cannot produce trustworthy action records, and a system without inter-agent authentication cannot support responsibility attribution at all.

This analysis uses agent-audit (Zhang et al., 2026a), a static security analysis tool with 53 detection rules mapped to the OWASP Agentic Top 10 categories (2026). Agent-audit performs tool-boundary taint tracking, MCP configuration auditing, and semantic credential detection across Python agent codebases. At ecosystem scale, the tool has been validated against 18,899 community-contributed skills on ClawHub.

For a detailed view, a public scan report accompanying agent-audit (Zhang et al., 2026a) analyzed 6 prominent open-source agent projects—OpenHands (Wang et al., 2025), Generative Agents (Park et al., 2023), SWE-agent (Yang et al., 2024), Gorilla (Patil et al., 2024), MLAgentBench (Huang et al., 2024), and CodeAct (Wang et al., 2024)—representing the current state of agentic AI development. Agent-audit identified 617 security findings, of which 269 (44%) were classified as critical severity and 134 findings were in the highest confidence tier. The OWASP category breakdown reveals where the gaps concentrate:

Tool Misuse dominates: 64% of findings involve tool functions that accept unvalidated input from the LLM, enabling injection, exfiltration, and command execution. These are not exotic attack paths—they are the default development pattern. General-purpose static analysis tools (Bandit, Semgrep) achieve 0% recall on MCP configuration vulnerabilities and substantially lower recall on agent-specific patterns; agent-audit achieved 3–4× higher recall on a curated agent-vulnerability benchmark (Zhang

et al., 2026a).

These findings are security measurements, not direct auditability metrics. But the connection is tight: every OWASP category maps to at least one auditability dimension (Figure 2), and systems that fail these basic security checks are unlikely to produce the trustworthy evidence that auditing requires. A supplementary platform-level scan of an AI assistant with a skills marketplace found analogous patterns, including supply-chain risks in community-contributed skill definitions (Appendix E). The public ecosystem is not merely under-instrumented for auditing—it is under-secured for the actions agents already take.

### 3.2 Runtime feasibility: auditable control points are practical

**Question. Are auditable control points practical on the execution path?**

The second evidence block addresses a different objection: perhaps auditability is conceptually attractive but too expensive, too invasive, or too awkward to implement at runtime. Runtime feasibility evidence argues otherwise.

The runtime evaluation uses the Aegis pre-execution firewall (Yuan et al., 2026), which inserts a framework-agnostic mediation point between the LLM's tool-call decision and the underlying execution layer. Before any tool call executes, Aegis recursively extracts string content from the call arguments, scans for risk signals, and evaluates configurable policies. Each call receives one of three decisions: *allow*, *block*, or *pending* (escalated to a human reviewer via a compliance dashboard).

Across 14 supported agent frameworks (Python, JavaScript, Go), Aegis blocked all 48 curated attack instances (spanning 7 OWASP categories, including prompt injection, unauthorized tool use, and data exfiltration) before side effects occurred. On 500 benign tool calls (SELECT queries, file reads, API requests, text processing), it yielded a 1.2% false positive rate. End-to-end overhead including SDK extraction, HTTP round-trip, classification, and policy evaluation was 8.3 ms median across 1,000 consecutive interceptions (P95: 14.7 ms, P99: 23.1 ms—negligible relative to typical LLM inference latency of 1–30 s). The same runtime path generated tamper-evident evidence via Ed25519-signed, SHA-256 hash-chained records.

These aggregate numbers matter, but the dimension-level meaning matters more:

- *Action Recoverability*: the runtime layer records tool name, full arguments, output, timestamp, and policy decision for each intercepted call—supporting high record fidelity.
- *Lifecycle Coverage*: allow, block, pending, and approval events are recorded as distinct execution states rather than collapsed into a single success-path trace.
- *Policy Checkability*: structural policies are evaluated at interception time and the resulting decision is stored alongside the trace, making compliance mechanically decidable.
- *Evidence Integrity*: Ed25519-signed, SHA-256 hash-chained records correspond to the strongest integrity level in our framework (Level 3).
- *Responsibility Attribution*: only partially supported—current traces capture the immediate executor and session context but not the full upstream delegation chain.

These results do not show that one runtime layer solves agent safety. That is not the claim. They show something narrower and more important for this paper: pre-execution control, structured evidence generation, approval workflows, and tamper-evident logging are engineering-feasible with bounded overhead. The mechanisms required for auditable agents are not hypothetical.

### 3.3 Recovery frontier: accountability under missing logs

**Question. Does accountability collapse when logs are missing, redacted, or detached?**

The third evidence block addresses the hardest setting. As argued in §A, missing or detached logs are a structural feature of realistic agent deployment, not an edge case. If auditability depended only on ideal centralized logs, it would be too brittle for multi-party deployment.

The recovery analysis uses implicit execution tracing (IET) (Nian et al., 2026), a method that embeds agent-specific signals into the token distribution during generation using keyed watermarking. These signals are statistically undetectable without the key but allow post-hoc recovery of which agent produced which portion of a multi-agent output—even when explicit identity metadata, turn boundaries, and orchestration logs have been removed. At recovery time, a sliding-window scoring pass combined with change-point detection reconstructs

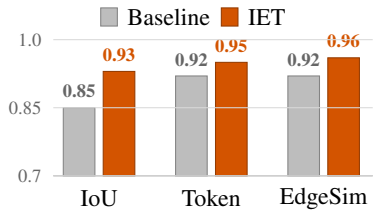


Figure 3: Recovery frontier: IET vs. best baseline (MAMA, 4–6 agents). Y-axis is zoomed into [0.7, 1.0] to highlight differences.

agent boundaries and interaction topology from the final text alone.

Evaluated across diverse multi-agent topologies (chain, star, tree) with 4–6 agents, IET achieved:

- *Token attribution accuracy* near 0.95 across topologies and agent counts.
- *Segment overlap* (IoU) of approximately 0.93 between recovered and ground-truth action segments.
- *EdgeSim* approaching 0.96—near-perfect reconstruction of the interaction graph.

