From Agent Traces to Trust: A Survey of Evidence Tracing and Execution Provenance in LLM Agents

Yiqi Wang *Yiqi.Wang@unisq.edu.au*
University of Southern Queensland, Australia

Jiaqi Zhang *3230602065@stmail.ujs.edu.cn*
Jiangsu University, China

Zhangkai Wu *zhangkai.wu@sydney.edu.au*
The University of Sydney, Australia

Taotao Cai* *taotao.cai@unisq.edu.au*
University of Southern Queensland, Australia

Zirui Liu *zirui.liu@pku.edu.cn*
Peking University, China

Zequn Sun *sunzq@nju.edu.cn*
Nanjing University, China

Mingkai Zheng *zhengmk@sustech.edu.cn*
Southern University of Science and Technology, China

Manqing Dong *manqing.dong@unswalumni.com*
The University of New South Wales, Australia

Yu-Xuan Qiu *qiu@bit.edu.cn*
Beijing Institute of Technology, Zhuhai, China

Xuefei Yin *x.yin@griffith.edu.au*
Griffith University, Australia

Yanming Zhu *yanming.zhu@griffith.edu.au*
Griffith University, Australia

Reviewed on OpenReview: *https://openreview.net/forum?id=iwYk6keMAM*

Abstract

Large language model (LLM)-based agents are moving beyond text generation to planning, tool use, retrieval, memory access, environmental interaction, and multi-agent collaboration. Because their outputs depend on information and execution steps across an agent run, their execution processes are difficult to verify, debug, and audit. Final-answer evaluation alone cannot resolve these difficulties: even a correct answer does not reveal which evidence supported its claims, whether tool calls were justified, how memory shaped decisions, or where execution failures originated. Relevant research addresses these questions across evidence attribution, tool-use safety, agent memory, and observability, but these areas emphasize different parts of the execution process. This survey brings them into a common framework through *execution provenance*, a typed graph of an agent run, and *evidence tracing*, its projection onto evidence-support relations. We define what this projection retains,

*Corresponding author.

discards, and adds, distinguishing evidence that *supports* a conclusion from execution that *influences* it. We introduce a six-dimensional taxonomy covering trace sources, evidence and execution units, provenance relations, tracing granularity and timing, representation forms, and trust functions. We complement this taxonomy with operational definitions of nine provenance relations and two cross-cutting attributes: provenance origin, describing how records are produced, and trace assurance, describing their protection against modification. Applying this framework to the surveyed systems and benchmarks reveals three gaps: limited integration of fine-grained provenance with multiple trust functions, no benchmark in our catalog that evaluates the complete provenance chain, and little reporting of tamper-evident or attested traces. These findings highlight why final-answer correctness alone is insufficient to establish process-level trust and motivate research on unified trace schemas, semantic provenance, memory and multi-agent lineage, provenance-aware safety and recovery, realistic benchmarks, trace assurance, and privacy-aware auditing. Related resources are collected and continuously updated at https://github.com/xiaoqi-7/Agent-Tracing-Survey.

Keywords LLM Agents, Agent Traces, Evidence Tracing, Execution Provenance, Provenance Graphs, Tool-Use Safety, Memory Provenance, Agent Observability, Runtime Guardrails, Trustworthy AI

1 Introduction

Large language model (LLM)-based agents are emerging as a powerful paradigm for complex task execution, moving beyond standalone text generation toward interactive execution pipelines that support reasoning, tool use, retrieval, memory access, environment interaction, and multi-agent collaboration (Schick et al., 2023; Yao et al., 2023; Shinn et al., 2023). As these capabilities expand, recent systems and benchmarks show that LLM agents increasingly operate across tools, external knowledge sources, code execution, web or GUI environments, and multi-agent workflows, creating complex execution contexts in which many intermediate sources, actions, and observations may jointly shape the final output (Qin et al., 2024; Liu et al., 2024; Zhou et al., 2024; Yao et al., 2025).

This creates a process-level accountability gap in LLM agent systems. Final-answer accuracy evaluates only the endpoint of an execution; it does not explain how an output was produced, which evidence supported each claim, whether tool calls were justified, whether memory items were relevant and trustworthy, or which execution step caused a failure. Recent studies make this limitation concrete: tool-safety benchmarks show that risks can arise from intermediate tool-use decisions, adversarial tool interactions, realistic tool environments, and MCP-based workflows (Ruan et al., 2024; Debenedetti et al., 2024; Vijayvargiya et al., 2026; Zong et al., 2026), while trace-oriented and multi-agent failure studies show that errors often require localizing responsible steps, components, agents, or error modes within long executions (Deshpande et al., 2025; Cemri et al., 2025; Zhang et al., 2026; Kong et al., 2026). These findings suggest that trustworthy agent systems require mechanisms that can record, connect, and reason over the evidence and execution steps underlying agent behavior.

Relevant work exists across several research areas, but each emphasizes different relations: attribution links claims to supporting sources, usually treating each source as given (Gao et al., 2023; Schreieder et al., 2026); observability records execution events (Dong et al., 2024; AlSayyad et al., 2026); information-flow defenses track the propagation of untrusted data (Debenedetti et al., 2026; Costa et al., 2025); and memory systems manage what is stored, updated, and retrieved across interactions (Packer et al., 2023; Zhong et al., 2024). These perspectives address complementary questions, yet their connections are often left implicit. Connecting them requires a framework that distinguishes what happened, what influenced an outcome, and what evidence supports it.

In this survey, we examine this need through the lens of *evidence tracing* and *execution provenance* (Figure 1).[1]

[1]This perspective is informed by provenance and trace modeling in software and distributed systems, including W3C PROV-DM for representing entities, activities, agents, and provenance relations (World Wide Web Consortium, 2013), OpenTelemetry for distributed execution traces (OpenTelemetry, 2026), and recent agent observability systems such as AgentOps and AgentTrace (Dong et al., 2024; AlSayyad et al., 2026).

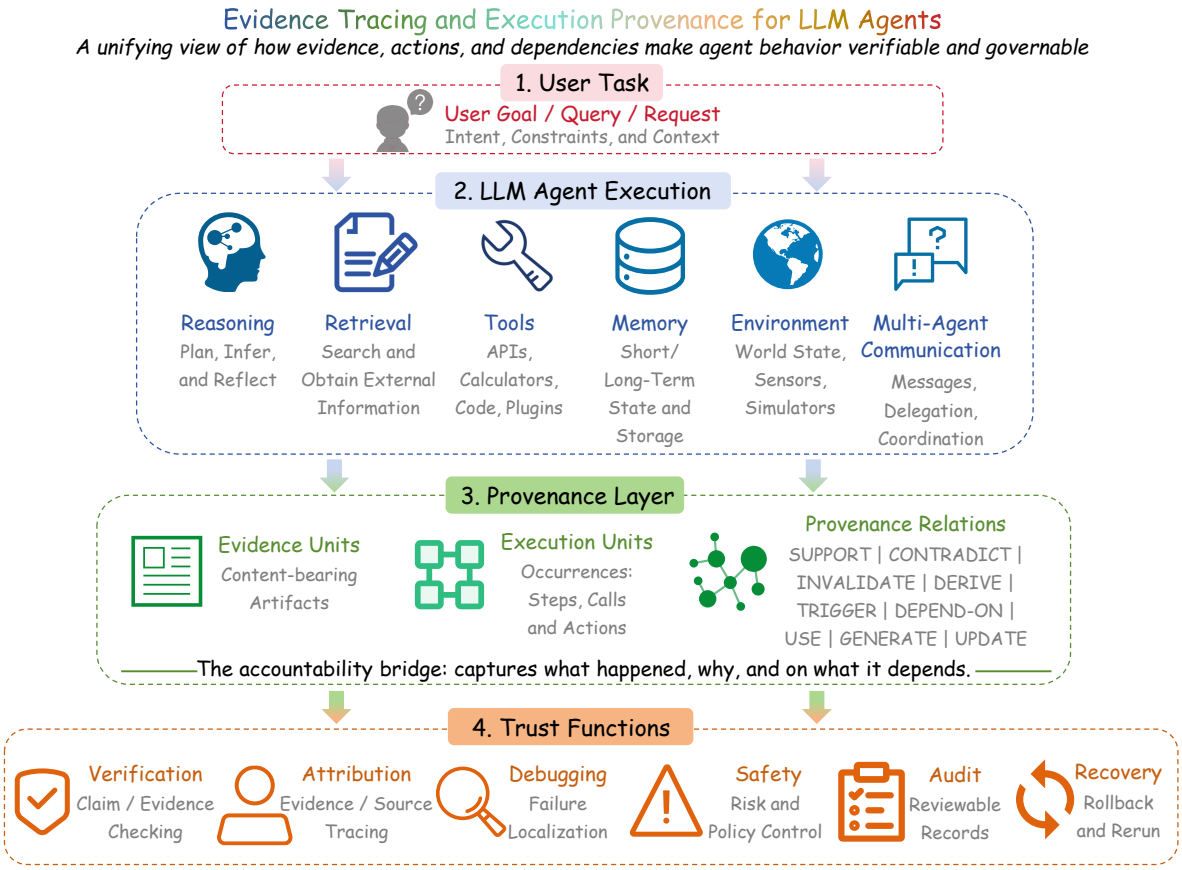


Figure 1: Overview of the proposed provenance framework. A user task initiates a heterogeneous agent execution involving reasoning, retrieval, tool use, memory, environment interaction, and multi-agent communication. The provenance layer records evidence units, execution units, and typed relations produced during this process, connecting agent execution to downstream trust functions such as verification, attribution, debugging, safety, audit, and recovery.

This perspective starts from **agent traces**: the recorded artifacts generated or consumed during an agent run, including instructions, retrieval queries, retrieved documents, tool calls, tool outputs, memory operations, observations, intermediate claims, inter-agent messages, actions, and final responses. Traceability makes these artifacts inspectable, while provenance goes further by modeling the typed relations among them.

Execution provenance. A structured and typed representation of an agent run, including evidence units (the artifacts, such as user queries, retrieved documents, observations, tool outputs, memory items, generated tool parameters, intermediate claims, inter-agent messages, and final outputs); execution units (the occurrences, such as reasoning steps, retrieval calls, tool invocations, memory accesses, environment actions, and delegation steps); and the semantic, derivational, causal, and procedural relations among them.

Evidence tracing. A view of execution provenance that records support, contradiction, invalidation, and content derivation for claims, decisions, and actions. It excludes causal and procedural edges and adds *candidate* links for qualifying content-flow paths where no SUPPORT or INVALIDATE relation is recorded between the evidence and the claim. Candidate links indicate possible content contribution, not established support.

We represent execution provenance as a graph whose nodes are evidence and execution units and whose typed edges record their relations. This graph is the *process view* of an agent run; evidence tracing is its

support view, formalized as a projection in Appendix C. The two views are complementary rather than interchangeable. Different executions can share the same support view, so evidence tracing alone cannot reconstruct how an agent ran. Conversely, recording execution dependencies does not establish evidential support: a tool output can influence a decision without justifying it, and a false memory item can cause an action while supporting nothing. This distinction motivates reviewing evidence attribution alongside tool-use provenance, memory lineage, and failure diagnosis within a common framework.

Survey scope. This survey focuses on works that make LLM-agent behavior traceable beyond final outputs. We include methods, systems, benchmarks, and evaluation protocols when they record, represent, attribute, inspect, constrain, or diagnose the information and execution steps that shape agent behavior. We discuss RAG, memory, graph-based retrieval, tool-use safety, and observability not as standalone research areas, but as settings in which evidence tracing and execution provenance become necessary. The organizing principle is provenance-centered: what should be traced, how trace units are connected, when tracing occurs, how traces are represented, and which trust functions they support.

Section 1.1 states the selection protocol, the inclusion and exclusion criteria, and the resulting corpus. In summary, this survey makes the following contributions:

- We formulate evidence tracing and execution provenance as a unified process-level accountability framework for trustworthy LLM agents, and give a formal definition of the execution-provenance graph together with the projection that produces its evidence-support view, including what the projection discards and what it adds.
- We introduce a six-dimensional taxonomy covering trace sources, evidence and execution units, provenance relations, tracing granularity and timing, representation forms, and trust functions, with two cross-cutting attributes for provenance origin and trace assurance, and we give each of the nine provenance relations an operational definition that separates semantic support from causal and procedural influence.
- We review existing methods by research thread: provenance representation and evidence attribution (Section 3), tool-use provenance and runtime guardrails (Section 4), provenance-bearing memory (Section 5), and observability and failure diagnosis, which cut across all three and are assessed in Section 6.
- We map benchmarks, datasets, and metrics to provenance-related capabilities, showing how evaluation can move beyond final-answer correctness toward trace availability, evidence support, tool-use safety, memory lineage, observability, and recovery.
- We identify three gaps that recur across threads: limited integration of fine-grained provenance with multiple trust functions in the reviewed systems, no benchmark in our catalog that evaluates the complete provenance chain, and little reporting of tamper-evident or attested traces.
- We identify open challenges for provenance-aware LLM agents, including unified trace schemas, semantic provenance, memory and multi-agent provenance, provenance-aware safety, realistic execution-trace benchmarks, recovery-oriented evaluation, trace assurance and attested provenance, and privacy-aware audit infrastructure.

1.1 Survey Methodology

This is a *structured curated review* rather than an exhaustive systematic review; the protocol below defines the corpus and the limits of its coverage.

Search and screening. The corpus was assembled by *seeded snowballing* rather than by keyword search over bibliographic databases. We began from a seed set of foundational and canonical works spanning the areas in scope (classical provenance modeling and information-flow security, retrieval attribution, tool-use safety, memory-augmented agents, agent observability, and trace-based debugging), and expanded it by backward and forward citation following, both from each retained paper and from the benchmarks it evaluates on,

iterating until no newly encountered work satisfied the inclusion criteria below. Screening proceeded in two ordered stages, and a candidate had to pass both. *Stage 1 (title and abstract)*: retain the candidate if it plausibly satisfies at least one inclusion criterion below; the exclusions for general surveys and for work evaluated only by final-answer accuracy are applied here. *Stage 2 (full text)*: retain only if the work demonstrably records, relates, constrains, or evaluates an intermediate execution artifact, rather than merely noting that agents produce traces; the remaining exclusions are applied here. Screening was performed by the authors jointly and disagreements were resolved by discussion; we did not use a second independent coder, which is a limitation of the procedure.

Seed set. Because the reach of the closure is fixed by its seeds, we list all fifteen, so that the expansion can be re-run and its boundary checked.[2] Expansion ran backward and forward from each seed and from every work subsequently retained.

This procedure fixes the boundary of the corpus by the inclusion criteria and the snowballing closure, not by a query. Its reach therefore depends on the seed set, which is the principal reason we describe this review as curated rather than systematic; the limitations below follow from this choice.

Inclusion criteria. A work is included if it satisfies at least one of the following, each of which is a property of the artifact rather than a judgment about quality:

1. it *records* intermediate execution artifacts, that is, it defines or releases a trace format, schema, or log covering retrieval, tool calls, memory operations, observations, or inter-agent messages;
2. it *relates* such artifacts, that is, it introduces edges among them, including claim–evidence support, tool-argument dependency, memory lineage, or failure attribution;
3. it *constrains* execution using such artifacts at runtime, including taint tracking, information-flow control, permission enforcement, or policy checking;
4. it *evaluates* such artifacts, that is, it provides a benchmark, dataset, or metric whose ground truth concerns intermediate steps or evidence rather than only final answers; or
5. it supplies a *foundation* that the above build on directly, namely classical provenance modeling, information-flow security, and fact verification.

Exclusion criteria. We exclude work that improves agent capability without exposing process-level structure: prompting and reasoning methods evaluated only by final-answer accuracy; retrieval and re-ranking work that does not attribute claims to sources; agent architectures without released traces or trace schemas; and safety work operating only on model inputs and outputs rather than on execution artifacts. We also exclude general surveys of RAG, agents, or LLM safety except where they define trace or provenance structure, since our unit of analysis is the execution artifact rather than the task.

Resulting corpus. Starting from the fifteen seeds, the protocol yields the 71 works that constitute the reviewed corpus: 13 from 2023, 16 from 2024, 13 from 2025, and 24 from 2026, together with five earlier foundations in provenance modeling, information-flow security, and fact verification (World Wide Web Consortium, 2013; Thorne et al., 2018; Saltzer and Schroeder, 1975; Denning, 1976; Sabelfeld and Myers, 2003). The corpus mixes 51 papers published at peer-reviewed venues, 17 unrefereed preprints, and 3 standards or specifications; the preprint fraction follows directly from the topic's recency. Venue information was verified against the published record rather than taken from first posting, and a work is counted as refereed only when its acceptance at a peer-reviewed venue is publicly recorded; a submission under review counts as a preprint. Twenty-six of the cited works are benchmarks or datasets, cataloged in Table 8 as 25 rows: three rows name two closely related artifacts each, and one of those three, AgenTracer/Aegis, covers

[2]The seeds, by area. *Provenance modeling and information-flow security*: W3C PROV-DM (World Wide Web Consortium, 2013), Saltzer and Schroeder (Saltzer and Schroeder, 1975), the lattice model of secure information flow (Denning, 1976), and language-based information-flow security (Sabelfeld and Myers, 2003); *retrieval attribution*: FEVER (Thorne et al., 2018), ALCE (Gao et al., 2023), and FActScore (Min et al., 2023); *tool-use safety*: ToolEmu (Ruan et al., 2024) and AgentDojo (Debenedetti et al., 2024); *memory-augmented agents*: MemGPT (Packer et al., 2023) and Generative Agents (Park et al., 2023); *agent observability*: OpenTelemetry (OpenTelemetry, 2026) and AgentOps (Dong et al., 2024); and *trace-based debugging*: TRAIL (Deshpande et al., 2025) and MAST (Cemri et al., 2025).

two separately cited works. The per-benchmark coding in that catalog is ours, and the family-level coverage ratings of Section 6 are computed from it by the rule stated in Figure 5. Appendix D.2 states the annotation protocol behind these per-benchmark ratings and the memory-capability ratings of Table 6. By review thread, the citation counts are 14 works cited in the representation and attribution review of Section 3, 28 in the tool-use review of Section 4, 27 in the memory review of Section 5, and 42 in the benchmark and metric review of Section 6; the totals overlap because many works are cited in more than one thread.

How the main findings were established. Two of the survey's main findings concern coverage, and they rest on different evidence. The first, that the reviewed systems integrate fine-grained provenance representation with multiple trust functions only to a limited extent, is an observation drawn from Table 3 together with the system descriptions of Sections 3–5; because that table records each system's primary trust function only, it supports a statement about limited integration rather than an absolute negative. The second, that no benchmark in our catalog evaluates the complete provenance chain, is computed from Table 8 by a fixed rule and re-derived cell by cell in Appendix D.1, so each rating can be traced to a specific benchmark and column. Both findings concern the works reachable from the seeds, not the literature as a whole.