These results held under three degradation conditions: identity removal (all agent identifiers stripped), boundary corruption (turn boundaries randomly reshuffled), and redaction (sensitive information replaced with placeholders). Figure 3 compares IET against the best baseline on each metric. Under full identity removal, IET maintained non-trivial agent-level attribution (23.8%) while baselines collapsed to near zero.

The dimensional interpretation is deliberately narrow:

- *Responsibility Attribution* (●): token attribution and topology recovery let an auditor infer which component produced which portion of the output and how components interacted.
- *Action Recoverability* (◐): segment overlap suggests that action-relevant boundaries can sometimes be reconstructed from surviving output alone.
- *Policy Checkability* and *Evidence Integrity* (–): not supported. Recovery cannot check policies or repair tampered records.

Recovery is therefore not a substitute for good logging. It is a frontier capability that becomes relevant when the sixth question on the Auditability Card—*what happens when logs are missing, redacted, or detached?*—has a non-trivial answer.

Table 3: Dimension-level coverage of the three evidence blocks. ● = direct evidence, ◐ = partial or proxy evidence, – = not addressed. No single block covers all five dimensions; together, they cover the full framework from complementary directions.

| | **Ecosystem (detect)** | **Runtime (enforce)** | **Recovery (recover)** |
|---|---|---|---|
| Action Recov. | ◐ | ● | ◐ |
| Lifecycle Cov. | ◐ | ● | – |
| Policy Check. | ◐ | ● | – |
| Respons. Attr. | ◐ | ◐ | ● |
| Evidence Int. | ◐ | ● | – |

**Synthesis.** Table 3 summarizes the dimensional coverage of the three evidence blocks. The ecosystem lower bound provides proxy evidence across all five dimensions but cannot certify any of them—it shows that public defaults are not audit-ready. The runtime feasibility block provides direct evidence for four dimensions, confirming that auditable control points are practical. The recovery frontier provides direct evidence for Responsibility Attribution and partial evidence for Action Recoverability, showing that accountability need not collapse completely when logging assumptions break down. Together, the three blocks cover the full auditability framework from complementary directions, justifying agent auditability as a first-class research and systems target.

## 4 Conclusion

This paper's core claim is straightforward: *agent systems should be auditable*. Once a system can take actions in the world, trust cannot depend only on alignment, evaluation, or runtime blocking. It also depends on whether the system leaves enough trustworthy evidence for others to determine.

We introduced *Auditable Agents* as a technical object, defined auditability through five dimensions—Action Recoverability, Lifecycle Coverage, Policy Checkability, Responsibility Attribution, and Evidence Integrity—and showed that temporal information-and-intervention constraints make three mechanism classes (detect, enforce, recover) each necessary in practice. Layered evidence from ecosystem scans, runtime mediation, and missing-log recovery supports the position that auditable agents are both needed and feasible. The Auditability Card and the six open problems in §C provide a concrete path from position to practice.

## Limitations

This paper proposes a framework and supporting evidence for agent auditability, but several limitations apply. First, the framework is system-level and does not yet address knowledge-specific audit concerns such as retrieval faithfulness or citation provenance in detail. Second, the ecosystem evidence relies on security findings as proxies for auditability gaps; these proxies are conservative lower bounds, not direct auditability measurements. Third, the runtime and recovery evidence comes from controlled settings; deployment-scale validation remains future work.

## References


Abeba Birhane, Ryan Steed, Victor Ojewale, Briana Vecchione, and Inioluwa Deborah Raji. 2024. AI auditing: The broken bus on the road to AI accountability. In *Proceedings of the 2nd IEEE Conference on Secure and Trustworthy Machine Learning*, pages 612–643.

Alan Chan, Carson Ezell, Max Kaufmann, Kevin Wei, Lewis Hammond, Herbie Bradley, Emma Bluemke, Nitarshan Rajkumar, David Krueger, Noam Kolt, Lennart Heim, and Markus Anderljung. 2024. Visibility into AI agents. In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*, pages 958–973. ACM.

Scott A. Crosby and Dan S. Wallach. 2009. Efficient data structures for tamper-evident logging. In *Proceedings of the 18th USENIX Security Symposium*, pages 317–334. USENIX Association.

Liming Dong, Qinghua Lu, and Liming Zhu. 2024. AgentOps: Enabling observability of LLM agents. *arXiv preprint arXiv:2411.05285*.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. 2021. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2024. MLAgentBench: Evaluating language agents on machine learning experimentation. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pages 20271–20309. PMLR.

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and 1 others. 2023. Llama guard: Llm-based input-output safeguard for human-ai conversations. *arXiv preprint arXiv:2312.06674*.

Weidi Luo, Shenghong Dai, Xiaogeng Liu, Suman Banerjee, Huan Sun, Muhao Chen, and Chaowei Xiao. 2025. AGrail: A lifelong agent guardrail with effective and adaptive safety detection. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pages 8104–8139. Association for Computational Linguistics.

Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, and 1 others. 2024. Harmbench: a standardized evaluation framework for automated red teaming and robust refusal. In *Proceedings of the 41st International Conference on Machine Learning*, pages 35181–35224.

Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. 2019. Model cards for model reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 220–229. ACM.

Jakob Mökander, Jonas Schuett, Hannah Rose Kirk, and Luciano Floridi. 2023. Auditing large language models: A three-layered approach. *AI and Ethics*, 4:1085–1115.

Zachary Newman, John Speed Meyers, and Santiago Torres-Arias. 2022. Sigstore: Software signing for everybody. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2353–2367. ACM.

Yi Nian, Haosen Cao, Shenzhe Zhu, Henry Peng Zou, Qingqing Luan, and Yue Zhao. 2026. When only the final text survives: Implicit execution tracing for multi-agent attribution. *arXiv preprint arXiv:2603.17445*.

Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. 2020. Backstabber's knife collection: A review of open source software supply chain attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 23–43. Springer.

Victor Ojewale, Harini Suresh, and Suresh Venkatasubramanian. 2026. Audit trails for accountability in large language models. *arXiv preprint arXiv:2601.20727*.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*.

Xiangyu Qi, Yi Zeng, Tinghao Xie, Pin-Yu Chen, Ruoxi Jia, Prateek Mittal, and Peter Henderson. 2024. Fine-tuning aligned language models compromises safety, even when users do not intend to! In *International Conference on Learning Representations (ICLR)*. Oral.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations (ICLR)*. Spotlight.