Publication status of the corpus. Because the area is young and much of it circulates as preprints, we record the publication status of every work. Every entry in Tables 3, 5, 6 and 8 that had not been peer reviewed at the time of writing is marked † in place, so that confidence can be weighed while reading the row rather than by consulting this section. The exposure is uneven: 6 of 27 rows in Table 3, 8 of the 20 systems named in Table 5, 4 of 11 rows in Table 6, and 3 of 25 rows in Table 8. Across the whole corpus, 17 of the 71 works are unrefereed preprints, one of them a non-archival workshop poster (AlSayyad et al., 2026), while 51 are peer-reviewed papers and 3 are standards or specifications.

Two consequences follow. First, the concentration matters more than the count: the marked rows cluster in runtime enforcement, information-flow auditing, and observability (AgentTrace (AlSayyad et al., 2026), Agent-Sentry (Sequeira et al., 2026), NeuroTaint (Cai et al., 2026), Fides (Costa et al., 2025), AgentOps (Dong et al., 2024)), which is where the survey identifies its sharpest gaps, so conclusions about those areas deserve more discount than conclusions about retrieval attribution, where the underlying work is refereed. Second, *no claim in this survey rests on a single unrefereed work*: each marked system is grouped with refereed work making the same structural point, such as PROV-AGENT (Souza et al., 2025) and CaMeL (Debenedetti et al., 2026), so discounting a marked row costs an instance rather than a claim. Section 7.6, whose supporting works (Ojewale et al., 2026; Nian et al., 2026; Zheng et al., 2026) are also preprints, states a gap rather than a result.

Limitations of the protocol. Four limitations follow from the above and bound how the survey should be read. Enumerating the seed set makes the closure re-runnable, but not reproducible in the sense a database query would be, and we report no screening yield counts, because screening was concurrent with reading rather than logged prospectively. The corpus is curated rather than exhaustive, so absence from our tables is not evidence that no such work exists. Screening and coding were performed by the authors without an independent second coder, so the coverage judgments carry unquantified subjectivity and we report no inter-coder agreement. Finally, the area is moving quickly enough that a substantial part of the corpus is not yet peer reviewed.

1.2 Relation to Existing Surveys

Most of the threads reviewed here have been surveyed before, and the contribution of this survey is not to cover works those surveys miss but to organize them by a different unit of analysis. Existing surveys take the *task* as the unit (how well does attribution work, how well does memory work) and are therefore organized by method family within one thread. We take the *execution artifact* as the unit, which cuts across threads, and the consequence is that questions invisible within a thread become statable across threads.

The closest work by subject is the survey of attribution, citation, and quotation in evidence-based text generation (Schreieder et al., 2026), which covers claim–source linking in far more depth than Section 3 does, but treats the source as given rather than as something an agent produced through a tool call or wrote into memory; in our terms it studies SUPPORT with the procedural half of the graph out of scope. The survey of memory mechanisms in LLM agents (Zhang et al., 2025) is similarly the better reference on

memory architectures, and organizes them by read/write/management operation, whereas Section 5 asks the orthogonal question of what a memory item's *lineage* is once written. Surveys of agent hallucination (Lin et al., 2025) classify failure by output symptom; the provenance view instead asks which recorded relation is missing when a symptom appears, which is what makes DERIVE without SUPPORT, the attributable hallucination, a distinct and separately measurable case. Retrieval-structure surveys such as that on graph-based RAG (Peng et al., 2025) share our use of typed graphs, but the graph is over the *corpus*, built before the run, while ours is over the *execution*, built by it.

Reading these surveys through a common provenance view brings out three points that none of them shows on its own. First, the same object appears in several of them under different names: a retrieved passage is evidence in attribution work, context in RAG work, and a taint source in security work. A common vocabulary makes it easier to see that they are the same node with different edges attached. Second, the gaps are *shared* rather than thread-specific: none of these surveys treats the origin of a recorded relation as a dimension, which is why Section 2.1 has no counterpart in them. Third, evaluation is where the fragmentation costs most, since a benchmark built for one thread rarely supplies ground truth for another, as Figure 5 shows. We are not aware of a survey of agent observability or of execution-trace infrastructure as such, which is itself consistent with the maturity gap Section 7.6 reports. On corpus membership: the attribution and memory surveys are counted among the 71, retained under the exclusion criteria's exception for surveys that define trace or provenance structure; the agent-hallucination and graph-RAG surveys are cited for positioning only, so the counts of Section 1.1 are unchanged.

Structure of the Survey. The survey follows a single through-line: turning raw agent traces into accountable provenance. Section 2 first develops a taxonomy of trace sources, evidence and execution units, provenance relations, tracing granularity and timing, representation forms, and trust functions. We then proceed through four steps: Section 3 reviews provenance representations, ranging from structured logs to execution graphs, evidence graphs, claim-support graphs, and runtime provenance structures; Sections 4 and 5 examine two settings where provenance failures are especially consequential, namely tool use and memory; Section 6 maps benchmarks and metrics to provenance-related capabilities; and Section 7 discusses open challenges and future directions before Section 8 concludes the survey.

The section allocation follows one criterion: a setting earns a dedicated section when its provenance failure has a *distinct mechanism* and there is a reviewable body of released artifacts that act on it. Tool use qualifies on both counts: a contaminated tool argument propagates through execution rather than through content, and Section 4 reviews 28 works that address it. So does memory, where provenance must survive writes, updates and retrievals across sessions, reviewed over 27 works. Multi-agent and environment provenance are first-class in the taxonomy, as trace sources and unit types. We treat them as cross-cutting concerns, discussing them where they intersect with tool use, memory lineage, responsibility attribution, and evaluation; the corpus also contains far fewer works acting specifically on them. They are accordingly developed where they bear: responsibility attribution in Section 6.3, open problems in Section 7.3, and requirements in Appendix E.3.

2 Taxonomy of Evidence Tracing and Execution Provenance

Having defined agent traces, evidence tracing, and execution provenance, this section introduces the taxonomy used throughout the survey. The goal is not to review every system in detail, but to provide a compact vocabulary for comparing provenance-aware agent work. We organize the space along six dimensions: trace sources, evidence and execution units, provenance relations, tracing granularity and timing, representation forms, and trust functions. Figure 2 illustrates the relationships among five of these dimensions. Representation forms describe how provenance is encoded; Table 1 summarizes all six dimensions and provides a reference for the comparisons throughout the survey.

The taxonomy draws on two existing traditions. Provenance models such as W3C PROV-DM describe how entities, activities, and agents participate in the production of artifacts through relations such as usage, generation, derivation, revision, and invalidation (World Wide Web Consortium, 2013). Distributed observability frameworks such as OpenTelemetry model executions as traces and spans, which is useful for describing multi-step agent runs (OpenTelemetry, 2026). LLM agents, however, add semantic and procedural objects that these traditions do not fully capture: retrieved passages, generated claims, tool-call

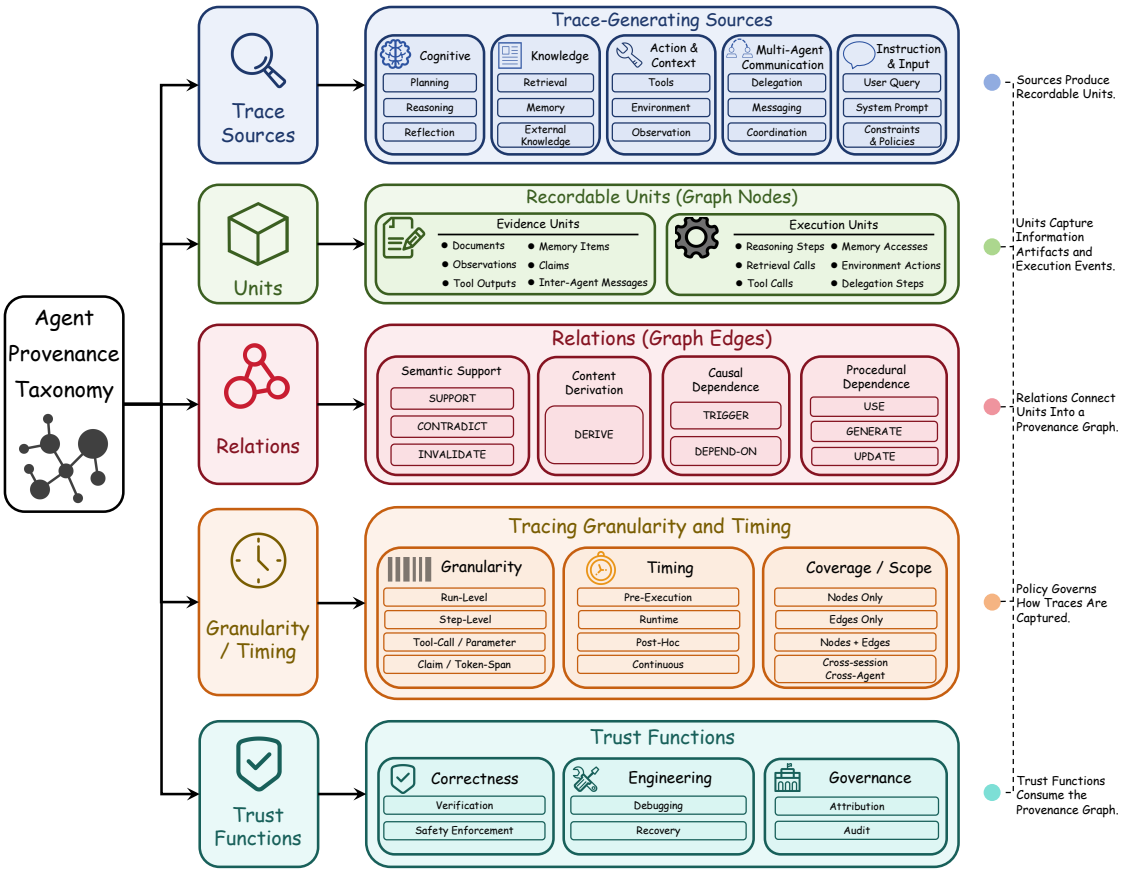


Figure 2: Overview of the agent provenance taxonomy, showing trace sources, evidence and execution units, provenance relations, tracing granularity and timing, and trust functions. Table 1 summarizes all six dimensions, including representation forms.

rationales, memory items, natural-language observations, external state changes, and inter-agent messages. Agent-oriented observability systems such as AgentOps and AgentTrace further motivate structured records of both execution artifacts and context (Dong et al., 2024; AlSayyad et al., 2026). We therefore treat classical provenance and observability as foundations, and specialize them to the execution structure of LLM agents.

2.1 Taxonomy Dimensions

Trace sources. Trace sources are the components and execution boundaries from which provenance-relevant artifacts are recorded. In LLM agents, they include instructions and inputs, reasoning, retrieval, tool use, memory, environment interaction, and multi-agent communication. Instruction and input traces record user requests, system prompts, constraints, and policies; reasoning traces record stated plans and rationales for action choices; retrieval traces provide candidate evidence; tool traces record calls, arguments, outputs, permissions, and errors, including interactions across MCP server/host boundaries; memory traces record reads, writes, and updates across sessions; environment traces record observations and state-changing actions; and multi-agent traces capture delegation, messaging, and coordination (Yao et al., 2023; Shinn et al., 2023; Qin et al., 2024; Zhou et al., 2024; Yao et al., 2025; Wu et al., 2024; Li et al., 2023). Later sections examine tool and memory sources in depth; here we identify the sources covered by the taxonomy.

Evidence and execution units. Trace sources produce two broad kinds of units. *Evidence units* are artifacts whose content can support, contradict, invalidate, or contextualize claims and actions, such as user queries, system prompts, documents, passages, observations, tool outputs, memory items, generated

Table 1: Six-dimensional taxonomy of evidence tracing and execution provenance in LLM agents. Unit types and representative works are illustrative rather than exhaustive.

Dimension	Categories	Description	Representative Work
Trace sources	Instructions and inputs, reasoning, retrieval, tool use, MCP server/host boundaries, memory, environment, multi-agent communication	Components and execution boundaries that generate provenance-relevant artifacts during agent execution.	ReAct, Reflexion, ToolLLM, WebArena, AutoGen, AgentTrace, MCP-SafetyBench (Yao et al., 2023; Shinn et al., 2023; Qin et al., 2024; Zhou et al., 2024; Wu et al., 2024; AlSayyad et al., 2026; Zong et al., 2026)
Evidence and execution units	*Evidence units (artifacts):* User queries, system prompts, documents, observations, tool outputs, memory items, generated tool parameters, policies, tool manifests, inter-agent messages, intermediate claims, final answers	Content-bearing artifacts generated or consumed during a run, whose content can be compared across units.	ALCE, FActScore, FEVER, RAGChecker (Gao et al., 2023; Min et al., 2023; Thorne et al., 2018; Ru et al., 2024)
	Execution units (occurrences): Reasoning steps, retrieval calls, tool calls, memory accesses, environment actions, delegation steps	Occurrences identified by an actor and a time interval, recording what the agent did.	AgentOps, AgentTrace, TRAIL, AgenTracer, Aegis (Dong et al., 2024; AlSayyad et al., 2026; Deshpande et al., 2025; Zhang et al., 2026; Kong et al., 2026)
Provenance relations	*Semantic support:* SUPPORT, CONTRADICT, INVALIDATE	SUPPORT: evidence warrants a claim or action precondition; CONTRADICT: two contents conflict. Neither requires source access; INVALIDATE also needs authority and temporal context.	ALCE, FActScore, SourceCheckup, RAGChecker (Gao et al., 2023; Min et al., 2023; Wu et al., 2025; Ru et al., 2024)
	Content derivation: DERIVE	The target content was produced by transforming the source content. Derivation establishes content lineage, not the correctness of the transformation or semantic support.	W3C PROV-DM, PROV-AGENT (World Wide Web Consortium, 2013; Souza et al., 2025)
	Causal dependence: TRIGGER, DEPEND-ON	TRIGGER: what caused a step; DEPEND-ON: what shaped its arguments or behavior. DEPEND-ON is always inferred; TRIGGER can be logged directly for event-driven dispatch.	AgenTracer, Fides, NeuroTaint (Zhang et al., 2026; Costa et al., 2025; Cai et al., 2026)
	Procedural dependence: USE, GENERATE, UPDATE	Records artifact use, generation, and version succession. UPDATE may be instrumented or inferred; USE over-approximates DEPEND-ON only for dependencies on artifacts.	W3C PROV-DM, Agent-Sentry (World Wide Web Consortium, 2013; Sequeira et al., 2026)
Tracing granularity and timing	*Granularity:* Run-level, step-level, tool-call-level, parameter-level, claim-level, token/span-level	Determines the level of detail at which units and relations are recorded.	FActScore, SourceCheckup, Agent-Sentry (Min et al., 2023; Wu et al., 2025; Sequeira et al., 2026)
	Timing: Pre-execution, runtime, post-hoc, continuous	Determines when trace collection, verification, enforcement, or diagnosis occurs.	AgentSpec, Agent-Sentry, TRAIL (Wang et al., 2026; Sequeira et al., 2026; Deshpande et al., 2025)
Representation forms	Structured logs, execution graphs, evidence and claim-support graphs, provenance graphs, runtime state	Forms used to encode execution records and relations for inspection, comparison, enforcement, or evaluation.	W3C PROV-DM, AgentOps, AgentTrace, PaperTrail (World Wide Web Consortium, 2013; Dong et al., 2024; AlSayyad et al., 2026; Martin-Boyle et al., 2026)
Trust functions	Verification, attribution, debugging, safety enforcement, audit, recovery	Downstream purposes served by evidence tracing and execution provenance.	RAGAS, AgentOps, AgentTrace, LADYBUG, AgenTracer, OpenAgentSafety (Es et al., 2024; Dong et al., 2024; AlSayyad et al., 2026; Rorseth et al., 2025; Zhang et al., 2026; Vijayvargiya et al., 2026)

tool parameters, policies, inter-agent messages, intermediate claims, and final answers. *Execution units* are occurrences that record what the agent did, including reasoning steps, retrieval calls, tool invocations, memory

Table 2: Two cross-cutting attributes of agent provenance: provenance origin records how a provenance edge came to exist, and trace assurance records how a trace is protected against modification.

Attribute	Values	Description	Motivating Work
Provenance origin	INSTRUMENTED, INFERRED, DECLARED, DERIVED	How a provenance edge came to exist: emitted by the runtime, judged post hoc by a model or human, or self-reported by the agent; DERIVED is reserved for edges computed from a graph rather than observed.	AgentOps, PROV-AGENT, AgenTracer, SourceCheckup (Dong et al., 2024; Souza et al., 2025; Zhang et al., 2026; Wu et al., 2025)
Trace assurance	Logged, tamper-evident, attested	Whether the record is protected against post-hoc modification. The reviewed descriptions do not establish tamper-evident or attested trace guarantees.	Audit trails, auditable agents, checkpoint integrity (Ojewale et al., 2026; Nian et al., 2026; Zheng et al., 2026)

reads or writes, environment actions, and delegation steps. Membership is decided by a single test, so that the same object is always classified the same way: a unit is an *evidence unit* if it is an artifact carrying a content field that can be compared with the content of another unit, and an *execution unit* if it is an occurrence carrying an actor and a time interval, so that its identity is *that it happened* rather than *what it says*. Under this test the two roles are disjoint at graph level, following the entity/activity split of W3C PROV-DM (World Wide Web Consortium, 2013). The familiar intuition that a tool output is "both" the result of an execution step and evidence for a later claim reflects the granularity of a flat log, in which one record carries the call, its arguments, and its result; the graph representation splits that record into a tool call (execution unit) and a tool output (evidence unit) joined by a GENERATE edge. What varies across edges is therefore not a unit's role but which part of it an edge reads: a semantic edge reads its content, a procedural edge reads its occurrence. Appendix B.1 gives the complete type assignment. Provenance-aware agents still need to connect semantic support with procedural execution rather than recording only one side of the trace (Gao et al., 2023; Min et al., 2023; Thorne et al., 2018; Ru et al., 2024; Dong et al., 2024; AlSayyad et al., 2026).

Provenance relations. Relations turn raw trace units into provenance. We retain PROV-compatible base relations (USE, GENERATE, and DERIVE) and specialize four further relations that PROV expresses in a weaker form: DEPEND-ON and TRIGGER refine PROV usage and communication into counterfactual claims, UPDATE refines revision, and INVALIDATE refines PROV invalidation from ending an entity's existence to marking a persisting record epistemically unusable. Only SUPPORT and CONTRADICT have no PROV counterpart at all, since they compare content rather than record how artifacts were produced.

The nine fall into four families, distinguished by *what kind of evidence establishes them*: *semantic support* (SUPPORT, CONTRADICT, INVALIDATE) is decided by reading the contents of both endpoints, and is therefore agent-independent, although INVALIDATE additionally requires authority and temporal context; *content derivation* (DERIVE) asks whether the target's content is a transformation of the source's; *causal dependence* (TRIGGER, DEPEND-ON) is decided counterfactually, by ablation and replay, unless the harness itself dispatches the step on an event, in which case the counterfactual holds by construction and the edge can be logged directly; it says nothing about whether content is true; and *procedural dependence* (USE, GENERATE, UPDATE) is read directly from what the harness emitted, except that UPDATE is inferred when the memory store does not version its keys. Keeping the families apart is what lets the taxonomy record that a tool output changed a decision without justifying it, or that a false memory item drove an action while supporting nothing.

Temporal succession is deliberately *not* a relation: ordering is carried by node timestamps and never by an edge, so temporal adjacency alone does not establish dependence; without an explicit relation, the graph records no dependency between the steps. Appendix B gives each relation a signature over node types, a direction, temporal semantics, a necessary and sufficient condition, and worked examples with the test that separates it from the relation it is most often confused with; Appendix B.3 works through the boundary cases on which annotators most often disagree.

A relation is admitted only if it passes two tests: *necessity*, meaning that some trust-function query of this section cannot be answered without it, and *distinctness*, meaning that it carries a decision test that no other relation's test subsumes. Appendix B.6 applies both tests relation by relation and states what is lost under three alternative decompositions we considered and rejected, including PROV-DM unextended. It also explains why we prefer explicit relation labels to a single influence edge with a kind attribute, an encoding that carries the same information but makes the distinctions easier to drop. The set is minimal only relative to the six trust functions and the corpus surveyed here, and Appendix B.6 gives the test a tenth relation would have to pass.

Provenance origin and assurance. The six dimensions describe *what* is traced and *why*. Cutting across all of them are two attributes that describe *how a provenance record came to exist*, and that determine how much weight it can bear. We treat them as edge and node attributes rather than as a seventh dimension, because they qualify every relation in the taxonomy rather than partitioning the space. Table 2 summarizes the values of each attribute and the work that motivates it; the rest of this subsection explains both.

Origin distinguishes three sources of a provenance edge recorded from an execution; a fourth value, DERIVED, is reserved for edges computed from a graph rather than observed, and is used only by the projection of Appendix C. INSTRUMENTED edges are emitted by the agent harness or runtime and are reproducible from the log alone; USE and GENERATE are always of this kind, and UPDATE and TRIGGER are Instrumented when the memory store versions its keys or the harness dispatches on an event. INFERRED edges are assigned post hoc by a model or a human annotator; DEPEND-ON falls here, since counterfactual dependence requires replay, ablation, fault injection, or information-flow tracking rather than reading a log (Zhang et al., 2026; Costa et al., 2025; Cai et al., 2026); SUPPORT, CONTRADICT, and INVALIDATE are never directly observable either, and are INFERRED when an annotator or judge assigns them and DECLARED when the agent asserts them itself. DECLARED edges are agent self-reports, such as a generated citation, and are the weakest: a declared citation need not support the claim it annotates, which is precisely the failure that source-checking work measures (Wu et al., 2025). A provenance graph is therefore heterogeneous in reliability. Its procedural skeleton can be logged, but its semantic layer is a judgment, and evaluation must treat the two differently rather than reporting a single accuracy number over a mixed graph. The distinction matters for two downstream uses that appear later in this survey. A *responsibility* claim (that a particular agent or step is answerable for an outcome, as multi-agent attribution work asserts (Cemri et al., 2025; Zhang et al., 2026; Kong et al., 2026)) is a claim about counterfactual dependence, so it inherits the origin of DEPEND-ON and TRIGGER and is INFERRED: a log records that a sub-agent produced a message, never that the outcome would have differed without it. And a *reasoning trace* is an artifact the model emitted, not an observation of the computation that produced the output, so an edge read off a stated rationale is DECLARED and does not, by itself, establish faithful dependence; its trace assurance is a separate attribute: a model may cite a passage it did not use, or act on a memory item it never mentions. Both uses remain legitimate, provided such claims are reported as inferred or declared rather than as recorded, and validated by replay or annotation rather than by pointing at the log.

Assurance distinguishes whether a trace is merely written to a log, or additionally protected against modification after the fact. This is the least developed attribute in the works reviewed here. The descriptions of the observability and provenance systems in Table 3 do not report tamper-evidence, signing, or attestation for their traces (Dong et al., 2024; AlSayyad et al., 2026; Souza et al., 2025), so a trace is trustworthy only to the extent that the party holding it is. Recent work has begun to argue for audit-trail infrastructure and auditable agent design (Ojewale et al., 2026; Nian et al., 2026), and work on checkpoint-restore integrity shows that agent state itself can be attacked (Zheng et al., 2026), but the reviewed descriptions do not establish tamper-evident or attested trace guarantees for the systems we survey. This matters for audit, where the reader of a trace and its producer are often not the same party. We return to it as an open problem in Section 7, and Appendix B.4 records the available origins for each relation.

Tracing granularity and timing. Granularity specifies how fine-grained the trace is. Run-level traces support coarse audit; step-level traces support debugging; tool-call and parameter-level traces support safety enforcement; claim-level traces support factual attribution; and token- or span-level traces provide the finest but most expensive view. Timing specifies when traces are collected or used: pre-execution checks can block

unauthorized actions, runtime tracing supports policy enforcement and information-flow control, post-hoc tracing supports audit and diagnosis, and continuous tracing supports persistent memory and long-horizon workflows (Min et al., 2023; Wu et al., 2025; Costa et al., 2025; Cai et al., 2026; Wang et al., 2026; Sequeira et al., 2026; Deshpande et al., 2025). The appropriate choice depends on the risk and persistence of the workflow: low-risk RAG may require claim-level post-hoc attribution, whereas high-impact tool use may require runtime parameter-level provenance.

Representation forms. Representation forms determine how traces are encoded. Structured logs preserve chronological events and metadata, but often leave support, contradiction, and influence implicit. Execution graphs make procedural dependencies explicit; evidence and claim-support graphs emphasize semantic grounding; provenance graphs combine evidence and execution units through typed edges; and runtime state representations support online checking and enforcement. We keep this dimension concise here because Section 3 reviews representation choices in detail.

Trust functions. The purpose of tracing is not to store more logs, but to support trust functions. *Verification* asks whether claims or actions are supported by evidence; *attribution* identifies which evidence, tool output, memory item, or message contributed to a claim or action; *debugging* localizes failures within an execution; *safety enforcement* prevents untrusted or unauthorized influence from reaching sensitive actions; *audit* reconstructs executions for review and accountability; and *recovery* uses provenance to retry, compensate, roll back, or repair affected states (Gao et al., 2023; Es et al., 2024; Deshpande et al., 2025; Rorseth et al., 2025; Debenedetti et al., 2026; Costa et al., 2025; Sequeira et al., 2026). These functions explain why the taxonomy includes both evidence-support links and execution-dependency links.

2.2 Design Tensions

The taxonomy exposes four design tensions that guide the rest of the survey. First, chronological logging and semantic provenance are not the same: a log can record what happened, but it may not show whether evidence supported a claim or whether memory influenced a tool call. Second, component-level tracing does not guarantee end-to-end accountability, because failures may propagate across retrieval, tools, memory, environment states, and other agents. Third, static schemas help standardize trace objects and relations, but runtime safety requires online source tracking, policy checking, and dependency-aware enforcement. Fourth, trace completeness must be balanced against deployability, since fine-grained provenance improves verification, debugging, audit, and recovery but also increases storage cost, privacy exposure, annotation burden, and system complexity.

The taxonomy is used throughout the paper as a compact comparison vocabulary. Sections 3–6 return to these tensions when reviewing representation methods, tool-use provenance, memory provenance, and evaluation. Before turning to these topics, Table 3 positions representative systems within the taxonomy to expose coverage gaps across trace sources, granularity, timing, representation, and trust functions. Appendix A provides an explicit mapping to W3C PROV-DM.

2.3 Two Decisions the Taxonomy Changes

To show what the taxonomy adds beyond enumerating trace artifacts, we work through two decisions where the two approaches diverge, and in both the divergence comes from the same source: the taxonomy separates *which relation* is at stake from *how a record of it can be obtained*, so it constrains what a design or an evaluation can deliver before either is built.

Decision 1: what to instrument in an auditable tool-using agent. Consider an agent with tools and persistent memory, deployed where an auditor must later reconstruct why an action was taken. Enumerating trace artifacts yields the familiar answer: record every artifact in Section 2.1 at the finest affordable granularity, and if the budget allows, go from step level to token level. The taxonomy gives a different and partly opposite answer, because granularity and origin are independent axes. USE and GENERATE are INSTRUMENTED: they are already reproducible from an ordinary call log, and refining that log buys no new relation. DEPEND-ON is INFERRED only, so *no* logging granularity yields it: counterfactual dependence requires replay, ablation,

or information-flow tracking, which is a runtime capability rather than a logging setting. And SUPPORT is never observable at any granularity, so claim-level audit requires an annotation protocol with a fixed warrant threshold and reported agreement, not a finer trace. The recommendation is therefore to spend the budget on *replay hooks and an annotation protocol*, and to keep logging at step level, whereas the enumeration answer spends it on token-level logging, which cannot produce either missing layer. The taxonomy also fixes what the audit may claim: without replay the record supports attribution and reconstruction but not counterfactual causal claims, and saying so in advance is more useful than discovering it during an audit.

Decision 2: how to evaluate a memory-poisoning defense. Now consider evaluating a defense against poisoned memory. Enumeration suggests the obvious protocol: use a memory benchmark and check whether the defense preserves task success. The taxonomy identifies the failure differently. A poisoned item that drives an action while justifying nothing is a TRIGGER edge with no SUPPORT edge, causal dependence without semantic support, so the quantity to measure is whether that TRIGGER is detected, which is established counterfactually rather than by answer comparison. Two consequences follow that the enumeration protocol misses. First, the memory family of Figure 5 is rated **L** on safety attacks: it supplies no adversarial ground truth, so a memory benchmark cannot measure the defense's actual target, and the relevant labels (source-to-sink influence, that is, causal dependence of the kind TRIGGER and DEPEND-ON record) sit in the taint benchmarks of the provenance/security family instead. Second, task success and defense quality *can come apart*. As an illustrative scenario, assume that poisoned items make up a small fraction of the memory store, that they are retrieved only for the queries they target, and that the defense does not also filter benign items. A defense that correctly blocks a poisoned item then removes a TRIGGER that never carried SUPPORT; under these assumptions, answers to unaffected queries need not change, and aggregate accuracy may move only slightly even when the defense is working. The taxonomy does not guarantee this outcome: a defense that also removes benign items, or an attack aimed at frequent queries, can change accuracy appreciably in either direction. What the scenario shows is that answer accuracy is not a reliable proxy for defense quality, so dependency coverage should be reported separately from answer accuracy, and the reason lies in the relation type rather than in a list of artifacts.

2.4 Positioning Existing Systems in the Taxonomy

Table 3 positions representative agent, attribution, safety, memory, and observability systems within the taxonomy. The table is not intended as an exhaustive benchmark, but as a diagnostic map of where current work concentrates and where coverage remains incomplete. Three patterns emerge. First, *trace-generating systems* such as ReAct, AutoGen, MemGPT, and Generative Agents produce rich execution artifacts, but usually do not implement trust functions such as verification, attribution, enforcement, audit, or recovery on their own. They provide substrates on which provenance mechanisms can be built. Second, *attribution and verification systems* such as ALCE, FActScore, SourceCheckup, and RAGChecker provide claim-level support analysis, but mostly operate over retrieval evidence and final answers rather than tools, memory, environment actions, or multi-agent communication. Third, *safety and enforcement systems* such as CaMeL, Fides, and Agent-Sentry operate at runtime over tool calls, parameters, permissions, or information-flow labels, and NeuroTaint audits the same flows offline, but often lack semantic claim-evidence attribution.

This mapping shows why evidence tracing and execution provenance should be treated as a unified layer rather than as separate literatures. Existing systems usually cover one or two of these capabilities well: they may record traces, verify claims, enforce tool-use policies, audit executions, or diagnose failures. Within the corpus surveyed here, however, these capabilities are integrated only to a limited extent: few lines of work combine diverse trace sources, fine-grained units, runtime timing, explicit provenance representation, and multiple trust functions. This gap motivates the rest of the survey, which reviews representation, attribution, tool-use provenance, memory provenance, observability, safety, and evaluation as connected parts of the same provenance problem.

Table 3: Representative systems mapped to the taxonomy. Rows are grouped by each system's primary trust function; other functions a system supports are not recorded. † marks a work that had not been peer reviewed at the time of writing; for an entry naming two works, at least one of the two.

Primary trust	System	Trace source(s)	Granularity	Timing	Representation
Substrate	ReAct (Yao et al., 2023)	Reasoning, tool, env	Step	Runtime	Reasoning–action log
	Generative Agents (Park et al., 2023)	Memory, env	Step	Continuous	Memory stream
	Toolformer (Schick et al., 2023)	Tool	Tool-call	Pre-exec.	—
	ToolLLM (Qin et al., 2024)	Tool	Tool-call, param.	Runtime	API-call path
	AutoGen (Wu et al., 2024)	Multi-agent, tool	Step	Runtime	Message log
	MemGPT† (Packer et al., 2023)	Memory	Step	Continuous	Tiered memory
	A-MEM (Xu et al., 2025)	Memory	Step	Continuous	Linked memory
Attribution / Verification	ALCE (Gao et al., 2023)	Retrieval	Claim	Post-hoc	Citation links
	FActScore (Min et al., 2023)	Retrieval	Atomic claim	Post-hoc	Claim–evidence
	SourceCheckup (Wu et al., 2025)	Retrieval	Claim	Post-hoc	Source–support
	RAGChecker (Ru et al., 2024)	Retrieval	Claim	Post-hoc	Claim–evidence
	RAGAS (Es et al., 2024)	Retrieval	Claim	Post-hoc	Claim–context
Safety	CaMeL (Debenedetti et al., 2026)	Tool	Parameter	Runtime	Control/data flow
	Fides† (Costa et al., 2025)	Tool, memory	Parameter	Runtime	IFC labels
	NeuroTaint† (Cai et al., 2026)	Tool, memory	Param. /token	Post-hoc	Taint labels
	AgentSpec (Wang et al., 2026)	Tool, env	Tool-call	Pre /Runtime	Policy rules
	AgentBound (Bühler et al., 2026)	Tool	Tool-call	Pre /Runtime	Permissions
	Agent-Sentry† (Sequeira et al., 2026)	Tool	Parameter	Runtime	Provenance graph
Safety (eval)	ToolEmu (Ruan et al., 2024)	Tool	Tool-call	Runtime	Action log
	InjecAgent (Zhan et al., 2024)	Tool, env	Tool-call	Runtime	—
	AgentDojo (Debenedetti et al., 2024)	Tool, env	Tool-call	Runtime	—
	τ-bench (Yao et al., 2025)	Tool, env	Tool-call	Post-hoc	State log
Recovery	Reflexion (Shinn et al., 2023)	Reasoning, memory	Step	Continuous	Episodic memory
Debugging / Audit	TRAIL† (Deshpande et al., 2025)	All sources	Step	Post-hoc	Annotated trace
	MAST (Cemri et al., 2025)	Multi-agent	Step	Post-hoc	Failure taxonomy
	WebArena (Zhou et al., 2024)	Env, tool	Step	Post-hoc	Action/state log
	AgentOps / AgentTrace† (Dong et al., 2024; AlSayyad et al., 2026)	All sources	Step	Post-hoc/Cont.	Structured log

3 Provenance Representation and Evidence Attribution

The previous section identifies what should be traced; this section asks how traces should be represented. We organize provenance representation into three layers. *Structured logs* record typed execution events. *Execution graphs* connect instructions, retrievals, tool calls, memory operations, observations, claims, and actions through dependency relations. *Evidence and claim-support graphs* further specify whether evidence units support, contradict, or fail to support generated claims. This layered view is central to the rest of the survey: logs make executions reconstructable, graphs make dependencies inspectable, and claim-support links make grounding and accountability measurable. Figure 3 provides a running example. It shows how evidence acquisition, claim construction, tool execution, memory update, and recovery can be represented as a typed provenance graph rather than a flat chronological transcript.

3.1 From Logs to Execution Graphs

A natural starting point is the structured execution log. Instead of storing free-form transcripts alone, logs record typed events such as user requests, model outputs, retrieval calls, tool invocations, tool outputs, memory reads and writes, environment observations, inter-agent messages, and final responses. W3C PROV-

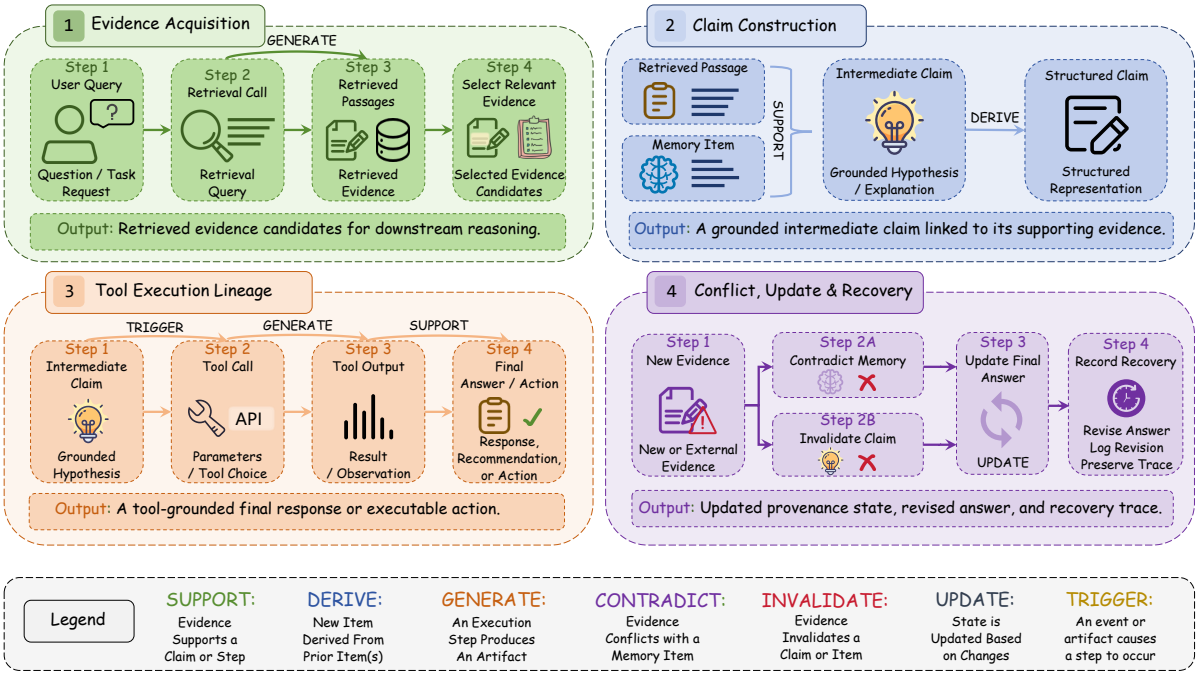


Figure 3: Running example of a provenance graph for an LLM-agent execution, linking user queries, retrieved passages, intermediate claims, tool calls and outputs, memory items, new evidence, and final actions through typed relations. For legibility, the panels show seven of the nine relations and omit USE and DEPEND-ON; Table 15 lists every edge of the example.