Inioluwa Deborah Raji, Andrew Smart, Rebecca N. White, Margaret Mitchell, Timnit Gebru, Ben Hutchinson, Jamila Smith-Loud, Daniel Theron, and Parker Barnes. 2020. Closing the AI accountability gap: Defining an end-to-end framework for internal algorithmic auditing. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 33–44. ACM.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. 2024. Identifying the risks of LM agents with an LM-emulated sandbox. In *Proceedings of the Twelfth International Conference on Learning Representations*.

Tobin South, Samuele Marro, Thomas Hardjono, Robert Mahari, Cedric Deslandes Whitney, Dazza Greenwood, Alan Chan, and Alex Pentland. 2025. Position: AI agents need authenticated delegation. In *Proceedings of the 42nd International Conference on Machine Learning*.

Peter Steinberger and OpenClaw Contributors. 2026. OpenClaw: Your own personal AI assistant. `https://github.com/openclaw/openclaw`.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. 2026. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. In *Proceedings of the 48th IEEE/ACM International Conference on Software Engineering (ICSE)*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better LLM agents. In *International Conference on Machine Learning (ICML)*.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, and 1 others. 2025. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations (ICLR)*.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Jinyuan Jia, Bill Yuchen Lin, and Radha Poovendran. 2024. Safedecoding: Defending against jailbreak attacks via safety-aware decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5587–5605.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Aojie Yuan, Zhiyuan Su, and Yue Zhao. 2026. AEGIS: No tool call left unchecked – a pre-execution firewall and audit layer for AI agents. *arXiv preprint arXiv:2603.12621*.

Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, Rui Wang, and Gongshen Liu. 2024. R-Judge: Benchmarking safety risk awareness for LLM agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1467–1490. Association for Computational Linguistics.

Haiyue Zhang, Yi Nian, and Yue Zhao. 2026a. Agent audit: A security analysis system for LLM agent applications. *arXiv preprint arXiv:2603.22853*.

Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. 2026b. Agent-SafetyBench: Evaluating the safety of LLM agents. In *AAAI 2026 Workshop on Trustworthy AI Agents (TrustAgent)*. Poster.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. 2023. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*.

# Supplementary Material for Auditable Agents

## A Realizing the Five Dimensions

The five dimensions in §2 define the conditions a defensible audit must satisfy. In practice, no single temporal vantage point can supply all of them. Before deployment, one can inspect code and configuration but not realized behavior. During execution, one can observe and mediate live actions but only within the runtime boundary. After the fact, one can aggregate surviving evidence across systems but cannot recreate evidence that was never captured or protected. We therefore identify three classes of mechanism—*detect*, *enforce*, and *recover*—each operating where its temporal vantage point provides the strongest observational access and intervention affordance. §3 provides empirical support for each class in turn.

Table 1: Which mechanism classes support which dimensions. ● = direct support (evidence generation or post-hoc establishment), ◐ = partial or proxy signal, ✓ = verification target. No single column is fully filled: in practice, robust auditability draws on all three.

| | Detect | Enforce | Recover |
|---|---|---|---|
| Action Recoverability | ◐ | ● | ● |
| Lifecycle Coverage | ◐ | ● | ● |
| Policy Checkability | ◐ | ● | ● |
| Responsibility Attribution | ◐ | ◐ | ● |
| Evidence Integrity | ◐ | ● | ✓ |

Table 1 shows that no single mechanism class covers all five dimensions, and the reason is informational: Detect can inspect code and configuration artifacts, but it cannot certify properties of a realized execution $X$ or its actual record $\mathcal{L}$ (§2). Enforce is the only temporal point that can both observe live actions and create protected evidence as those actions occur—which is why it fills four direct cells in the table. Recover operates on the surviving record $\mathcal{L}$ as given; it cannot add fields that were never captured or strengthen integrity guarantees retroactively (Appendix D.8). In practice, robust audit support therefore draws on all three, and the evidence in §3 is organized accordingly: ecosystem measurements validate detect, runtime feasibility validates enforce, and missing-log recovery validates recover.

### A.1 Detect before deployment

**Mechanism: static analysis of code, configuration, and supply-chain artifacts.** Return to the enterprise agent from §1. Before deployment, a static scan could flag that the email-sending skill has no structured logging at the invocation site, that the customer-record query exposes no approval hook, and that no signature or version pin protects the skill's provenance metadata. None of these findings prove that a violation will occur at runtime. But each one identifies an auditability gap that, left unaddressed, will make post-deployment accountability harder or impossible.

More generally, detect asks: Are policy-relevant actions instrumented? Are approval paths visible? Are risky skills signed or version-pinned? Table 1 marks detect as partial (◐) across all five dimensions because static artifacts can flag likely gaps but cannot guarantee that those properties will hold at runtime. The ecosystem scan in §3 provides quantitative evidence for the scale of these gaps in the public agent ecosystem.

### A.2 Enforce during execution

**Mechanism: runtime mediation of side-effecting actions.** The *enforce* class intercepts actions before they execute, evaluates them against active policy, supports human approval where needed, and emits structured, policy-relevant records. In the enterprise-agent scenario, enforce is the mechanism that would intercept the email-send action, check whether the drafted content violates a data-handling policy, require human approval if configured, and—regardless of the allow or block decision—emit a signed, timestamped record of the action, the policy evaluation, and the outcome.

Enforcement does two things at once: it reduces risk in real time and improves the quality of later auditing by generating complete, structured evidence. Runtime enforcement does not compete with auditability—it is one of the main ways auditability becomes technically feasible. Table 1 marks enforce as direct (●) for four dimensions. It is only partial for Responsibility Attribution because runtime systems often capture the immediate executor more reliably than the full upstream delegation chain. The runtime feasibility experiments in §3 provide quantitative support for the practicality of this mechanism class.

### A.3 Recover after the fact

**Mechanism: post-hoc reconstruction and responsibility assignment.** The *recover* class operates after deployment. It reconstructs policy-relevant behavior from whatever evidence remains.

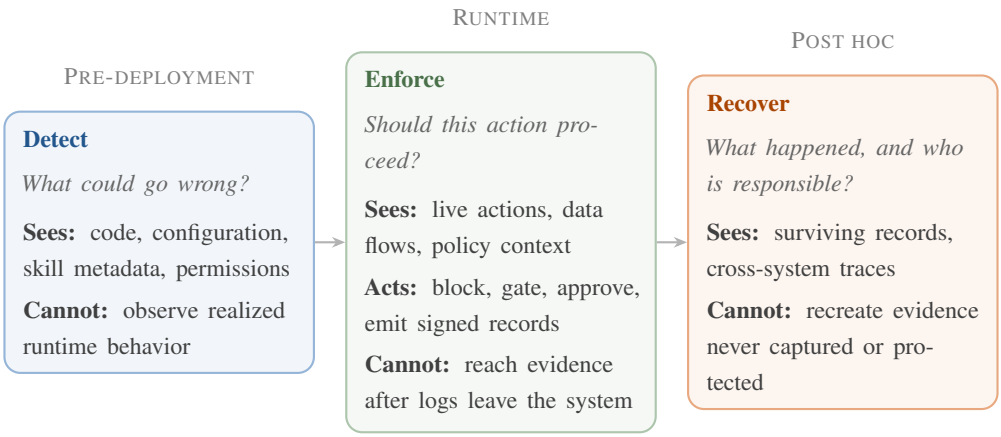


Figure 1: Information-and-intervention asymmetry across the three mechanism classes. Each box details what the class can observe (**Sees**) and what lies beyond its reach (**Cannot**), explaining why no single temporal vantage point can satisfy all five auditability dimensions (Table 1). Figure 1 introduces the classes at a high level; this figure unpacks their operational constraints.

In the enterprise-agent scenario, suppose the email has already been sent and the recipient reports an information disclosure the next day. If enforce was active, the auditor queries the signed trace, reconstructs the full execution path, and determines whether the data-handling policy was violated. But if the agent's output was forwarded into a ticketing system that stripped execution metadata, or if the deployment spans vendors who each hold only a partial trace, recovery must work from incomplete evidence.

This setting is not hypothetical. In multi-party deployments, agent outputs are routinely copied into reports, emails, or downstream systems where execution metadata is stripped. No single party may hold the complete trace. Recovery under missing or detached logs is therefore not an edge case but a structural feature of realistic agent deployment.

Recover is the canonical setting in which accountability is established: the auditor determines what happened, whether it complied with policy, and who or what was responsible. The missing-log recovery experiments in §3 show that partial recovery of actions and responsibility is feasible even when conventional logs fail.

**Why no single temporal point suffices.** Auditability cannot be retrofitted at a single point in the system lifecycle. A system that detects risks before deployment but generates no structured evidence at runtime will fail on Policy Checkability and Evidence Integrity when it matters most. A system that enforces policy at runtime but cannot support post-hoc recovery will fail when logs are incomplete, disputed, or detached. The enterprise-agent scenario illustrates this directly: detect would have flagged the missing hooks before deployment, enforce would have intercepted the email and generated a signed record, and recover would have reconstructed responsibility even after metadata was stripped. No single class could have done all three.

## B Related Work and Alternative Views

Table 2 positions existing work against the five auditability dimensions defined in §2. The table reveals two patterns. First, no existing work covers all five dimensions jointly. Second, Evidence Integrity and Lifecycle Coverage are the most neglected dimensions across all categories. We discuss each category below and then engage the strongest counterarguments directly.

**Safety evaluation benchmarks.** A growing body of work evaluates the safety of LLM agents before deployment. ToolEmu (Ruan et al., 2024) uses an LM-emulated sandbox to identify risks across 36 high-stakes toolkits. R-Judge (Yuan et al., 2024) benchmarks safety risk awareness across multi-turn agent interactions, and Agent-SafetyBench (Zhang et al., 2026b) evaluates 16 agents across 8 risk categories, finding that none achieves a safety score above 60%. All three record structured agent traces (partial Action Recoverability) and evaluate safety through LLM-based classifiers (partial Policy Checkability). However, none produces policy-grade audit records, models the full execu-

Table 2: Positioning of existing work against the five auditability dimensions (§2). ● = directly addressed, ◐ = partially addressed, – = not addressed. **Target**: what is audited. **Scope**: Pre = pre-deployment, Run = runtime, Post = post hoc, All = full lifecycle. Assessments reflect what each paper explicitly specifies or implements. Dimension abbreviations: AR = Action Recoverability, LC = Lifecycle Coverage, PC = Policy Checkability, RA = Responsibility Attribution, EI = Evidence Integrity (see Eq. 1).

| Category | Work | Target | Scope | AR | LC | PC | RA | EI |
|---|---|---|---|---|---|---|---|---|
| *Safety eval.* | ToolEmu (Ruan et al., 2024) | Agent behavior | Pre | ◐ | – | ◐ | – | – |
| | R-Judge (Yuan et al., 2024) | Agent behavior | Pre | ◐ | – | ◐ | ◐ | – |
| | Agent-SafetyBench (Zhang et al., 2026b) | Agent behavior | Pre | ◐ | ◐ | ◐ | ◐ | – |
| *Runtime enf.* | AgentSpec (Wang et al., 2026) | Agent actions | Run | ◐ | – | ◐ | – | – |
| | AGrail (Luo et al., 2025) | Agent actions | Run | ◐ | – | ◐ | – | – |
| *Observability* | AgentOps (Dong et al., 2024) | Agent traces | Run | ◐ | ◐ | ◐ | – | – |
| *Audit frmwk.* | SMACTR (Raji et al., 2020) | Dev. process | Pre | ◐ | ◐ | ◐ | ● | – |
| | Three-layer (Mökander et al., 2023) | LLM | All | ◐ | ◐ | ◐ | ◐ | – |
| *Accountability* | Auth. deleg. (South et al., 2025) | Agent deleg. | Run | ◐ | – | ● | ● | ◐ |
| | Visibility (Chan et al., 2024) | Agent deploy. | Run | ◐ | ◐ | ◐ | ◐ | – |
| | Audit trails (Ojewale et al., 2026) | LLM lifecycle | All | ● | ◐ | ◐ | ● | ● |
| | **This paper** | **Agent exec.** | **All** | ● | ● | ● | ● | ● |

tion lifecycle, or protects evidence from tampering. These benchmarks ask whether agents *will* behave safely. They do not ask whether, after deployment, an auditor can reconstruct what the agent *did*.