DM provides a general vocabulary of entities, activities, agents, and derivation relations (World Wide Web Consortium, 2013). PROV-AGENT adapts this vocabulary to agentic workflows by modeling prompts, responses, decisions, tool interactions, and workflow context (Souza et al., 2025). AgentOps emphasizes relationships among artifacts across the agent lifecycle (Dong et al., 2024), while AgentTrace organizes execution records into operational, cognitive, and contextual logs (AlSayyad et al., 2026).

Logs provide chronological observability, but dependency analysis requires graph structure. A log can show that a retrieval, tool call, and memory write occurred; it may not show which retrieved passage supported a claim, which tool output changed the plan, or which memory item influenced an action. Execution graphs address this gap by representing execution artifacts as nodes and their influence relations as typed edges. TRAIL demonstrates how annotated trajectories can localize failures to particular steps, components, or interactions (Deshpande et al., 2025). AgenTracer further uses counterfactual replay and fault injection to attribute failures to responsible agents and steps (Zhang et al., 2026), whereas Aegis uses positive–negative trajectory pairs to identify faulty agents and error modes (Kong et al., 2026). These systems differ in method, but share one contrast: logs answer *what happened*; execution graphs answer *what depended on what*.

3.2 Evidence and Claim-Support Attribution

Evidence attribution adds a semantic layer on top of execution dependencies. The key issue is whether a generated claim is actually supported by the evidence it cites or depends on. Existing work motivates this distinction at different granularities: citation-based evaluation links generated answers to sources (Gao et al., 2023); atomic-fact evaluation checks fine-grained factual support (Min et al., 2023); source-checking work shows that a cited source may be relevant without supporting the specific claim (Wu et al., 2025); and claim–evidence mapping explicitly distinguishes supported claims, unsupported claims, and omitted evidence (Martin-Boyle et al., 2026).

Table 4: Main provenance representation forms for LLM agents.

Form	What it captures	Main role
Structured logs	Typed execution events such as requests, retrieval calls, tool calls, memory operations, observations, and outputs.	Execution observability and reconstruction.
Execution graphs	Dependencies among instructions, evidence, tool calls, memory items, claims, and actions.	Influence analysis for debugging, audit, and recovery.
Evidence graphs	Links among evidence units, citations, atomic claims, and generated answers.	Separation of citation presence, relevance, support, contradiction, and omission.
Static schemas	Object types, relation types, trust labels, and required provenance fields.	A reusable vocabulary for trace representation and comparison.
Runtime provenance	Concrete source-to-sink, tool, memory, and action dependencies during execution.	Runtime monitoring, enforcement, recovery, and information-flow analysis.

For provenance representation, these correspond to distinct relationships rather than a single citation relation. A source may be CITED, asserted by the agent as supporting, without being SUPPORTED, that is, without an adjudicated support link; a relevant passage may be available but OMITTED from the answer; a tool output may CONTRADICT a memory item; and new evidence may INVALIDATE an earlier belief. CITED and OMITTED are not additional taxonomy relations: they are predicates derived from the nine, and Appendix B.5 defines them in those terms. Evidence graphs therefore distinguish evidence availability from evidence use. In LLM agents, evidence-bearing units include not only retrieved documents, but also tool outputs, memory items, environment observations, policies, and inter-agent messages.

3.3 Representation Trade-offs

Static schemas and runtime provenance play complementary roles in provenance representation. A static schema defines object types, relation types, trust labels, and required fields, including evidence units, tool calls, memory items, claims, actions, and relations such as SUPPORT, DEPEND-ON, CONTRADICT, UPDATE, and INVALIDATE. Runtime provenance instantiates this schema during a concrete execution, recording the actual dependencies among sources, tool calls, memory operations, claims, and actions. Observability systems emphasize runtime trace capture and inspection (Dong et al., 2024; AlSayyad et al., 2026). Safety-oriented systems additionally use runtime provenance for enforcement (Sequeira et al., 2026), information-flow control (Costa et al., 2025), and source-to-sink influence analysis (Cai et al., 2026).

The central trade-off concerns granularity. Fine-grained provenance enables claim verification, failure localization, rollback, audit, and policy enforcement, but increases storage cost, logging complexity, privacy exposure, and annotation burden. Coarse-grained provenance is easier to collect, but may miss the evidence that supported a claim, the memory item that caused an error, or the tool argument that crossed a trust boundary. The appropriate granularity therefore depends on the trust function: claim-level links may suffice for low-risk RAG, tool-using agents require parameter-level provenance, and long-term personalized agents require memory lineage, invalidation, and recovery support. Table 4 summarizes the main representation forms discussed in this section.

4 Execution Provenance in Tool-Using Agents

Tool use is the *outward* axis of agent provenance: it is where an LLM agent moves from generating text to interacting with APIs, databases, code interpreters, browsers, filesystems, enterprise systems, and other external environments. A tool call may retrieve private data, modify external state, send messages, execute code, or trigger irreversible actions. Trustworthy tool use therefore requires tracing why a tool was selected, where its arguments came from, whether its output was reliable, and how the result shaped later decisions. This section reviews five questions central to tool-use provenance: what to trace in tool execution, how external content contaminates tool use, how unsafe influence can be constrained, how action boundaries are enforced, and how traces support verification and recovery.

4.1 Tool-Call, Argument, and Output Provenance

Tool-call provenance records the path from tool selection to downstream influence: which tool was selected, which arguments were passed, what output was returned, and how that output affected later reasoning or actions. Toolformer (Schick et al., 2023) and ToolLLM (Qin et al., 2024) show that external tools can expand LLM capabilities through API calls, calculation, search, and other operations. Beyond execution success, provenance asks whether the call is justified, whether its arguments are derived from trusted or authorized sources, and whether its output is used appropriately.

A tool call contains several objects that provenance must record: the selected tool, schema, argument values, execution result, and downstream consumers. Argument provenance is especially important because legitimate tools can become unsafe when their parameters are derived from untrusted or incorrectly generated values. Agent-Sentry tracks the sources of values flowing into sensitive tool arguments, which motivates parameter-level lineage beyond tool-level permission checks (Sequeira et al., 2026). Tool outputs are also provenance-bearing units: they may support claims, update memory, trigger additional tools, modify external state, or conflict with other evidence. WebArena highlights the importance of correctly interpreting observations and executing state-changing actions in realistic web environments (Zhou et al., 2024), while τ-bench evaluates tool-agent-user interactions under domain-specific policies (Yao et al., 2025). Tool-use provenance should therefore connect sources, arguments, outputs, and downstream consumers into a single dependency chain.

4.2 Indirect Prompt Injection and Tool-Use Contamination

Tool-using agents are vulnerable to indirect prompt injection because they consume external content that may contain adversarial instructions. Unlike direct prompt injection, where the malicious instruction is placed in the user prompt, indirect injection embeds the instruction in webpages, documents, emails, database entries, tool outputs, or third-party API responses. The attack exploits a boundary failure: untrusted data may be interpreted as executable guidance rather than passive evidence (Greshake et al., 2023; Liu et al., 2023).

InjecAgent benchmarks indirect prompt injection in tool-integrated agent scenarios (Zhan et al., 2024), while AgentDojo evaluates attacks and defenses in realistic task environments (Debenedetti et al., 2024). Related benchmarks broaden the tool-risk surface. ToolEmu evaluates risky tool execution in an emulated sandbox (Ruan et al., 2024); OpenAgentSafety covers real tools such as browsers, code execution, filesystems, shells, and messaging platforms (Vijayvargiya et al., 2026); and MCP-SafetyBench targets real MCP servers and multi-server workflows (Zong et al., 2026). Together, these benchmarks show that contaminated tool contexts can cause unauthorized tool use, privacy leakage, ignored user intent, and unintended state changes.

From a provenance perspective, indirect prompt injection is an information-flow and influence-tracking problem. The key question is whether malicious content influenced a later tool call, argument value, memory update, or external action. A flat log may show that a webpage was retrieved and an email tool was later called, but not whether the recipient, subject, or body came from the webpage, the user instruction, or the model's reasoning. Relations such as Derive, Depend-on, Trigger, and Use make this influence explicit. Tool-use provenance should therefore distinguish trusted intent from untrusted content and support enforcement against unsafe flows into privileged execution channels.

4.3 Information Flow, Taint Tracking, and Provenance Enforcement

Recent defenses increasingly treat tool-use safety as an information-flow problem. Instead of relying only on prompting or output filtering, they track where information originates, how it propagates, and whether it is allowed to influence sensitive actions. This view extends classical information-flow control, where security policies regulate flows among subjects, objects, and security classes (Denning, 1976; Sabelfeld and Myers, 2003). For LLM agents, the analogous goal is to prevent low-integrity or untrusted data from exerting unauthorized influence over privileged tool calls, memory updates, or state-changing actions.

Defenses instantiate this principle at different layers. Instruction Hierarchy establishes precedence among instruction sources (Wallace et al., 2024), whereas StruQ separates instructions from untrusted data through structured channels (Chen et al., 2025). CaMeL makes this separation explicit at execution time by isolating

control flow from data flow and restricting how external data can influence agent behavior (Debenedetti et al., 2026). Fides formalizes agent-level information-flow control through confidentiality and integrity labels enforced during execution (Costa et al., 2025). At finer granularity, NeuroTaint propagates taint across neural and symbolic components (Cai et al., 2026), while Agent-Sentry tracks whether sensitive tool arguments are influenced by untrusted sources (Sequeira et al., 2026).

The key lesson is that unsafe behavior can arise from influence, not content alone. A webpage may be safe to summarize but unsafe if an embedded instruction determines an email recipient, database parameter, or authorization signal. Runtime provenance makes this distinction explicit by connecting sources, transformations, and sinks, allowing enforcement policies to account for a value's origin and use as well as its content.

4.4 Access Control and Execution Boundaries

Information-flow tracking must be paired with access control and execution boundaries. Even if an agent can reason about many tools, it should not be allowed to invoke every tool in every context. Classical principles such as least privilege and complete mediation require sensitive actions to be explicitly authorized and access decisions to be checked at the point of use (Saltzer and Schroeder, 1975). In tool-using agents, these principles translate into tool permissions, argument constraints, user confirmations, sandboxing, and policy-mediated execution.

Recent systems operationalize these principles at different layers. AgentSpec enforces user-defined runtime constraints over agent actions and tool use (Wang et al., 2026). AgentBound provides capability-constrained access control for MCP servers, limiting the resources available to each server (Bühler et al., 2026). Agent-Sentry instead uses execution provenance, including per-argument data flow, to bound tool calls according to learned behavior and user intent (Sequeira et al., 2026). Together, these systems show that tool-use safety requires controlling which tools and resources can be accessed, under what context, with which arguments, and with what external effects.

Provenance makes these controls context-sensitive. Agent-Sentry, for example, uses per-argument data flow and user-intent alignment to assess proposed tool calls (Sequeira et al., 2026). A read-only search tool may be permitted to consume untrusted web content, whereas an email, database-update, or code-execution tool may require arguments derived from authorized user intent, verified tool outputs, or explicit approval. Access control and provenance are therefore complementary: capability policies delimit the tools and resources available to an agent (Bühler et al., 2026), while provenance and runtime rules determine whether a particular action should proceed, be restricted, require confirmation, or be blocked (Wang et al., 2026; Sequeira et al., 2026).

4.5 Runtime Guardrails and Pre-/Post-Execution Verification

Runtime guardrails check agent behavior before, during, or after execution. Pre-execution guardrails decide whether a proposed tool call should proceed; runtime monitors track information flow and state changes; post-execution verification inspects traces, outputs, and side effects for debugging, audit, and recovery. General guardrail systems illustrate how safety mechanisms can be placed around LLM applications: NeMo Guardrails provides programmable conversational guardrails (Rebedea et al., 2023), while Llama Guard provides model-based input/output safety classification (Inan et al., 2023). Tool-using agents, however, require checks over execution state, tool arguments, and downstream effects, not only input–output text.

For high-impact tools, pre-execution verification is critical. A provenance-aware guardrail can check whether the tool is authorized, whether user intent is present, whether arguments are well formed, whether sensitive fields come from trusted sources, and whether the action violates policy. AgentSpec formalizes runtime constraints over agent actions (Wang et al., 2026). Agent-Sentry uses execution provenance to verify whether sensitive tool arguments are influenced by untrusted sources (Sequeira et al., 2026). ToolEmu complements these defenses by evaluating risky tool executions in an emulated sandbox (Ruan et al., 2024).

Post-execution verification supports failure localization and repair. When an agent gives a wrong answer or performs an unsafe action, traces can help identify whether the failure arose from evidence, tool use,

Table 5: Representative work on tool-use execution provenance. The table groups systems by the objects they track, the mechanisms they use, and the provenance role they support. † marks a work that had not been peer reviewed at the time of writing; for an entry naming two works, at least one of the two.

Line of Work	Representative Systems	Tracked Objects	Core mechanism (how)	Provenance Role
Tool-use learning	Toolformer (Schick et al., 2023); ToolLLM (Qin et al., 2024)	Tool calls, API arguments, tool outputs	Learn when and how to call tools from (self-)supervised tool-use trajectories	Tool-call trace construction
Prompt-injection and tool-risk benchmarks	InjecAgent (Zhan et al., 2024); AgentDojo (Debenedetti et al., 2024); ToolEmu (Ruan et al., 2024)	External content, injected instructions, private data	Embed adversarial instructions in external content and measure unsafe actions in (emulated) tool execution	Contamination and risk evaluation
Instruction–data separation	Instruction Hierarchy† (Wallace et al., 2024); StruQ (Chen et al., 2025); CaMeL (Debenedetti et al., 2026)	Privileged instructions, untrusted data, control/data flow	Mark privileged instructions vs. untrusted data and separate control flow from data flow via training or structured channels	Data–instruction boundary control
Information-flow tracking	Fides† (Costa et al., 2025); NeuroTaint† (Cai et al., 2026)	Security labels, tainted values, influence paths	Attach confidentiality/integrity (or taint) labels and propagate them source-to-sink, including through semantic transformations	Source-to-sink enforcement
Argument provenance	Agent-Sentry† (Sequeira et al., 2026)	Tool arguments, sources, trust labels	Build an execution-provenance graph and check whether sensitive tool arguments derive from untrusted sources	Sensitive-argument verification
Execution boundaries	AgentSpec (Wang et al., 2026); AgentBound (Bühler et al., 2026); MCP-SafetyBench (Zong et al., 2026)	Agent actions, permissions, policy constraints, and tool interfaces	Specify, enforce, or evaluate constraints over agent actions and tool access	Runtime action constraints and boundary-aware provenance
Guardrails	NeMo Guardrails (Rebedea et al., 2023); Llama Guard† (Inan et al., 2023)	Inputs, outputs, dialogue states, safety policies	External rule/classifier layer that filters inputs, outputs, and dialogue states against safety policies	Application-level safety filtering
Trace-based debugging	AgentOps† (Dong et al., 2024); AgentTrace† (AlSayyad et al., 2026); TRAIL† (Deshpande et al., 2025); LADYBUG (Rorseth et al., 2025)	Structured traces, artifacts, failure locations	Record structured execution traces and localize failures post hoc for audit and repair	Audit, localization, and recovery

memory, or downstream reasoning. AgentOps records agent artifacts and their lifecycle relationships (Dong et al., 2024), while AgentTrace structures execution records into operational, cognitive, and contextual traces (AlSayyad et al., 2026). TRAIL localizes failures using annotated trajectories (Deshpande et al., 2025), and LADYBUG uses trace-based debugging to support failure localization and repair (Rorseth et al., 2025).

Effective guardrails therefore require both semantic and procedural provenance: semantic provenance links claims to supporting or contradicting evidence, while procedural provenance links actions to authorized paths and trusted inputs. Together, they help detect unsupported claims, contaminated arguments, unauthorized tool calls, and invalid state updates.

Table 5 summarizes representative systems according to the objects they track, the mechanisms they use, and their role in tool-use provenance.

4.6 Summary

Tool-using agents make execution provenance necessary because tool calls can consume untrusted information, expose private data, modify external state, and trigger downstream consequences. The core questions are where tool arguments come from, whether tool outputs are trustworthy, how external content influences actions, and whether execution remains within authorized boundaries. Recent work on prompt-injection benchmarks, information-flow control, taint tracking, execution bounding, runtime guardrails, observability, and trace-based debugging marks a shift from output-only safety toward provenance-aware execution control.