**Runtime enforcement.** AgentSpec (Wang et al., 2026) provides a domain-specific language for specifying runtime constraints with millisecond-level enforcement overhead. AGrail (Luo et al., 2025) uses cooperative LLMs to iteratively refine safety checks during test-time adaptation. Both gate actions at runtime (partial Policy Checkability) and maintain in-memory trajectories (partial Action Recoverability), but neither persists durable audit records, covers the full execution lifecycle, traces responsibility chains, or protects evidence integrity. As argued in §A, enforcement is not an alternative to auditability—it is one of the main ways auditability becomes technically feasible, because it generates structured evidence as a byproduct.

**Observability and audit frameworks.** AgentOps (Dong et al., 2024) surveys 17 observability tools and defines a span taxonomy for tracing agent behavior, but the paper itself notes that "trace links and interactions between different steps may have been missed" and provides no mechanisms for attribution or evidence integrity. At the framework level, SMACTR (Raji et al., 2020) proposes an end-to-end internal audit process with explicit stakeholder mapping (full Responsibility Attribution), and Mökander et al. (2023) distinguish governance, model, and application audit layers. Birhane et al. (2024) show that 58% of auditors lack necessary access, revealing structural barriers. These frameworks target development processes, governance structures, and model-level properties. They do not address the system-level properties that arise when models are embedded in agent architectures that take actions, delegate tasks, and interact with external services. Evidence Integrity is unaddressed across all of them.

**Agent accountability.** The most closely related work addresses accountability at the agent level. Chan et al. (2024) taxonomize what visibility measures are needed for agent oversight but operate at the "what should exist" level without implementation mechanisms. South et al. (2025) extend OAuth-style credentials with agent-specific delegation tokens, achieving full Policy Checkability and Responsibility Attribution through machine-readable policies and signed tokens, but covering only the authorization moment and not the full execution lifecycle. Ojewale et al. (2026) come closest to our goals: their audit trails use SHA-256 hash-chained, append-only records with explicit actor fields, covering five model lifecycle stages with strong Evidence Integrity. However, their lifecycle is the *model* lifecycle (pre-training, fine-tuning, deployment, monitoring), not the *agent runtime* lifecycle (tool calls, sub-agent delegation, retries, approval workflows, fallback paths). Their event taxonomy includes `FineTuneStart` and `DeploymentCompleted` but not the side-effecting actions, dynamic skill invo-

cations, and multi-step execution chains that define agent behavior. Their Policy Checkability is also partial: governance checkpoints are recorded as first-class events, but no policy engine or formal verification mechanism is specified. Our framework is complementary: it defines auditability conditions specifically for the agent runtime setting, with formal metrics that make each dimension measurable across the detect–enforce–recover lifecycle.

**Software audit infrastructure and documentation.** Tamper-evident logging (Crosby and Wallach, 2009) provides the cryptographic foundation for our Evidence Integrity dimension. Sigstore (Newman et al., 2022) demonstrates that verifiable signing can be made practical at ecosystem scale, a model transferable to agent action attestation. Software supply-chain security (Ohm et al., 2020) faces similar provenance challenges, compounded in agent ecosystems by dynamic skill selection. Model cards (Mitchell et al., 2019) and datasheets (Gebru et al., 2021) established the norm that ML artifacts should ship with structured documentation. Our minimal Auditability Card (§C) extends this paradigm from static artifacts to deployed agent executions.

We now engage four alternative views that a skeptical reviewer might reasonably hold.

**Alternative view 1: observability is enough.** Modern tracing and dashboarding systems provide useful infrastructure. AgentOps (Dong et al., 2024) defines detailed span types and metadata for agent monitoring. But as Table 2 shows, observability tools leave Responsibility Attribution and Evidence Integrity entirely unaddressed. A dashboard that records prompts and tool names but omits approval events, data categories, caller identities, or integrity guarantees can still fail Auditability on multiple dimensions. Observability tells an operator *that* something happened. Auditability asks whether the record is sufficient to determine *what* happened, *whether* it complied with policy, and *who* was responsible.

**Alternative view 2: runtime blocking matters more than post hoc audit.** Runtime blocking matters a great deal. Systems like AgentSpec (Wang et al., 2026) and AGrail (Luo et al., 2025) demonstrate that enforcement is practical and effective. But blocking does not remove the need for auditing. A blocked action still needs an explanation. An allowed action still needs a trace. A human override still needs provenance. Blocking reduces risk in real time. Auditing makes behavior answerable after the fact. Our runtime feasibility evidence (§3) confirms this: pre-execution mediation adds single-digit-millisecond overhead while generating tamper-evident records as a byproduct.

**Alternative view 3: stronger alignment will make auditability less important.** This view assumes that harmful behavior is mainly a model-internal problem. In deployed agent systems, many failures arise from dynamic tool behavior, changing external services, skill updates, human approvals, and multi-agent composition (Ruan et al., 2024; Zhang et al., 2026b). These are deployment properties, not model properties. Better alignment helps, but it does not by itself establish accountability for what a specific deployed system actually did. Even a perfectly aligned model embedded in a poorly instrumented agent system remains unauditable.

**Alternative view 4: auditability is too costly or too invasive.** Poor evidence design can indeed be expensive or privacy-invasive. But the correct response is not to abandon auditability. It is to design evidence better. Selective recording, policy-scoped schemas, redaction, encryption, hashed summaries (Crosby and Wallach, 2009), and metadata-light recovery are all ways to reduce cost while preserving accountability. Our own evidence (§3) shows that pre-execution mediation adds 8.3 ms median overhead, and that partial recovery of actions and responsibility can be recovered even when conventional logs are missing (Nian et al., 2026). The software supply-chain community has shown that verifiable signing can be made practical at scale (Newman et al., 2022). Cost and privacy are evidence-design constraints, not reasons to give up on answerable systems.