Synthesis: from tool logging to influence control. Tool-use provenance is moving from recording events to governing influence. Tool-level traces show which API was called, but not whether the call was necessary, justified, or consistent with the user's goal. Parameter lineage can detect untrusted values in sensitive arguments, but may miss invalid interpretation of tool outputs. Information-flow and taint-based methods constrain unsafe influence, but remain difficult under summarization, paraphrasing, memory consolidation,

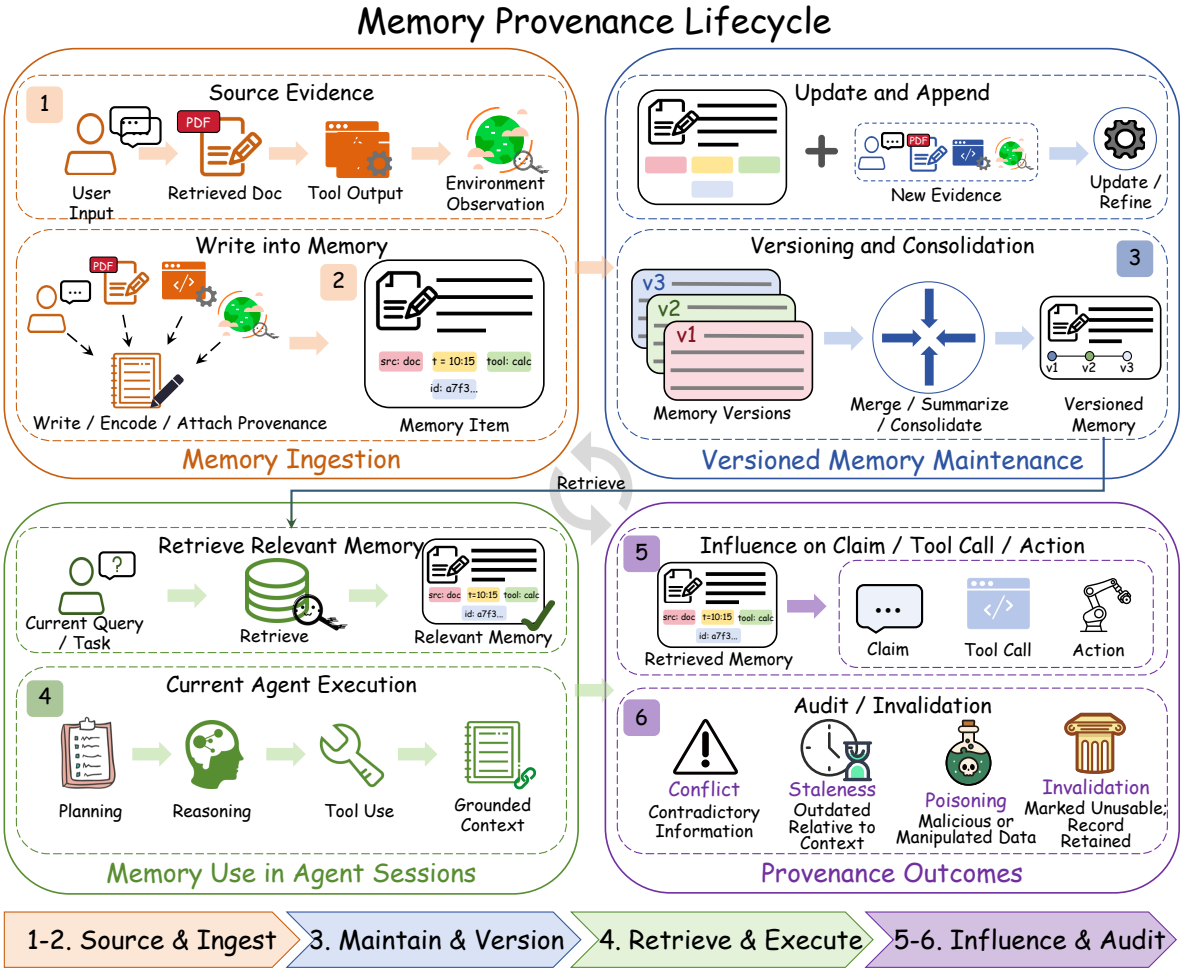


Figure 4: Memory as provenance-bearing evidence across agent sessions. Memory items are written from source evidence, maintained through updates and consolidation, retrieved into later agent executions, used to influence claims and actions, and audited for conflicts, staleness, poisoning, and invalidation.

and multi-step reasoning. The next step is to unify tool-call lineage, semantic evidence support, source trust, state changes, policy compliance, and recovery dependencies across the full agent workflow.

5 Memory as Provenance-Bearing Evidence

Memory is the *inward* counterpart of tool use. Whereas tool calls act on external systems, memory retains evidence and transforms, retrieves, and reuses it across turns, tasks, sessions, and environments. This makes memory a persistent part of the provenance chain: a memory item may later support a claim, shape a plan, fill a tool argument, update another memory, or justify a final answer. Memory therefore sharpens the trace-completeness tension: agents benefit from compact, persistent, and adaptive memory, but accountability requires source lineage, retrieval context, temporal validity, conflict status, and downstream influence.

This section does not survey memory architectures in general. Existing work has already reviewed memory mechanisms and evaluation protocols for LLM agents (Zhang et al., 2025; Du, 2026), as well as graph-based memory systems (Yang et al., 2026). We instead treat memory items as evidence-bearing objects. Under this view, a memory is not only something stored and retrieved; it is an artifact with an origin, transformations, revisions, validity conditions, and later effects. Without these links, long-term memory becomes an opaque evidence source rather than an auditable component of agent execution.

Figure 4 summarizes this lifecycle. Unlike storage-centric views of memory, it treats memory items as provenance-bearing evidence objects whose origins, updates, retrievals, validity, and downstream influence should remain inspectable.

5.1 Memory Writes, Source Attribution, and Lineage

Memory provenance begins when a memory item is written. A memory write may originate from user input, retrieved documents, environment observations, tool outputs, feedback, reflections, or messages from other agents. Existing memory systems illustrate different write paths: Generative Agents store observations for later reflection and planning (Park et al., 2023); Reflexion writes verbal feedback into episodic memory after failures (Shinn et al., 2023); MemoryBank updates long-term user memories through interaction (Zhong et al., 2024); MemGPT manages movement across memory tiers through explicit read and write operations (Packer et al., 2023); and A-MEM links new memories into an evolving memory structure (Xu et al., 2025).

From a provenance perspective, the important unit is not the memory record alone, but the relation between the record and the evidence that produced it. Each memory write should carry source and lineage metadata, including source type, timestamp, authoring agent, supporting evidence, transformation operation, confidence, and update history. This follows the broader provenance principle that generated artifacts should be linked to the entities, activities, and agents that produced them (World Wide Web Consortium, 2013; Souza et al., 2025), and aligns with tracing frameworks that record execution artifacts and operational context (AlSayyad et al., 2026) and with provenance graphs that track the sources of tool-call arguments (Sequeira et al., 2026). Such lineage is crucial because memory writes often transform information: documents are summarized, observations are merged, preferences are inferred, and failures are abstracted into reflections. A provenance-bearing memory should therefore distinguish raw observations, extracted facts, inferred memories, and revised memories.

Synthesis. Memory accountability starts at write time. If a memory is stored without source and transformation metadata, later verification can inspect only its surface content, not its evidential basis.

5.2 Memory Retrieval, Temporal Validity, and Downstream Influence

Retrieval determines which stored information re-enters the current execution. Retrieved memories may affect task decomposition, answer generation, tool choice, argument construction, user modeling, and future memory updates. Existing memory systems motivate this retrieval-centered view: MemoryBank and Mem0 emphasize personalization and multi-session continuity (Zhong et al., 2024; Chhikara et al., 2025); MemGPT and LongMem manage information beyond the current context window (Packer et al., 2023; Wang et al., 2023); and A-MEM improves retrieval through dynamic memory linking (Xu et al., 2025).

The provenance risk is that retrieved memory may be only partially relevant, outdated, over-generalized, derived from hallucinated content, poisoned by adversarial interaction, or invalid under the current context. Memory poisoning and injection attacks show that persistent memory can steer later responses and actions across sessions (Chen et al., 2024; Dong et al., 2025; Tian et al., 2026; Pulipaka et al., 2026). Temporal validity is therefore central: a memory item should record when it was created or updated, what evidence supported it, whether that evidence remains valid, and whether later observations superseded it.

A provenance-aware retrieval trace should record the triggering query or context, retrieved memory items, original sources, relevance signals, validity status, and downstream links to claims, tool calls, actions, final answers, or later memory updates. This would allow systems to identify whether a final output was influenced by a particular memory and whether that influence was justified. It also enables selective invalidation: when a memory is stale, unsupported, private, or contaminated, affected claims, actions, and derived memories can be located instead of treating the execution history as opaque (Wang et al., 2025; Wei et al., 2026; Sequeira et al., 2026).

Synthesis. Retrieval is not only a relevance problem. It is an influence problem: once memory enters context, provenance must track what it shaped and whether that influence remains defensible.

5.3 Conflicting, Contradictory, and Poisoned Memory

Persistent memory can accumulate incompatible, invalidated, or contaminated information. A stored memory may conflict with newer evidence, encode an adversarial instruction, or preserve a weak inference as if it were a stable fact. Recent attacks make this risk concrete. AgentPoison shows that poisoned long-term memory or RAG knowledge bases can act as backdoor substrates for agents (Chen et al., 2024). Memory-injection attacks show that malicious records can be introduced through ordinary interactions (Dong et al., 2025). Environment-injected and sleeper memory poisoning studies further show that untrusted webpages, documents, or repositories can enter memory, persist silently, and later steer behavior (Zou et al., 2026; Pulipaka et al., 2026).

This connects memory provenance to tool safety and information-flow control. Fides tracks confidentiality and integrity labels to restrict unsafe information flows (Costa et al., 2025), while NeuroTaint emphasizes that unsafe influence can survive semantic transformation and memory reuse rather than appearing as exact text reuse (Cai et al., 2026). Indirect prompt-injection benchmarks such as InjecAgent and AgentDojo show how untrusted external content can manipulate tool-using agents (Zhan et al., 2024; Debenedetti et al., 2024). When such content is written into memory, a single-run attack becomes a long-horizon provenance failure.

Synthesis. Memory quality cannot be measured only by recall, personalization, or coherence. It also depends on source trust, write-path legitimacy, conflict status, contamination risk, and whether a memory is later activated in consequential decisions.

5.4 Memory-Grounded Verification and Long-Term Audit

Memory-grounded verification extends evidence tracing from retrieved documents and current tool outputs to persistent memory. When a claim or tool action relies on memory, the system should link that memory item to the observation, document, tool call, user statement, or reflection that created it, as well as to later revisions, conflicts, invalidations, and downstream uses. Claim–evidence interfaces such as PaperTrail motivate this granularity by decomposing answers and sources into discrete claims and evidence units (Martin-Boyle et al., 2026). AgentOps and AgentTrace provide foundations for structured execution logs and inspection (Dong et al., 2024; AlSayyad et al., 2026), while Agent-Sentry shows how provenance graphs can support source-aware enforcement over tool-call arguments (Sequeira et al., 2026). Long-term memory requires an analogous structure across sessions.

Despite progress in systems such as MemGPT, A-MEM, and Mem0 (Packer et al., 2023; Xu et al., 2025; Chhikara et al., 2025), the memory systems reviewed here mainly optimize for recall, personalization, efficiency, or coherence, with limited support for tracing origins, temporal validity, conflicts, contamination, privacy exposure, and downstream influence. This gap is critical because memory itself can become an attack surface: poisoned, injected, stale, or private memories may later shape claims and actions without appearing as external evidence (Chen et al., 2024; Tian et al., 2026; Pulipaka et al., 2026; Wang et al., 2025). Defense work such as A-MemGuard suggests that memory should support checking and correction, not only input-time filtering (Wei et al., 2026).

Table 6 summarizes this gap. Across representative memory systems and security frameworks, write and retrieval support is common, but conflict handling, staleness detection, and evidence-aware verification remain much less developed.

Synthesis: memory usefulness versus memory accountability. Compression, consolidation, and persistence make memory useful; lineage, validity, conflict tracking, and influence links make it governable. The columns of Table 6 divide by literature. Write and retrieval support is present in every memory system reviewed here, and not one of them reaches full support on conflict handling or on evidence-aware verification; the only rows anywhere above partial in those two columns are Fides/NeuroTaint, which reaches full support on both, and Agent-Sentry, which reaches it on verification and stays partial on conflict. Both are information-flow and execution-provenance systems that observe memory from outside rather than memory systems that record their own lineage. The division has a consequence. Those systems establish conflict and support by tracking influence during a run, so what they recover is the current execution's use of a memory item rather than the item's history, which is why staleness remains partial even for them. Among the systems reviewed

Table 6: Agent memory through a provenance lens. ✓ explicit, documented mechanism; ▲ achieved indirectly or only for some memory types; ❍ information exposed but not implemented; ✗ absent; ★ capability studied adversarially rather than provided. † marks a work that had not been peer reviewed at the time of writing; for an entry naming two works, at least one of the two.

Paper / System	Memory Type	Update / Evolution	Write	Retr.	Conf.	Stale	Verify
Generative Agents (Park et al., 2023)	Memory stream / reflection	Reflection-based	▲	▲	✗	✗	✗
Reflexion (Shinn et al., 2023)	Episodic memory / feedback	Feedback-driven	▲	▲	✗	✗	▲
MemoryBank (Zhong et al., 2024)	Long-term interaction memory	Continuous update	▲	✓	❍	❍	✗
MemGPT† (Packer et al., 2023)	Hierarchical virtual memory	Memory-tier management	✓	✓	❍	❍	✗
LongMem (Wang et al., 2023)	Long-term memory for LMs	Cache and update	▲	✓	✗	▲	✗
Mem0 (Chhikara et al., 2025)	Production long-term memory	Extraction and consolidation	✓	✓	❍	▲	▲
A-MEM (Xu et al., 2025)	Agentic interconnected memory	Dynamic linking and evolution	✓	✓	❍	▲	✗
AgentPoison / Memory Injection (Chen et al., 2024; Dong et al., 2025)	Poisoned or injected memory	Cross-session persistence	★	★	✗	✗	✗
Fides / NeuroTaint† (Costa et al., 2025; Cai et al., 2026)	Information-flow and taint tracking	Tracks influence	✓	✓	✓	▲	✓
AgentOps / AgentTrace† (Dong et al., 2024; AlSayyad et al., 2026)	Execution logs and traces	Trace-level history	▲	▲	✗	✗	▲
Agent-Sentry† (Sequeira et al., 2026)	Execution provenance graph	Runtime provenance	✓	✓	▲	▲	✓

here, memory accountability is therefore supplied mainly from outside memory systems, and it does not persist beyond the session in which it was computed.

6 Benchmarks, Datasets, and Metrics

Evaluation is central to evidence tracing and execution provenance. Beyond final-answer correctness, provenance-aware agent evaluation must assess evidence support, tool-call justification, unsafe influence, trace completeness, failure localization, and recovery. Existing benchmarks provide partial views across RAG attribution, citation evaluation, agent execution, tool-use safety, memory, and multi-agent analysis, but no entry in the catalog of Table 8 evaluates complete provenance across evidence, tools, memory, traces, safety, and recovery. This section summarizes representative benchmark families, metrics, and limitations.

To provide a high-level view of these gaps, Figure 5 summarizes the coverage of major benchmark families across provenance-related evaluation capabilities.

The heatmap shows that strong coverage is concentrated in the capabilities each family was built around, such as evidence grounding, tool use, memory, multi-agent interaction, or safety attacks, and that no family reaches it on provenance relations or recovery. This motivates the more detailed discussion of benchmark families below.

6.1 Benchmark Families for Evidence Tracing

RAG and attribution benchmarks provide the closest foundation for evidence tracing because they evaluate whether generated outputs are grounded in retrieved or cited evidence (Gao et al., 2023; Es et al., 2024; Saad-Falcon et al., 2024; Min et al., 2023; Niu et al., 2024; Ru et al., 2024; Wu et al., 2025). These benchmarks generally focus on the relationship between a final answer and its supporting sources, rather than the full execution trajectory of an agent. Nevertheless, they provide important metrics and datasets for evaluating evidence recall, citation quality, faithfulness, and claim-level support.

At the answer level, ALCE treats citations as explicit links between answer spans and external sources and checks whether those citations support the generated content, while RAGAS and ARES score reference-free

Benchmark Family	Evidence Labels	Tool Calls	Memory	Multi-agent	Safety Attacks	Provenance Relations	Recovery
RAG benchmarks	S	L	L	L	L	P	L
Tool-use benchmarks	L	S	P	L	P	L	L
Memory benchmarks	P	P	S	L	L	P	P
Multi-agent benchmarks	P	P	P	S	L	P	L
Trace-debugging benchmarks	P	P	P	S	L	P	P
Provenance/security benchmarks	P	S	S	P	S	P	P

Legend: S Strong coverage P Partial coverage L Limited or absent coverage. *Coding rule.* Each cell reports the highest capability rating among the benchmarks in that family. Ratings follow the column-specific criteria detailed in Appendix D.1, which derives every cell. Family-level ratings therefore indicate the strongest available coverage, not coverage shared by all members. *Family membership.* **RAG**: ALCE, RAGAS, ARES, RAGChecker, RAGTruth, FActScore, SourceCheckup. **Tool-use**: AgentBench, WebArena, ToolBench/ToolLLM, τ-bench. **Memory**: LoCoMo, MemoryArena. **Multi-agent**: MAST, AgenTracer/Aegis. **Trace-debugging**: TRAIL, MAST. **Provenance/security**: ToolEmu, InjecAgent, AgentDojo, OpenAgentSafety, MCP-SafetyBench, Agent-Sentry Bench, NeuroTaint/TaintBench, AgentSpec, AgentBound.

Figure 5: Benchmark coverage heatmap across provenance-related evaluation capabilities. The heatmap summarizes benchmark families at a high level rather than listing individual datasets. It highlights that existing benchmarks provide partial coverage of isolated components, while end-to-end evaluation of evidence tracing and execution provenance remains underdeveloped.

faithfulness, context relevance, and answer relevance using lightweight model judges (Gao et al., 2023; Es et al., 2024; Saad-Falcon et al., 2024). These frameworks share one assumption, that an answer should be supported by retrieved evidence. RAGAS does decompose the answer, extracting a set of statements and scoring each for support, but the check is made against the retrieved context as a whole; what none of these frameworks reports is a link from a claim to the particular passage that supports it.

Finer-grained benchmarks move toward claim-level provenance, differing mainly in the unit they score: retrieval-versus-generation error separation (RAGChecker), word-level hallucination annotation (RAGTruth), atomic-fact support (FActScore), source-supportiveness of cited references (SourceCheckup), and supported/refuted/unverifiable claim classification (FEVER) (Ru et al., 2024; Niu et al., 2024; Min et al., 2023; Wu et al., 2025; Thorne et al., 2018). The trend is a shift from answer-level to claim-level grounding; SourceCheckup sharpens the point by showing that citation presence does not imply evidence support.

These benchmarks are important for evidence tracing, but they remain limited from an agent-provenance perspective. Their primary evaluation targets are answer-source grounding, citation support, faithfulness, hallucination annotation, or claim-level factual support (Gao et al., 2023; Es et al., 2024; Saad-Falcon et al., 2024; Min et al., 2023; Niu et al., 2024; Ru et al., 2024; Wu et al., 2025). As a result, they provide only partial coverage of agent provenance: they help evaluate whether claims are supported by evidence, but they do not directly evaluate how tool calls, memory operations, environment observations, intermediate decisions, or multi-agent communication shape the final answer or action. Thus, they should be viewed as foundations for evidence attribution rather than complete benchmarks for execution provenance.

6.2 Benchmarks for Agent Execution Traces and Tool-Use Safety

Agent benchmarks extend evaluation from static answer generation to interactive, multi-step execution, exposing trajectories, tool calls, environment states, and intermediate decisions. They differ mainly in setting and in what they score: multi-environment capability (AgentBench), long-horizon web interaction (WebArena), API selection and execution paths (ToolBench/ToolLLM), and tool-agent-user workflows that grade policy following and final database state rather than only the final response (τ-bench) (Liu et al., 2024;

Zhou et al., 2024; Qin et al., 2024; Yao et al., 2025). The common trend is a move from scoring the answer toward scoring the process and its end state.

Trace-oriented benchmarks connect execution provenance to failure diagnosis. AgentTrace provides structured operational, cognitive, and contextual logs for agent observability and downstream audit (AlSayyad et al., 2026), while TRAIL uses human-annotated traces to evaluate whether systems can localize failures to specific steps, components, or interactions (Deshpande et al., 2025). Recent multi-agent work moves further from trace inspection to explicit failure attribution: AgenTracer constructs TracerTraj through counterfactual replay and fault injection to identify responsible agents and steps (Zhang et al., 2026), and Aegis generates positive–negative trajectory pairs with faulty-agent and error-mode labels (Kong et al., 2026). Together, these works show that provenance can be evaluated by its ability to support failure localization and attribution, not only by whether a trace was recorded.

Tool-use safety benchmarks examine whether agents remain safe when tools introduce external risks. ToolEmu evaluates high-risk tool use in an LM-emulated sandbox (Ruan et al., 2024); InjecAgent and AgentDojo study indirect prompt injection and adversarial tool interactions in tool-integrated settings (Zhan et al., 2024; Debenedetti et al., 2024); and OpenAgentSafety extends evaluation to real tools and multi-turn, multi-user tasks across multiple risk categories (Vijayvargiya et al., 2026). MCP-SafetyBench further studies safety risks in real MCP servers and multi-server workflows (Zong et al., 2026). These benchmarks matter for provenance because unsafe behavior often depends on hidden flows from untrusted sources into tool arguments, execution boundaries, or state-changing actions.

Recent work evaluates more explicit execution-provenance and information-flow mechanisms. Agent-Sentry introduces execution provenance graphs to bound LLM-agent behavior and detect out-of-bounds tool calls (Sequeira et al., 2026). NeuroTaint and TaintBench evaluate semantic taint and source-to-sink influence in LLM agents, emphasizing that unsafe influence may be transformed through summarization, inference, or memory rather than copied verbatim (Cai et al., 2026). Fides applies information-flow control to secure AI agents by tracking confidentiality and integrity labels (Costa et al., 2025). AgentSpec evaluates runtime policy enforcement through customizable constraints (Wang et al., 2026). AgentBound extends this line of work to execution boundaries and access control in MCP-style agent-tool ecosystems (Bühler et al., 2026). Together, these benchmarks and systems shift evaluation from "can the agent use tools?" to "can the agent use tools safely, with traceable and enforceable provenance?"

6.3 Memory and Multi-Agent Provenance Evaluation

Memory and multi-agent settings introduce additional provenance challenges because information may persist across sessions, be updated over time, or propagate across agents. Existing memory benchmarks and systems mainly evaluate recall, personalization, long-term consistency, memory updating, retrieval, or multi-session task performance (Zhong et al., 2024; Packer et al., 2023; Xu et al., 2025; Chhikara et al., 2025; Maharana et al., 2024; He et al., 2026). MemoryBank evaluates long-term memory recall, continuous updating, and personalization (Zhong et al., 2024); MemGPT makes memory read and write operations explicit through hierarchical memory management (Packer et al., 2023); A-MEM organizes agent memory through dynamically evolving interconnected memory structures (Xu et al., 2025); Mem0 evaluates memory extraction, consolidation, and retrieval in multi-session dialogue settings (Chhikara et al., 2025); LoCoMo evaluates long-term conversational memory (Maharana et al., 2024); and MemoryArena evaluates memory use in interdependent multi-session agentic tasks (He et al., 2026). Existing memory and graph-memory surveys further summarize progress in memory mechanisms, evaluation protocols, and graph-based memory systems (Zhang et al., 2025; Du, 2026; Yang et al., 2026).

From a provenance perspective, these benchmarks provide useful but incomplete proxies for memory provenance. They often test whether an agent remembers useful information, but neither of the two memory benchmarks in Table 8 evaluates whether a memory item is traceable to its source, temporally valid, consistent with newer evidence, protected from contamination, or responsible for a downstream action. Dedicated memory-provenance evaluation should therefore assess memory source attribution, update history, conflict resolution, stale-memory invalidation, contamination detection, and downstream influence, which remain

underdeveloped in current memory evaluation protocols (Maharana et al., 2024; He et al., 2026; Zhang et al., 2025; Du, 2026).

Multi-agent systems introduce another layer of provenance complexity because evidence, assumptions, messages, and errors can propagate across agent boundaries. AutoGen and CAMEL provide representative frameworks for multi-agent communication, role assignment, and collaborative task solving (Wu et al., 2024; Li et al., 2023). Evaluating provenance in such settings requires more than measuring final task success: it requires identifying which agent contributed which evidence, which communication step introduced an error, and how responsibility is distributed across agents. MAST analyzes failures from specification errors, inter-agent misalignment, and weak task verification (Cemri et al., 2025), while TRAIL evaluates issue localization from agent traces (Deshpande et al., 2025). AgenTracer and Aegis further construct labeled multi-agent trajectories for identifying faulty agents, responsible steps, and error modes (Zhang et al., 2026; Kong et al., 2026). Together, these works suggest that multi-agent provenance benchmarks should evaluate evidence propagation, communication-level error introduction, cross-agent dependency, and responsibility attribution, rather than only final coordination success.

6.4 Metrics for Evidence Tracing and Execution Provenance

Existing benchmarks use diverse metrics for retrieval grounding, citation quality, tool use, safety, memory, and agent execution (Gao et al., 2023; Min et al., 2023; Ru et al., 2024; Wu et al., 2025; Ruan et al., 2024; Debenedetti et al., 2024; Maharana et al., 2024; Deshpande et al., 2025; Cemri et al., 2025). From the perspective of evidence tracing and execution provenance, these metrics can be reorganized into four evaluation aspects: evidence attribution, execution provenance, safety and robustness, and debugging and recovery. Table 7 summarizes representative metric dimensions and their related foundations. Some metrics are already used in RAG, attribution, tool-use, and agent benchmarks, while others are natural extensions for provenance-aware evaluation.

Research agenda for provenance metrics. The execution-provenance row of Table 7 is the only row marked ★: trace completeness, provenance accuracy, dependency coverage, and temporal consistency are *not* yet available as measures. None has an agreed definition in the work we surveyed, none has an established ground-truth construction there, and they are not offered as an evaluation framework ready for adoption. We describe each with the ground truth it would require, so that the gap can be acted on.

Trace completeness would ask what fraction of the units and relations that actually occurred are present in the recorded trace. Its ground truth is the hardest of the four to obtain from observation, because the missing part of a trace is by definition unrecorded; the practical route is synthetic or instrumented execution, where a reference harness emits an exhaustive trace and the system under evaluation is scored against it. *Dependency coverage* would ask what fraction of the DEPEND-ON and TRIGGER edges implied by an execution are recovered. Its ground truth is obtainable by ablation and replay: perturb each candidate dependency and observe whether the dependent step changes (Zhang et al., 2026). This makes it the most tractable of the three that need external ground truth, at a cost of one replay per candidate edge. *Temporal consistency* would ask whether the recorded ordering and validity intervals are internally coherent, for example whether an item was used after it was invalidated. This is the only one of the four that needs no external ground truth, since it is a self-consistency check on the graph, and it is best reported as a violation count rather than an accuracy. *Provenance accuracy* would ask whether the recorded relations are the correct ones, and it is the metric whose definition is genuinely open, for the reason given next.

Why provenance accuracy resists a single gold graph. The same execution frequently admits several defensible dependency graphs. Annotators may disagree on granularity, on whether a passage that was present but unnecessary is a dependency, and on whether a step was triggered by an observation or by the plan that anticipated it. Scoring exact-match against one designated gold graph therefore measures annotator agreement as much as system quality. We see three defensible responses; which one the field should adopt remains open. The first is to score against an *admissible set*: annotate the set of graphs consistent with the trace under stated conventions and count a system correct if its output lies in that set. The second is to score edges rather than graphs, reporting per-relation precision and recall against a union of annotations, weighted by how many

Table 7: Representative metric dimensions for evaluating evidence tracing and execution provenance in LLM agents. The **Status** column flags maturity: *Established* metrics are operationalized in existing RAG, attribution, tool-use, or safety benchmarks; *Partly established* metrics exist for some sub-tasks (e.g., failure localization in TRAIL/MAST) but not for recovery; *Proposed* metrics (trace completeness, provenance accuracy, dependency coverage, temporal consistency) are desiderata that are not yet operationalized with agreed definitions. ✓: Established; •: Partly established; ★: Proposed.

Evaluation Aspect	**Core Question**	**Representative Metrics**	**Status**	**Related Basis**
Evidence attribution	Are the agent's claims supported by reliable evidence?	Evidence recall, citation precision, source supportiveness, faithfulness, claim support accuracy.	✓	RAG and attribution evaluation (Gao et al., 2023; Min et al., 2023; Niu et al., 2024; Ru et al., 2024; Wu et al., 2025).
Execution provenance	Is the agent's execution process completely and correctly traceable?	Trace completeness, provenance accuracy, dependency coverage, temporal consistency.	★	Provenance standards and agent observability (World Wide Web Consortium, 2013; Dong et al., 2024; AlSayyad et al., 2026; Souza et al., 2025).
Safety and robustness	Can the system detect unsafe or untrusted influence during execution?	Unsafe influence detection, attack success rate, policy violation rate, false-positive rate.	✓	Tool-use safety, prompt injection, and information-flow control (Ruan et al., 2024; Zhan et al., 2024; Debenedetti et al., 2024; 2026; Costa et al., 2025; Zong et al., 2026).
Debugging and recovery	Can traces help identify and repair failures?	Failure localization accuracy, diagnosis correctness, auditability, recovery success.	•	Trace-based debugging, multi-agent failure analysis, and failure-attribution datasets (Deshpande et al., 2025; Cemri et al., 2025; AlSayyad et al., 2026; Zhang et al., 2026; Kong et al., 2026).

annotators asserted each edge, so that contested edges contribute less than unanimous ones. The third is to define a canonical form and score against it, for instance the *minimal sufficient graph*, a subgraph in which every edge is necessary to explain the observed outputs under replay. This sidesteps disagreement about redundant edges at the cost of discarding genuine over-determination. In all three cases the measurement is meaningful only if inter-annotator agreement is reported alongside it, since that agreement is the ceiling on any achievable score. This is compounded by the origin heterogeneity of Section 2.1: instrumented edges can be checked against the log and inferred edges cannot, so a single accuracy number over a mixed graph conflates a reproducible check with a judgment call, and the two should be reported separately.

The table highlights that provenance-aware evaluation should go beyond final-answer correctness and component-level task success. Building on prior work in attribution evaluation, provenance modeling, agent observability, tool-use safety, and trace-based debugging (Gao et al., 2023; Min et al., 2023; Ru et al., 2024; World Wide Web Consortium, 2013; Dong et al., 2024; AlSayyad et al., 2026; Souza et al., 2025; Ruan et al., 2024; Debenedetti et al., 2024; Deshpande et al., 2025; Cemri et al., 2025), we organize these metrics by their evaluation function. Evidence attribution metrics assess claim grounding; execution-provenance metrics, once operationalized, would assess traceability; safety and robustness metrics assess unsafe influence and policy violations; and debugging and recovery metrics assess whether traces support failure localization, audit, and repair. Together, these dimensions provide a broader evaluation view for agents whose outputs depend on retrieval, tools, memory, communication, and external state changes.

Table 8: Benchmark catalog for evidence tracing and execution provenance (25 entries). ✓ per-instance ground-truth labels released with a metric over them; • present in the released data but unlabeled or only partly covered; ✗ neither released; T coverage varies across the benchmark's task suite; S realized as external environment state rather than as an agent-side artifact. † marks a work that had not been peer reviewed at the time of writing; for an entry naming two works, at least one of the two.

Benchmark	Task Type	Trace Avail.	Evid. Labels	Tool Calls	Mem.	Multi Agent	Safety Attack	Main Metrics
ALCE (Gao et al., 2023)	Citation generation	•	✓	✗	✗	✗	✗	Citation quality, correctness, fluency
RAGAS (Es et al., 2024)	RAG evaluation	✗	•	✗	✗	✗	✗	Faithfulness, context relevance, answer relevance
ARES (Saad-Falcon et al., 2024)	RAG evaluation	✗	•	✗	✗	✗	✗	Context relevance, answer faithfulness, answer relevance
RAGChecker (Ru et al., 2024)	Fine-grained RAG diagnosis	•	•	✗	✗	✗	✗	Retrieval and generation diagnosis
RAGTruth (Niu et al., 2024)	RAG hallucination annotation	•	✓	✗	✗	✗	✗	Word-level hallucination labels
FActScore (Min et al., 2023)	Atomic factuality	✗	✓	✗	✗	✗	✗	Atomic fact support
SourceCheckup (Wu et al., 2025)	Source supportiveness	✗	✓	✗	✗	✗	✗	Source relevance and supportiveness
AgentBench (Liu et al., 2024)	Agent capability evaluation	•	✗	•	T	✗	✗	Task success, environment-specific scores
WebArena (Zhou et al., 2024)	Web-agent execution	✓	✗	✓	✗	✗	✗	Task success, web interaction outcome
ToolBench / ToolLLM (Qin et al., 2024)	API tool use	✓	✗	✓	✗	✗	✗	Tool selection, argument correctness, execution success
τ-bench (Yao et al., 2025)	Tool-agent-user interaction	✓	✗	✓	S	✗	•	Task success, policy following, final state
TRAIL† (Deshpande et al., 2025)	Trace debugging	✓	•	•	•	•	✗	Issue localization, error type diagnosis
MAST (Cemri et al., 2025)	Multi-agent failure analysis	✓	✗	T	T	✓	✗	Failure taxonomy, error attribution
ToolEmu (Ruan et al., 2024)	Tool-use risk evaluation	✓	✗	✓	✗	✗	✓	Risk detection, unsafe action rate
InjecAgent (Zhan et al., 2024)	Indirect prompt injection	✓	✗	✓	✗	✗	✓	Attack success rate, defense performance
AgentDojo (Debenedetti et al., 2024)	Prompt injection attacks and defenses	✓	✗	✓	✗	✗	✓	Attack success, utility, defense robustness
Agent-Sentry Bench† (Sequeira et al., 2026)	Out-of-bounds tool execution	✓	•	✓	✗	✗	✓	Out-of-bounds detection, false positives
NeuroTaint / TaintBench† (Cai et al., 2026)	Semantic taint tracking	✓	•	✓	✓	•	✓	Unsafe influence detection, taint propagation
AgentBound (Bühler et al., 2026)	MCP execution boundary	✓	✗	✓	✗	✗	✓	Access-control violations, policy enforcement
AgenTracer / Aegis (Zhang et al., 2026; Kong et al., 2026)	Multi-agent failure attribution	✓	•	T	T	✓	✗	Responsible-agent and error-mode attribution
OpenAgentSafety (Vijayvargiya et al., 2026)	Real-tool safety evaluation	✓	•	✓	T	•	✓	Unsafe action rate, task utility, risk categories
MCP-SafetyBench (Zong et al., 2026)	MCP workflow safety	✓	✗	✓	✗	•	✓	Attack success, policy violation, workflow risk
AgentSpec (Wang et al., 2026)	Runtime policy enforcement	✓	✗	✓	✗	•	✓	Policy violation rate, enforcement accuracy
LoCoMo (Maharana et al., 2024)	Long-term conversational memory	•	•	✗	✓	✗	✗	Long-range QA, event summarization, memory recall
MemoryArena (He et al., 2026)	Multi-session agent memory	✓	•	T	✓	✗	✗	Multi-session task success, memory use

6.5 Limitations of Current Evaluation Protocols

Existing benchmarks provide useful foundations for evaluating retrieval grounding, tool use, memory, web interaction, safety, and multi-agent behavior, but they remain insufficient for full evidence tracing and execution provenance. RAG and attribution benchmarks mainly focus on answer-source grounding or citation quality (Gao et al., 2023; Ru et al., 2024; Wu et al., 2025). Tool-use and agent benchmarks emphasize task success, unsafe tool behavior, prompt injection, environment interaction, real-world tool safety, or MCP-specific attacks (Liu et al., 2024; Zhou et al., 2024; Ruan et al., 2024; Debenedetti et al., 2024; Yao et al., 2025; Vijayvargiya et al., 2026; Zong et al., 2026). Memory and multi-agent benchmarks evaluate recall, multi-session memory use, coordination, failure modes, or error attribution (Maharana et al., 2024; He et al., 2026; Deshpande et al., 2025; Cemri et al., 2025; Zhang et al., 2026; Kong et al., 2026). Although recent work has improved realistic tool-safety evaluation and multi-agent failure attribution, no benchmark in the catalog of Table 8 evaluates the complete provenance chain from evidence and tool outputs to memory updates, intermediate claims, final answers, external state changes, and recovery actions.

A key limitation is the lack of unified execution traces and provenance-relation annotations. Trace-oriented benchmarks and observability frameworks have made useful progress, but they still differ in what artifacts they record and how provenance is represented (Dong et al., 2024; AlSayyad et al., 2026; Souza et al., 2025;

Deshpande et al., 2025; Cemri et al., 2025). No entry in Table 8 jointly captures user requests, retrieved evidence, tool calls and outputs, memory reads and writes, inter-agent messages, intermediate claims, final responses, and state changes. None annotates the typed relations of such a joint trace as ground truth; existing labels cover single relations in isolation, such as claim-level SUPPORT in attribution benchmarks or source-to-sink influence in taint benchmarks, despite the availability of general provenance concepts in standards such as W3C PROV-DM (World Wide Web Consortium, 2013). As a result, current evaluation often measures correctness, faithfulness, or attack success indirectly, rather than directly assessing trace completeness, provenance accuracy, claim support, memory validity, or cross-agent responsibility.