## C Auditability Card and Open Problems

The evidence in the preceding sections shows that the auditability gap is real, that closing it is engineering-feasible, and that partial recovery of actions and responsibility survives even when logs fail. The remaining question is adoption. This section proposes two deliverables: an *Auditability Card* that agent systems can report immediately, and six open research problems that the community must solve to make auditable agents the default.

Table 3: The Auditability Card. Q1–Q5 map to the five dimensions; Q6 is a stress test. The rightmost column is an illustrative partial card for Aegis (Yuan et al., 2026), not a canonical answer.

| Question | What to disclose | Illustrative example (Aegis) |
|---|---|---|
| Q1: Actions (Action Recov.) | What policy-relevant actions does the system record? | Tool calls with name, full arguments, output, timestamp, and policy decision |
| Q2: Phases (Lifecycle Cov.) | Which execution phases are covered? | Allow, block, pending, and approval as distinct states; delegation chains not covered |
| Q3: Policies (Policy Check.) | What policies can be mechanically checked? | Configurable structural rules; decision stored alongside each call |
| Q4: Attribution (Respons. Attr.) | What responsibility chain is available? | Immediate executor and session context; upstream delegation chain partial |
| Q5: Integrity (Evidence Int.) | What protects the record from modification? | Level 3: Ed25519-signed, SHA-256 hash-chained records |
| Q6: Missing logs | What happens when logs are missing or detached? | No built-in recovery; depends on external evidence |

**The Auditability Card.** Table 3 defines the Auditability Card: six questions that any agent paper, benchmark, framework, or skill ecosystem should answer. Q1–Q5 correspond to the five auditability dimensions (§2); Q6 stress-tests what happens when logging assumptions break down. The rightmost column shows an illustrative partial card for a runtime firewall (Yuan et al., 2026), demonstrating the format on a real system.

The card is deliberately compact. Its value lies not in comprehensiveness but in forcing disclosure: a system can answer these questions well or badly, but it should not be allowed to answer them ambiguously.

We envision four adoption paths:

- **Papers** claiming safety, reliability, or deployment readiness should include the card in their evaluation or appendix.
- **Benchmarks** should require card-level disclosure alongside task-performance metrics.
- **Frameworks** should auto-generate a partial card from runtime configuration, specifying which phases are logged and what integrity level is provided.
- **Skill ecosystems** should require card-level provenance metadata as a publication prerequisite (Ohm et al., 2020).

**Open problems.** The card addresses reporting. The harder question is what the community must still solve. We identify six open research problems, organized by the mechanism classes from §A. Each is grounded in a specific evidence gap from this paper.

**Detect.** *OP1: Predicting auditability gaps from code.* Our ecosystem scan (§3) uses security findings as proxies for auditability gaps. Can static analysis directly predict which of the five dimensions will be under-supported at runtime—before deployment—by analyzing code structure, logging instrumentation, and event-schema coverage?

*OP2: Minimal provenance for dynamic skills.* Supply-chain risks appeared across all scanned projects. What is the minimal provenance record—signature, version, permission scope, auditability card—that a dynamically selected skill must carry to support post-hoc responsibility attribution?

**Enforce.** *OP3: Full-chain attribution at runtime.* Our runtime evidence shows that Responsibility Attribution is only partially supported: the immediate executor is captured but the upstream delegation chain is not (§3). How can a mediation layer capture the full responsibility chain across multi-agent delegation without requiring all frameworks to share a single event schema?

*OP4: Semantic policy decidability.* Our framework restricts Policy Checkability to structural policies (§2.3). What classes of semantic policies—e.g., "the agent should not disclose information that could identify the customer"—can be made mechanically decidable from the audit record, and at what cost in record schema complexity?

**Recover.** *OP5: Adversarial recovery.* Our recovery evidence tests benign degradation: identity removal, boundary corruption, and redaction (§3). How robust is metadata-light recovery when the degradation is adversarial—when an attacker deliberately targets the attribution signal? What are the information-theoretic limits of recovery without explicit logs?

*OP6: Cross-party audit aggregation.* Multi-party deployments structurally fragment evidence (§A). How can multiple parties, each holding a partial trace with independent integrity guarantees,

produce a joint audit verdict without any single party holding the complete record?

**From position to practice.** The Auditability Card provides a reporting standard that can be adopted immediately. The six open problems define a research agenda for making auditable agents the default rather than the exception.

# D Formal Execution Model and Metric Definitions

This section provides the formal machinery underlying the five auditability dimensions introduced in §2. The main text defines each dimension through intuition and concrete examples; here we give precise mathematical definitions.

## D.1 Shared execution–record model

An *agent execution* is a tuple

$$X = (\mathcal{A}, \mathcal{S}, \psi, \rho),$$

where $\mathcal{A} = \{a_1, \ldots, a_m\}$ is the set of participating components (agents, tools, skills, services, or human principals), and $\mathcal{S} = (s_1, \ldots, s_n)$ is the sequence of execution steps. Each step is

$$s_i = (\mathrm{type}_i,\ \mathrm{in}_i,\ \mathrm{out}_i,\ t_i,\ \mathrm{ctx}_i),$$

where $\mathrm{type}_i$ is an action type, $\mathrm{in}_i$ and $\mathrm{out}_i$ are the step input and output, $t_i$ is a timestamp, and $\mathrm{ctx}_i$ is execution context, such as approval state, caller chain, or phase label. The function $\psi(s_i)$ assigns each step to a lifecycle phase, and $\rho(s_i)$ assigns each step a responsibility chain from the immediate executor back to the originating principal.

An *audit record* for execution $X$ is a tuple

$$\mathcal{L} = (\mathcal{R}, \sigma),$$

where $\mathcal{R} = (r_1, \ldots, r_k)$ is a sequence of record entries and $\sigma$ is an integrity mechanism, which may be null. Each record entry $r_j$ is a partial observation of one or more execution steps. We write $\phi(j) \subseteq \mathcal{S}$ for the set of execution steps observed by entry $r_j$, and $\mathrm{fields}(r_j)$ for the set of fields preserved by that entry. For any step $s_i$, let

$$\widehat{F}(s_i; \mathcal{L}) = \bigcup_{j:\, s_i \in \phi(j)} \mathrm{fields}(r_j)$$

denote the set of fields about $s_i$ that are recoverable from the audit record.