Recovery and governance are also under-evaluated. The benchmarks cataloged in Table 8 commonly test whether an agent succeeds, fails, hallucinates, or takes an unsafe action, but none scores whether provenance can help the system repair execution, such as by invalidating stale memory, quarantining contaminated evidence, retrying a tool call, requesting human approval, or rolling back an unsafe state change (Ruan et al., 2024; Debenedetti et al., 2024; Deshpande et al., 2025; Cemri et al., 2025). Realistic evaluation is further complicated by long-horizon and enterprise traces involving private tools, changing data, user-specific memory, organizational policies, credentials, and confidential intermediate states (Maharana et al., 2024; He et al., 2026; Dong et al., 2024; AlSayyad et al., 2026). Future provenance benchmarks should therefore evaluate not only whether agents succeed, but also whether their behavior is traceable, auditable, safe, privacy-aware, and recoverable (Costa et al., 2025).

Table 8 catalogs all 25 benchmark entries of the corpus by their task type, trace availability, evidence labels, tool calls, memory, multi-agent setting, safety attacks, and main metrics.

Read across the catalog, the shortfall is structural rather than incidental. The two columns no family reaches the top rating on, provenance relations and recovery, are also the two that adding tasks of the current kind does not fill: a relation counts as ground truth only once it is released as a per-instance label, and recovery becomes measurable only once a benchmark scores a repair rather than a prevention. Every benchmark here is built by writing tasks, so adding more such benchmarks would change neither column. What the field lacks is therefore not more agent tasks but released relation annotations and replayable executions, which is the requirement Section 7 takes up. Appendix D.1 reports the per-family counts behind this reading and the size effect that qualifies them.

7 Open Problems and Future Directions

Despite progress in RAG attribution, agent observability, tool-use safety, memory systems, graph-based representation, debugging, and multi-agent evaluation, evidence tracing and execution provenance for LLM agents remain fragmented. Many existing systems record execution artifacts, but no system in Table 3 explains how evidence, tools, memory, observations, messages, and actions jointly shape an agent's final answer or external behavior. This section outlines open problems toward provenance-aware agents that support verification, audit, attribution, runtime safety, recovery, and governance.

7.1 Toward Unified and Interoperable Trace Schemas

A central open problem is the lack of unified trace schemas for LLM agents. Existing frameworks and benchmarks record different artifacts, including prompts, model responses, tool calls, retrieved documents, memory operations, environment observations, execution metadata, and error events (Dong et al., 2024; AlSayyad et al., 2026; Souza et al., 2025; Deshpande et al., 2025; Cemri et al., 2025). However, few schemas jointly represent reasoning steps, retrieval, tool calls and outputs, memory reads and writes, inter-agent messages, intermediate claims, final responses, and external state changes.

Existing standards provide useful building blocks but do not fully solve the agent-provenance problem. W3C PROV-DM offers a general vocabulary for entities, activities, agents, and derivation relations (World Wide Web Consortium, 2013); OpenTelemetry and OpenLineage provide abstractions for distributed traces and data workflow lineage (OpenTelemetry, 2026; OpenLineage Contributors, 2026); and PROV-AGENT adapts provenance modeling to agentic workflows (Souza et al., 2025). Future schemas should connect these operational trace structures with semantic evidence relations. They should represent claims, evidence units,

tool observations, memory items, environment states, and agent messages, together with relations such as SUPPORT, CONTRADICT, DEPEND-ON, UPDATE, and INVALIDATE. Such schemas would make agent traces easier to compare, replay, annotate, audit, and exchange across frameworks.

7.2 Claim-Level and Semantic Provenance

Current attribution methods often operate at coarse granularity, such as answer-level citation, context-level faithfulness, or step-level logging (Gao et al., 2023; Ru et al., 2024; Wu et al., 2025; Schreieder et al., 2026). However, agent outputs frequently contain multiple claims, intermediate assumptions, and action-relevant inferences. A final response may be partially correct: one claim may be supported by retrieved evidence, another may be inferred beyond the source, and another may be contradicted by a tool output or memory item. This motivates claim-level provenance, building on fine-grained factuality and attribution work such as ALCE, FActScore, RAGTruth, RAGChecker, and SourceCheckup (Gao et al., 2023; Min et al., 2023; Niu et al., 2024; Ru et al., 2024; Wu et al., 2025).

For agents, the harder problem is semantic provenance across transformations. Evidence may be paraphrased, summarized, aggregated, compressed into memory, passed through tools, or reused by another agent before influencing a claim or action. String-level matching and citation presence are therefore insufficient. Future work should trace semantic dependencies across retrieval, tool outputs, memory updates, reflection, and inter-agent communication, including cases where evidence is indirectly used or transformed across multiple steps. Work on semantic taint and causal influence points toward this direction, but robust claim-level provenance for long agent executions remains open (Cai et al., 2026).

7.3 Memory and Multi-Agent Provenance

Memory provenance remains underdeveloped. Long-term memory systems allow agents to store, retrieve, update, and reuse information across sessions, but they often provide limited support for tracing where memory items came from, whether they remain valid, whether they conflict with newer evidence, and how they influence later claims or actions. Existing memory and graph-memory surveys cover architectures, evaluation methods, and relational memory structures (Zhang et al., 2025; Du, 2026; Yang et al., 2026), but memory as provenance-bearing evidence remains a distinct open problem.

Future memory systems should attach provenance metadata to memory items, including source attribution, write time, update history, retrieval context, downstream usage, conflict status, and invalidation events. Memory should not be treated as an unqualified context reservoir: a memory item derived from user input, retrieved evidence, a tool output, an environment observation, a reflection, or another agent message should carry different trust implications. This is especially important when memory affects tool calls, personalization, or long-term decisions. Memory poisoning and memory-injection attacks further show that compromised memory can influence future behavior without modifying model weights (Chen et al., 2024; Dong et al., 2025).

Multi-agent provenance introduces additional complexity because agents may communicate, delegate tasks, verify one another's outputs, and jointly produce final decisions. Frameworks such as AutoGen and CAMEL demonstrate these distributed workflows (Wu et al., 2024; Li et al., 2023), while MAST shows that failures can arise from specification errors, inter-agent misalignment, weak verification, and coordination breakdowns (Cemri et al., 2025). AgenTracer and Aegis further show that responsible-agent and responsible-step attribution can be operationalized through labeled, counterfactual, or contrastive multi-agent trajectories (Zhang et al., 2026; Kong et al., 2026). Future provenance models should therefore track how claims, evidence, messages, and errors propagate across agents: which agent introduced a claim, which agent verified or reused it, which message carried it, which step induced failure, and where responsibility lies. The remaining challenge is to integrate responsibility attribution with claim-level evidence provenance, memory lineage, and recovery actions, rather than treating failure attribution as a separate diagnostic task.

7.4 Runtime Safety, Enforcement, and Recovery

Runtime safety for LLM agents requires more than output filtering. In tool-using agents, dangerous behavior may arise when untrusted content influences tool arguments, when private information flows into public

outputs, when stale memory affects a privileged action, or when an agent invokes a tool outside its intended boundary. Guardrails that inspect only the final answer or the current tool call may miss these source-to-sink dependencies. Provenance-aware guardrails should reason over the evidence, memory items, tool outputs, observations, and external content that influenced a proposed action (Zhan et al., 2024; Debenedetti et al., 2024; 2026; Costa et al., 2025; Sequeira et al., 2026; Bühler et al., 2026).

Recent work points toward provenance-aware enforcement through prompt-injection benchmarks, control/data-flow separation, information-flow control, semantic taint tracking, runtime constraints, execution provenance graphs, and access boundaries (Ruan et al., 2024; Zhan et al., 2024; Debenedetti et al., 2024; 2026; Costa et al., 2025; Cai et al., 2026; Wang et al., 2026; Sequeira et al., 2026; Bühler et al., 2026). The remaining challenge is recovery. Most systems focus on detecting or blocking unsafe behavior, but trustworthy agents also need to repair unsupported or unsafe executions. Future systems should use provenance to invalidate stale memory, quarantine contaminated evidence, retry tool calls with corrected parameters, request human approval, roll back unsafe state changes, or replay execution from safe checkpoints. This requires provenance representations that support intervention and recovery, not only explanation.

7.5 Realistic Benchmarks, Privacy, and Governance

Current benchmarks still lack realistic full execution traces. Read across Table 8, the pattern is consistent: RAG benchmarks provide evidence labels but none releases tools or memory; provenance and security benchmarks provide attacks and tool calls but none releases claim-level evidence labels; memory benchmarks evaluate long-range recall and multi-session memory use but neither evaluates lineage, conflict handling, or downstream influence; and multi-agent benchmarks analyze failures but neither releases complete provenance annotations. Benchmarks such as AgentBench, WebArena, τ-bench, TRAIL, MAST, LoCoMo, and MemoryArena provide important foundations for agent evaluation, trace analysis, multi-agent failure diagnosis, and long-term memory assessment (Liu et al., 2024; Zhou et al., 2024; Yao et al., 2025; Deshpande et al., 2025; Cemri et al., 2025; Maharana et al., 2024; He et al., 2026). However, none of the benchmark families derived from Table 8 in Appendix D.1 fully evaluates evidence tracing and execution provenance across retrieval, tool use, memory, multi-agent communication, safety attacks, and recovery.

Future benchmarks should move from isolated answer-, task-, or attack-level evaluation toward trace-level evaluation. Provenance standards such as W3C PROV-DM provide a useful vocabulary for representing entities, activities, agents, and derivation relations (World Wide Web Consortium, 2013). Agent observability frameworks such as AgentOps and AgentTrace show how agent executions can be collected and inspected as structured traces (Dong et al., 2024; AlSayyad et al., 2026). PROV-AGENT further adapts provenance modeling to agentic workflows (Souza et al., 2025). Building on these foundations, future benchmarks should capture complete execution traces across agent workflows and annotate the provenance relations needed to evaluate evidence support, unsafe influence, responsibility attribution, and recovery.

Existing work highlights different parts of this agenda. TRAIL shows how execution traces can support failure analysis in agent workflows (Deshpande et al., 2025). MAST focuses on diagnosing failure modes in multi-agent systems (Cemri et al., 2025). AgenTracer attributes failures to responsible agents and steps through counterfactual replay and fault injection (Zhang et al., 2026). Aegis generates multi-agent trajectories with injected errors labeled by faulty agent and error mode (Kong et al., 2026). OpenAgentSafety emphasizes realistic tool-use safety risks in open agent settings (Vijayvargiya et al., 2026). MCP-SafetyBench extends safety evaluation to protocol-level agent-tool environments (Zong et al., 2026). The remaining challenge is to integrate these dimensions into benchmarks that jointly test claim-level evidence support, memory validity, unsafe influence detection, auditability, and recovery.

Privacy and governance introduce a separate but equally important challenge. Provenance traces may contain sensitive user data, confidential documents, credentials, tool outputs, API arguments, internal reasoning artifacts, and long-term memory items. Observability frameworks such as AgentOps and AgentTrace provide useful foundations for collecting and inspecting traces (Dong et al., 2024; AlSayyad et al., 2026), while security mechanisms such as Fides highlight the need to track confidentiality and integrity constraints in agent workflows (Costa et al., 2025). Future provenance infrastructure should therefore support trace minimization,

access control, anonymization, retention policies, encryption, and selective disclosure. Without privacy-aware governance, provenance infrastructure itself may become a source of compliance and security risk.

7.6 Trace Assurance and Attested Provenance

A dimension that current work leaves almost entirely open is *assurance*: whether a trace can be trusted by a party other than the one that produced it. Section 2.1 introduces assurance as a cross-cutting attribute, and the reviewed descriptions of the systems in Table 3 do not establish tamper-evident or attested trace guarantees. As described, they write traces to a log that could be edited, truncated, or reordered after the fact by whoever operates the agent. This is adequate for debugging, where the reader and the producer are the same party, and inadequate for audit, where they are not. Work on audit-trail infrastructure and auditable agent design has begun to argue for auditable trace infrastructure (Ojewale et al., 2026; Nian et al., 2026), and work on checkpoint-restore integrity shows that agent state itself is an attack surface (Zheng et al., 2026), but no assurance tier has been standardized across the agent provenance systems we survey.

Three questions follow. First, what should be attested? Signing an entire trace is costly and may expose sensitive content, whereas committing to a trace's structure (its nodes, edges, and ordering) may be enough to detect deletion and reordering without disclosing tool arguments or memory contents. Second, how should assurance interact with the privacy requirements above? Trace minimization and tamper-evidence pull in opposite directions, since selective disclosure can remove exactly the material a verifier would check. Third, how should assurance interact with *origin*? An attested trace can show that recorded procedural edges were not altered after the fact, but does not establish whether an inferred SUPPORT edge is correct. Assurance and semantic correctness are independent axes: an attested trace can still contain an incorrect inferred edge, so attestation should not be read as evidence of correctness.

Overall, provenance-aware agents require more than richer logs. They require interoperable schemas, semantic claim-level attribution, memory and multi-agent lineage, runtime enforcement with recovery, assured traces, and benchmarks that evaluate traceability, auditability, safety, privacy, and recoverability as first-class capabilities.

8 Conclusion

LLM agents are moving beyond isolated text generation toward systems that retrieve evidence, plan intermediate steps, invoke tools, update memory, interact with environments, and coordinate with other agents. This shift makes final-answer correctness insufficient for trustworthiness: reliable agent systems must also expose where answers come from, which evidence supports them, how tools and intermediate actions contribute to them, and where failures or unsafe influences enter the execution process. This survey positions evidence tracing and execution provenance as a unified perspective for studying these questions across the agent lifecycle. We reviewed key provenance representations, including structured logs, execution graphs, evidence graphs, tool dependency graphs, memory provenance, and multi-agent communication traces, and discussed their roles in attribution, debugging, verification, safety enforcement, evaluation, and recovery.

Our review shows that the field has made important progress, but remains fragmented. Existing systems often record sources or tool calls, but the capabilities that build on these records remain fragmented across the reviewed systems, with limited integration of claim-level support, influence tracking, memory lineage, cross-agent provenance, and adversarially robust tracing. No benchmark in our catalog evaluates the complete provenance chain, and the reviewed descriptions do not establish tamper-evident or attested trace guarantees. Looking forward, evidence tracing and execution provenance should become a first-class infrastructure layer for reliable agent systems. We hope this survey helps consolidate this emerging research space and motivates the development of traceable, auditable, and recoverable LLM agents.

References

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves

to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=zAdUB0aCTQ.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=oKn9c6ytLx.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=roNSXZpUDN.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of LM agents with an LM-emulated sandbox. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=GEcwtMk1uA.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=m1YYAQjO3w.

Sanidhya Vijayvargiya, Aditya Bharat Soni, Xuhui Zhou, Zora Zhiruo Wang, Nouha Dziri, Graham Neubig, and Maarten Sap. OpenAgentSafety: A comprehensive framework for evaluating real-world AI agent safety. In *International Conference on Learning Representations*, 2026.

Xuanjun Zong, Zhiqi Shen, Lei Wang, Yunshi Lan, and Chao Yang. MCP-SafetyBench: A benchmark for safety evaluation of large language models with real-world MCP servers. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2512.15163.

Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. TRAIL: Trace reasoning and agentic issue localization. *arXiv preprint arXiv:2505.08638*, 2025. URL https://arxiv.org/abs/2505.08638.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 38, 2025. doi:10.52202/085713-4082.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. AgenTracer: Who is inducing failure in the LLM agentic systems? In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2509.03312.

Fanqi Kong, Ruijie Zhang, Huaxiao Yin, Guibin Zhang, Xiaofei Zhang, Ziang Chen, Zhaowei Zhang, Xiaoyuan Zhang, Song-Chun Zhu, and Xue Feng. Aegis: Automated error generation and attribution for multi-agent systems. In *International Conference on Learning Representations (ICLR)*, 2026.

Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. Enabling large language models to generate text with citations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6465–6488, Singapore, 2023. Association for Computational Linguistics. doi:10.18653/v1/2023.emnlp-main.398.

Tobias Schreieder, Tim Schopf, and Michael Färber. Attribution, citation, and quotation: A survey of evidence-based text generation with large language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 30956–31000, 2026. URL https://aclanthology.org/2026.acl-long.1430/. arXiv:2508.15396.

Liming Dong, Qinghua Lu, and Liming Zhu. AgentOps: Enabling observability of LLM agents. *arXiv preprint arXiv:2411.05285*, 2024. URL https://arxiv.org/abs/2411.05285.

Adam AlSayyad, Kelvin Yuxiang Huang, and Richik Pal. AgentTrace: A structured logging framework for agent system observability. *arXiv preprint arXiv:2602.10133*, 2026. URL https://arxiv.org/abs/2602.10133. Poster, AAAI 2026 TrustAgent Workshop (non-archival).

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design. In *IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, 2026. URL https://arxiv.org/abs/2503.18813. arXiv:2503.18813.

Manuel Costa, Boris Köpf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin. Securing AI agents with information-flow control. *arXiv preprint arXiv:2505.23643*, 2025. URL https://arxiv.org/abs/2505.23643.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19724–19731, 2024. doi:10.1609/aaai.v38i17.29946.

World Wide Web Consortium. PROV-DM: The PROV data model. W3C Recommendation, 2013. URL https://www.w3.org/TR/prov-dm/.

OpenTelemetry. OpenTelemetry specification: Traces. Online documentation, 2026. URL https://opentelemetry.io/docs/specs/otel/.

Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975. doi:10.1109/PROC.1975.9939.

Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976. doi:10.1145/360051.360056. URL https://doi.org/10.1145/360051.360056.

Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003. doi:10.1109/JSAC.2002.806121.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. FEVER: A large-scale dataset for fact extraction and VERification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 809–819. Association for Computational Linguistics, 2018. doi:10.18653/v1/N18-1074.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12076–12100. Association for Computational Linguistics, 2023. URL https://aclanthology.org/2023.emnlp-main.741/.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, 2023. doi:10.1145/3586183.3606763.

Rohan Sequeira, Stavros Damianakis, Umar Iqbal, and Konstantinos Psounis. Agent-Sentry: Bounding LLM agents via execution provenance. *arXiv preprint arXiv:2603.22868*, 2026. URL https://arxiv.org/abs/2603.22868.

Yuandao Cai, Wensheng Tang, Cheng Wen, and Shengchao Qin. Ghost in the agent: Redefining information flow tracking for LLM agents. *arXiv preprint arXiv:2604.23374*, 2026. URL https://arxiv.org/abs/2604.23374.

Renan Souza, Amal Gueroudji, Stephen DeWitt, Daniel Rosendo, Tirthankar Ghosal, Robert Ross, Prasanna Balaprakash, and Rafael Ferreira Da Silva. PROV-AGENT: Unified provenance for tracking AI agent interactions in agentic workflows. In *2025 IEEE International Conference on eScience (eScience)*, pages 467–473. IEEE, 2025.