## D.2 Action Recoverability metrics

Let $\mathcal{S}_{\mathrm{rel}} \subseteq \mathcal{S}$ denote the policy-relevant steps in execution $X$. This set is specified by the deployment context and is defined independently of any particular policy set $\Pi$. We define

$$\mathrm{ACR}(X, \mathcal{L}) = \frac{|\{s_i \in \mathcal{S}_{\mathrm{rel}} : \widehat{F}(s_i; \mathcal{L}) \neq \emptyset\}|}{|\mathcal{S}_{\mathrm{rel}}|}.$$

For each step $s_i$, let $F_{\mathrm{obs}}(s_i)$ be the minimal set of observational fields needed to reconstruct its externally meaningful effect. We then define

$$\mathrm{RF}(X, \mathcal{L}) = \frac{1}{|\mathcal{S}_{\mathrm{cov}}|} \sum_{s_i \in \mathcal{S}_{\mathrm{cov}}} \frac{|F_{\mathrm{obs}}(s_i) \cap \widehat{F}(s_i; \mathcal{L})|}{|F_{\mathrm{obs}}(s_i)|},$$

where $\mathcal{S}_{\mathrm{cov}} = \{s_i \in \mathcal{S}_{\mathrm{rel}} : \widehat{F}(s_i; \mathcal{L}) \neq \emptyset\}$.

**Edge cases.** If $|\mathcal{S}_{\mathrm{rel}}| = 0$ (no policy-relevant actions occurred), we define $\mathrm{ACR} = 1$ by convention: there is nothing to record and nothing is missing. If $|\mathcal{S}_{\mathrm{cov}}| = 0$ and $|\mathcal{S}_{\mathrm{rel}}| > 0$ (policy-relevant actions exist but none are covered), we define $\mathrm{RF} = 0$. If $|\mathcal{S}_{\mathrm{rel}}| = 0$, both ACR and RF are 1 by convention: there is nothing to record and nothing to recover.

**Why both metrics are needed.** ACR and RF are not redundant. A system can achieve $\mathrm{ACR} = 1$ (every policy-relevant action appears in the record) while $\mathrm{RF} \approx 0$ (each entry records only that a tool was called, omitting arguments, output, caller identity, and approval context). Such a system *covers* every action but *recovers* none. Conversely, a system with $\mathrm{ACR} \ll 1$ but high RF for its covered actions records fewer events but records them well. The two metrics capture orthogonal failure modes: missing events versus missing fields.

## D.3 Lifecycle Coverage metrics

Let $\mathcal{I}(X) = \{I_1, \ldots, I_q\}$ be the maximal contiguous lifecycle segments induced by $\psi$, where each segment $I_\ell$ contains consecutive steps with the same phase label. We say that a segment $I_\ell$ is *observed* if the audit record contains enough information to infer both its existence and its phase label. We define

$$\mathrm{LPC}(X, \mathcal{L}) = \frac{|\{I_\ell \in \mathcal{I}(X) : I_\ell \text{ is observed}\}|}{|\mathcal{I}(X)|},$$

and

$$\mathrm{GB}(X, \mathcal{L}) = \sum_{I_\ell \in \mathcal{I}(X):\, I_\ell \text{ not observed}} |I_\ell|,$$

where $|I_\ell|$ denotes either duration or step count. A deployment must fix one unit (duration or step count) and use the same unit in the GB threshold $\tau_{\text{GB}}$.

**Edge case.** If $|\mathcal{I}(X)| = 0$ (the execution has no identifiable lifecycle segments), we define $\text{LPC} = 1$ and $\text{GB} = 0$: there is no lifecycle structure to miss.

**Why both metrics are needed.** LPC and GB measure different aspects of lifecycle coverage. A system with $\text{LPC} = 0.9$ might be missing a single short phase (GB = 1 step) or a single very long phase (GB = 1,000 steps). The coverage rate is the same, but the gap burden differs by three orders of magnitude. LPC captures *how many* phases are observed; GB captures *how much* execution content is missing.

### D.4 Policy Checkability metrics

Let $\Pi = \{\pi_1, \ldots, \pi_k\}$ be a policy set, where each $\pi_j(\mathcal{L}) \in \{\text{comply},\ \text{violate},\ \perp\}$. We define

$$\text{SPDR}(\Pi, \mathcal{L}) = \frac{|\{\pi_j \in \Pi : \pi_j(\mathcal{L}) \neq \perp\}|}{|\Pi|}.$$

For each violated and detected policy $\pi_j$, let $t_{\text{violate}}(\pi_j)$ denote the timestamp of the earliest action that violates $\pi_j$, and let $t_{\text{detect}}(\pi_j, \mathcal{L})$ denote the earliest time at which the violation is determinable from the audit record. We define

$$\text{ADL}(\pi_j, \mathcal{L}) = t_{\text{detect}}(\pi_j, \mathcal{L}) - t_{\text{violate}}(\pi_j).$$

**Edge case.** If $|\Pi| = 0$ (no structural policies are specified), we define $\text{SPDR} = 1$: there are no policies to check and none are undecidable. ADL is undefined when no violation is detected.

**Proof of Proposition 1.** By definition, $\pi$ is decidable from $\mathcal{L}$ if $\pi(\mathcal{L}) \in \{\text{comply}, \text{violate}\}$. Let $\mathcal{S}_\pi$ be the evidence steps for $\pi$ as defined in the proposition. Decidability requires that the union of recovered fields across all evidence steps includes every required field: $F_\pi \subseteq \bigcup_{s_i \in \mathcal{S}_\pi} \widehat{F}(s_i; \mathcal{L})$. If there exists $f \in F_\pi$ such that $f \notin \widehat{F}(s_i; \mathcal{L})$ for every $s_i \in \mathcal{S}_\pi$, then $f \notin \bigcup_{s_i \in \mathcal{S}_\pi} \widehat{F}(s_i; \mathcal{L})$, the inclusion fails, and $\pi(\mathcal{L}) = \perp$. □

### D.5 Responsibility Attribution metrics

For each step $s_i$, let $\rho(s_i) = (a_i^{(1)}, a_i^{(2)}, \ldots, a_i^{(d_i)})$ denote the ground-truth responsibility chain, ordered from the immediate executor back to the originating principal. Let $\widehat{\rho}(s_i; \mathcal{L})$ denote the longest recoverable prefix. We define

$$\text{AC}(X, \mathcal{L}) = \frac{|\{s_i \in \mathcal{S}_{\text{rel}} : |\widehat{\rho}(s_i; \mathcal{L})| = |\rho(s_i)|\}|}{|\mathcal{S}_{\text{rel}}|},$$

and

$$\text{ACD}(X, \mathcal{L}) = \frac{1}{|\mathcal{S}_{\text{rel}}|} \sum_{s_i \in \mathcal{S}_{\text{rel}}} |\widehat{\rho}(s_i; \mathcal{L})|.$$

**Edge case.** AC and ACD share the denominator $|\mathcal{S}_{\text{rel}}|$ with ACR. If $|\mathcal{S}_{\text{rel}}| = 0$, we define $\text{AC} = 1$ and $\text{ACD} = 0$ by the same convention: there are no actions to attribute.

When outcomes arise from joint behavior of multiple components rather than a single delegation sequence, responsibility is no longer a chain but a subgraph of the interaction topology $G = (\mathcal{A}, E)$, where $(a_u, a_v) \in E$ indicates that component $a_u$'s output influenced component $a_v$'s action. In such settings, Attribution Completeness generalizes from chain recovery to subgraph recovery.

### D.6 Evidence Integrity metrics

Integrity Strength (IS) is defined on the ordinal scale in §2. Verification Cost is

$$\text{VC}(\mathcal{L}) = \text{wall-clock time required to verify } \sigma \text{ over the full record.}$$

### D.7 Formal definition of auditability

**Definition 2 (Auditability — full formal statement).**
*Let $\Pi$ be a structural policy set and let*

$$\theta = (\tau_{\text{ACR}},\ \tau_{\text{RF}},\ \tau_{\text{LPC}},\ \tau_{\text{GB}},\ \tau_{\text{SPDR}},\ \tau_{\text{AC}},\ \tau_{\text{IS}})$$

*be a deployment-specific threshold vector. Execution $X$ is* auditable *with respect to policy set $\Pi$, record $\mathcal{L}$, and threshold vector $\theta$ if*

$$\begin{aligned} \text{ACR}(X, \mathcal{L}) &\geq \tau_{\text{ACR}}, & \text{RF}(X, \mathcal{L}) &\geq \tau_{\text{RF}}, \\ \text{LPC}(X, \mathcal{L}) &\geq \tau_{\text{LPC}}, & \text{GB}(X, \mathcal{L}) &\leq \tau_{\text{GB}}, \\ \text{SPDR}(\Pi, \mathcal{L}) &\geq \tau_{\text{SPDR}}, & \text{AC}(X, \mathcal{L}) &\geq \tau_{\text{AC}}, \\ \text{IS}(\sigma) &\geq \tau_{\text{IS}}. \end{aligned}$$

### D.8 Recovery bounds

**Remark (post-hoc recovery bounds).** Recovery operates on the surviving record $\mathcal{L}$ and is bounded by its content in two ways.

*Field recovery.* If a field $f$ required by some policy $\pi$ was omitted from every record entry and left no trace in the surviving content, no post-hoc analysis can recover it. Recovery can sometimes infer

responsibility-relevant information from surviving content (e.g., stylistic attribution from output text), but it cannot materialize fields that are entirely absent from $\mathcal{L}$.

*Integrity.* Post-hoc signing can raise the current integrity level of a record artifact, but it cannot retroactively certify that the record was unmodified before the signing event. A record that was mutable at write time may have been silently altered before post-hoc protection was applied. The integrity guarantee therefore covers only the period after signing, not the full execution history.

These bounds limit how far the recover mechanism class (§A) can compensate for gaps left by detect and enforce.

## E Platform-Level Security Scan: OpenClaw

The ecosystem evidence in §3 scans six open-source agent *projects*. To complement that project-level view, we include a platform-level scan of OpenClaw (Steinberger and OpenClaw Contributors, 2026), a full-featured open-source AI assistant with a gateway, extensions, and a skills marketplace.[1] Using agent-audit v0.18.2 (Zhang et al., 2026a), the scan produced 680 raw findings, of which 615 (90.4%) were auto-classified as likely false positives—predominantly extension-privilege findings that reflect OpenClaw's intentional plugin trust model rather than security defects. After automated triage, 65 findings remained active (31 confirmed, 34 requiring manual review), concentrated in two OWASP categories:

- *Credential exposure (ASI-04, 58 findings):* direct macOS Keychain access, hardcoded secrets, and NOPASSWD sudoers configuration. These reflect OpenClaw's architecture as a personal assistant that integrates with system credential stores.
- *Code execution (ASI-05, 7 findings):* unsandboxed subprocess calls in extensions and a `curl | bash` pattern in a community-contributed skill definition—a classic supply-chain risk.

Two findings are particularly relevant to the auditability framework:

- A skill instructed the agent to modify `MEMORY.md`, a file that influences long-term agent behavior. A compromised skill or prompt injection could use this as a persistence mechanism, poisoning the agent's context across future sessions. This is an Evidence Integrity concern: the agent's behavioral state can be silently modified without detection.
- The `curl | bash` pattern in a skill definition means that a dynamically selected skill can trigger arbitrary code execution from an unverified external source—an Evidence Integrity risk (code runs outside any protected record) that also creates a Responsibility Attribution gap, since the resulting actions cannot be traced back through the skill's provenance chain.

OpenClaw documents a mature trust model that explicitly scopes out several finding categories (e.g., prompt injection without a boundary bypass, workspace file writes under operator control). This illustrates the importance of explicit policy scoping: what counts as a violation depends on which policies the deployment chooses to enforce, a prerequisite for meaningful Policy Checkability (§2.3). The full scan report is available in the agent-audit repository.

[1] https://github.com/openclaw/openclaw