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

Yi Nian, Aojie Yuan, Haiyue Zhang, Jiate Li, Li Li, Xiyang Hu, Hua Wei, Xiongye Xiao, Chaowei Xiao, and Yue Zhao. Auditable agents. *arXiv preprint arXiv:2604.05485*, 2026.

Yusheng Zheng, Yiwei Yang, Wei Zhang, and Andi Quinn. ACRFence: Preventing semantic rollback attacks in agent checkpoint-restore. *arXiv preprint arXiv:2603.20625*, 2026.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 2025. doi:10.1145/3748302.

Xixun Lin, Yucheng Ning, Jingwen Zhang, Yan Dong, Yilong Liu, Yongxuan Wu, Xiaohua Qi, Nan Sun, Yanmin Shang, Kun Wang, Pengfei Cao, Qingyue Wang, Lixin Zou, Xu Chen, Chuan Zhou, Jia Wu, Peng Zhang, Qingsong Wen, Shirui Pan, Bin Wang, Yanan Cao, Kai Chen, Songlin Hu, and Li Guo. LLM-based agents suffer from hallucinations: A survey of taxonomy, methods, and directions. *arXiv preprint arXiv:2509.18970*, 2025. URL https://arxiv.org/abs/2509.18970.

Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. Graph retrieval-augmented generation: A survey. *ACM Transactions on Information Systems*, 44(2):1–52, 2025.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *First conference on language modeling*, 2024.

Dongyu Ru, Lin Qiu, Xiangkun Hu, Tianhang Zhang, Peng Shi, Shuaichen Chang, Cheng Jiayang, Cunxiang Wang, Shichao Sun, Huanyu Li, Zizhao Zhang, Binjie Wang, Jiarong Jiang, Tong He, Zhiguo Wang, Pengfei Liu, Yue Zhang, and Zheng Zhang. RAGChecker: A fine-grained framework for diagnosing retrieval-augmented generation. In *Advances in Neural Information Processing Systems*, 2024.

Kevin Wu, Eric Wu, Ally Cassasola, Angela Zhang, Kevin Wei, Teresa Nguyen, Sith Riantawan, Patricia Shi Riantawan, Daniel E. Ho, and James Zou. An automated framework for assessing how well LLMs cite relevant medical references. *Nature Communications*, 16:3615, 2025. URL https://www.nature.com/articles/s41467-025-58551-6. Earlier version: arXiv:2402.02008.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. In *Proceedings of the IEEE/ACM 48th International Conference on Software Engineering (ICSE)*, 2026. URL https://arxiv.org/abs/2503.18666. arXiv:2503.18666.

Anna Martin-Boyle, Cara Leckey, Martha Brown, and Harmanpreet Kaur. PaperTrail: A claim-evidence interface for grounding provenance in LLM-based scholarly Q&A. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, pages 1–25, 2026.

Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. RAGAs: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 150–158, St. Julians, Malta, 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.eacl-demo.16/.

Joel Rorseth, Parke Godfrey, Lukasz Golab, Divesh Srivastava, and Jarek Szlichta. LADYBUG: An LLM agent DeBUGger for data-driven applications. In *Proceedings of the 28th International Conference on Extending Database Technology*, pages 1082–1085, 2025. URL https://openproceedings.org/2025/conf/edbt/paper-313.pdf.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. *Advances in neural information processing systems*, 36:51991–52008, 2023.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2025.

Christoph Bühler, Matteo Biagiola, Luca Di Grazia, and Guido Salvaneschi. AgentBound: Securing execution boundaries of AI agents. In *Proceedings of the ACM on Software Engineering (FSE)*, 2026. URL https://doi.org/10.1145/3808103. arXiv:2510.21236.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10471–10506. Association for Computational Linguistics, 2024. URL https://doi.org/10.18653/v1/2024.findings-acl.624.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, pages 79–90. Association for Computing Machinery, 2023. doi:10.1145/3605764.3623985. URL https://doi.org/10.1145/3605764.3623985.

Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, Leo Yu Zhang, and Yang Liu. Prompt injection attack against LLM-integrated applications. *arXiv preprint arXiv:2306.05499*, 2023. URL https://arxiv.org/abs/2306.05499.

Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. The instruction hierarchy: Training LLMs to prioritize privileged instructions. *arXiv preprint arXiv:2404.13208*, 2024. URL https://arxiv.org/abs/2404.13208.

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. StruQ: Defending against prompt injection with structured queries. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 2383–2400, 2025.

Traian Rebedea, Razvan Dinu, Makesh Narsimhan Sreedhar, Christopher Parisien, and Jonathan Cohen. NeMo guardrails: A toolkit for controllable and safe LLM applications with programmable rails. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 431–445, Singapore, December 2023. Association for Computational Linguistics. doi:10.18653/v1/2023.emnlp-demo.40. URL https://aclanthology.org/2023.emnlp-demo.40/.

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and Madian Khabsa. Llama Guard: LLM-based input-output safeguard for human-AI conversations. *arXiv preprint arXiv:2312.06674*, 2023. URL https://arxiv.org/abs/2312.06674.

Pengfei Du. Memory for autonomous LLM agents: Mechanisms, evaluation, and emerging frontiers. *arXiv preprint arXiv:2603.07670*, 2026. URL https://arxiv.org/abs/2603.07670.

Chang Yang, Chuang Zhou, Yilin Xiao, Su Dong, Luyao Zhuang, Yujing Zhang, Zhu Wang, Zijin Hong, Zheng Yuan, Zhishang Xiang, Shengyuan Chen, Huachi Zhou, Qinggang Zhang, Ninghao Liu, Jinsong Su, Xinrun Wang, Yi Chang, and Xiao Huang. Graph-based agent memory: Taxonomy, techniques, and applications. *arXiv preprint arXiv:2602.05665*, 2026. URL https://arxiv.org/abs/2602.05665.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. In *European Conference on Artificial Intelligence (ECAI)*, 2025. arXiv:2504.19413.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. In *Advances in Neural Information Processing Systems*, volume 36, pages 74530–74543, 2023.

Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. AgentPoison: Red-teaming LLM agents via poisoning memory or knowledge bases. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/eb113910e9c3f6242541c1652e30dfd6-Abstract-Conference.html.

Shen Dong, Shaochen Xu, Pengfei He, Yige Li, Jiliang Tang, Tianming Liu, Hui Liu, and Zhen Xiang. Memory injection attacks on LLM agents via query-only interaction. In *Advances in Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=QINnsnppv8.

Hanling Tian, Gengyu Zhang, Zeyang Sha, Jingying Wang, Yuhang Liu, Zhehao Huang, Kun Yang, and Xiaolin Huang. InjecMEM: Memory injection attack on LLM agent memory systems. In *Conference on Language Modeling (COLM)*, 2026. URL https://arxiv.org/abs/2608.23471.

Sidharth Pulipaka, Stanislau Hlebik, Leonidas Raghav, Sahar Abdelnabi, Vyas Raina, Ivaxi Sheth, and Mario Fritz. Hidden in memory: Sleeper memory poisoning in LLM agents, 2026. URL https://arxiv.org/abs/2605.15338.

Bo Wang, Weiyi He, Shenglai Zeng, Zhen Xiang, Yue Xing, Jiliang Tang, and Pengfei He. Unveiling privacy risks in LLM agent memory. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 25241–25260, Vienna, Austria, July 2025. Association for Computational Linguistics. doi:10.18653/v1/2025.acl-long.1227. URL https://aclanthology.org/2025.acl-long.1227/.

Qianshan Wei, Tengchao Yang, Yaochen Wang, Xinfeng Li, Lijun Li, Zhenfei Yin, Yi Zhan, Thorsten Holz, Zhiqiang Lin, and XiaoFeng Wang. A-MemGuard: A proactive defense framework for LLM-based agent memory. In *International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2510.02373.

Wei Zou, Mingwen Dong, Miguel Romero Calvo, Shuaichen Chang, Jiang Guo, Dongkyu Lee, Xing Niu, Xiaofei Ma, Yanjun Qi, and Jiarong Jiang. Poison once, exploit forever: Environment-injected memory poisoning attacks on web agents. *arXiv preprint arXiv:2604.02623*, 2026. doi:10.48550/arXiv.2604.02623.

Jon Saad-Falcon, Omar Khattab, Christopher Potts, and Matei Zaharia. ARES: An automated evaluation framework for retrieval-augmented generation systems. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 338–354, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi:10.18653/v1/2024.naacl-long.20. URL https://aclanthology.org/2024.naacl-long.20/.

Cheng Niu, Yuanhao Wu, Juno Zhu, Siliang Xu, Kashun Shum, Randy Zhong, Juntong Song, and Tong Zhang. RAGTruth: A hallucination corpus for developing trustworthy retrieval-augmented language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 10862–10878. Association for Computational Linguistics, 2024. doi:10.18653/v1/2024.acl-long.585. URL http://papers.nips.cc/paper_files/paper/2024/hash/27245589131d17368cccdfa990cbf16e-Abstract-Datasets_and_Benchmarks_Track.html.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 13851–13870, 2024.

Zexue He, Yu Wang, Churan Zhi, Yuanzhe Hu, Tzu-Ping Chen, Lang Yin, Ze Chen, Tong Arthur Wu, Siru Ouyang, Zihan Wang, Jiaxin Pei, Julian McAuley, Yejin Choi, and Alex Pentland. MemoryArena: Benchmarking agent memory in interdependent multi-session agentic tasks. In *International Conference on Machine Learning (ICML)*, 2026.

OpenLineage Contributors. OpenLineage Specification. https://github.com/OpenLineage/OpenLineage/blob/main/spec/OpenLineage.md, 2026. Accessed: 2026-05-25.

A Additional Taxonomy Mappings

This appendix provides supplementary material for the taxonomy in Section 2. The main text keeps the taxonomy compact by using Figure 2 and Table 1 as the backbone, while the mapping below makes explicit how the proposed agent-provenance vocabulary relates to standard provenance modeling. The concrete provenance-graph example is discussed in Section 3, where it serves as a running example for provenance representation rather than as appendix-only material.

A.1 Mapping to W3C PROV-DM

Table 9 makes the relationship to the W3C PROV data model explicit. Node types align with PROV entities, activities, and agents, so much of agent execution can reuse a standard provenance vocabulary. The relations are then grouped into three tiers. Use, Generate and Derive are *inherited*: their PROV semantics are retained unchanged. Depend-on, Trigger, Invalidate and Update are *specialized*: PROV expresses each in a weaker form, and agent execution requires the stronger reading given in the table. Only the third tier is *novel*, and it is what PROV does not directly express: agent execution introduces semantic relations such as Support and Contradict, which depend on comparing the content of evidence and claims rather than only on bookkeeping about how artifacts were produced. In this sense, agent provenance extends, rather than merely instantiates, classical provenance models.

B Operational Semantics of Provenance Relations

Section 2 introduces nine provenance relations. This appendix makes them operational, so that two independent annotators reading the same execution trace should produce the same typed graph. For each relation we give (i) a *signature* constraining which node types it may connect, (ii) its *direction*, (iii) whether it carries a *temporal* requirement, (iv) the *necessary and sufficient condition* under which it holds, and (v) a positive example, a negative or boundary example, and the relation it is most often confused with together with the test that separates them. Section B.3 then works through the two boundary cases on which annotators most often disagree. Section C then states formally what it means for evidence tracing to be a projection of execution provenance.

B.1 Node Types and Unit Roles

The distinction between evidence units and execution units in Section 2.1 becomes sharp once artifacts are separated from occurrences, following the entity/activity split of W3C PROV-DM (World Wide Web

Table 9: Correspondence between the W3C PROV data model (World Wide Web Consortium, 2013) and the agent-provenance model of this survey, grouped into inherited, specialized, and novel relations.

W3C PROV construct	**Agent-provenance counterpart**	**Notes and agent-specific extension**
Node types (PROV-aligned)		
Entity	Evidence / execution artifact	Documents, passages, tool outputs, memory items, generated claims, messages.
Activity	Execution unit	Reasoning steps, retrieval and tool calls, memory operations, environment actions.
Agent	Agent	LLM, tool/API, user, or sub-agent; multi-agent roles and delegation become first-class.
Tier 1: Inherited relations (PROV semantics retained)		
`used`	USE	A step consumed an artifact. Declarative and instrumentable, exactly as in PROV; agent provenance adds trust and taint labels on the consumed value.
`wasGeneratedBy`	GENERATE	A tool output is generated by a tool call; a claim by a reasoning step.
`wasDerivedFrom`	DERIVE	Memory derived from a document, or a claim from a table; captures summarization and transformation, not only copying.
Tier 2: Specialized relations (PROV-compatible notions refined for agent execution)		
`used` (strengthened)	DEPEND-ON	PROV `used` records consumption; agent DEPEND-ON additionally asserts *counterfactual* dependence, established by replay or ablation. The two are deliberately distinct relations here, since a step may consume ten artifacts and depend on one. Step-on-step dependence is outside `used` altogether, since `used` relates an activity to an entity; the nearest PROV construct is `wasInformedBy`, which records that one activity was informed by another without asserting counterfactual dependence.
`wasInformedBy`	TRIGGER	An observation triggers a tool call; an error triggers replanning.
`wasInvalidatedBy`	INVALIDATE	PROV invalidation ends an entity's existence; agent INVALIDATE marks a claim, plan, or memory item as epistemically invalid while the record persists.
`wasRevisionOf`	UPDATE	Memory, external-state, or delivered-output updates across turns or sessions; agent UPDATE identifies the superseded version by *location* rather than by content similarity.
Tier 3: Novel relations and units (no PROV counterpart)		
—	SUPPORT	Evidence supports a claim, decision, or action; central to attribution, but PROV has no support/justification semantics.
—	CONTRADICT	A tool output contradicts a memory item or retrieved passage; requires semantic comparison, not provenance bookkeeping.
—	Semantic units	Generated claims, tool-call rationales, natural-language observations, inter-agent messages, and trust labels, which PROV treats as opaque entities.

Consortium, 2013). An *evidence unit* is an artifact: it carries a content field $\rho(v)$ that can be compared with the content of another node. An *execution unit* is an occurrence: it carries an actor and a time interval, and its identity is *that it happened*, not *what it says*. Table 10 assigns every node type in the taxonomy to exactly one role.

Two consequences follow, and they answer the question of when an object is an evidence unit, an execution unit, or both.

1. **At graph level the roles are disjoint.** A tool output is an evidence unit; the tool call that produced it is an execution unit; the two are distinct nodes joined by a GENERATE edge. The intuition that a tool output is "both" reflects the granularity of a flat log, in which a single record carries the call, its arguments, and its result. Converting a log to a provenance graph therefore always involves splitting such records.

Table 10: Node types and the unit role assigned to each.

Role	Node types	Content ρ	Actor + interval	Eligible as
Evidence unit	User query or instruction, retrieved document or passage, environment observation, tool output, memory item, generated tool parameter, policy, tool manifest, inter-agent message, intermediate claim, final answer	✓	✗	Endpoint of every semantic and derivational relation, and of UPDATE; source of TRIGGER; target of USE, GENERATE, DEPEND-ON. Which *end* is permitted in each case is fixed by the signatures in Table 11: only assertive units may be the target of SUPPORT.
Execution unit	Reasoning step, retrieval call, tool call, memory read or write, environment action, delegation step	✗	✓	Source of USE, GENERATE, DEPEND-ON; source and target of TRIGGER; target of DEPEND-ON; target of SUPPORT and INVALIDATE via its precondition (Section B.2).
Actor	LLM, tool or API, user, sub-agent	✗	✗	Attribution of responsibility only; never an endpoint of a semantic relation.

2. **Roles are stable, but the *part of the node an edge reads* is not.** A semantic edge incident on a node reads its content $\rho(v)$; a procedural edge reads its occurrence. This is why an intermediate claim can be the target of SUPPORT (a statement about its content) and, in the same graph, the source of TRIGGER to a tool call (a statement about its occurrence) without any ambiguity of role.

B.2 Relation Signatures, Directionality, and Temporal Semantics

We write $\mathcal{E}$ for the set of evidence units, $\mathcal{X}$ for execution units, and $\mathcal{C} \subset \mathcal{E}$ for the *assertive* evidence units, namely intermediate claims and final answers. A relation is *semantic* if whether it holds is determined by the contents of its endpoints, and *procedural* if whether it holds is determined by the execution, independently of whether the contents are true. Table 11 records the signature of each relation; Table 12 records the condition under which it holds together with worked examples.

The "Temporal" column of Table 11 states the ordering each relation requires between its endpoints. We write $t(v)$ for the timestamp of an evidence unit and, for an execution unit, for the *start* of its interval. An artifact produced by an activity is timestamped at that activity's completion, so GENERATE admits equality; USE also admits equality, since a step may begin at the instant its input was produced. INVALIDATE is stated against $t_{\text{est}}(t)$, the time at which the target was *established* (created for an artifact, scheduled for a step), because a step may be invalidated before it starts. The required order is not always source-before-target: for INVALIDATE, DEPEND-ON, UPDATE, and USE, the source is the later node, because the invalidator, the dependent step, the new version, and the consuming activity all come after what they act on.

One further label appears in provenance graphs but is not one of the nine: SUPPORT-CANDIDATE, generated by the projection of Appendix C. Its signature is $\mathcal{E} \to \mathcal{C}$, it is asymmetric, its temporal constraint is $t(s) \leq t(t)$, inherited from the edges of its witness path and strict when the path contains a DERIVE edge, and its origin is always DERIVED, the value reserved for edges computed rather than observed. It asserts availability of content, never support, and no instrumented or annotated graph contains one.

Table 11: Signature, direction, and temporal ordering of the nine provenance relations.

Relation	Signature (source → target)	Direction	Temporal	Family and algebra
SUPPORT	$\mathcal{E} \to \mathcal{C} \cup \mathcal{X}$	Supporter → supported	None	Semantic support. Asymmetric; *not* transitive.
CONTRADICT	$\mathcal{E} \leftrightarrow \mathcal{E}$	Symmetric (unordered pair)	None	Semantic support. Symmetric; not transitive.
INVALIDATE	$\mathcal{E} \to \mathcal{C} \cup \{\text{memory item}\} \cup \mathcal{X}$	Invalidator → invalidated	$t(s) > t_{\text{est}}(t)$	Semantic support (epistemic status). Asymmetric.
DERIVE	$\mathcal{E} \to \mathcal{E}$	Origin → derived	$t(s) < t(t)$	Content transformation. Asymmetric; transitive.
DEPEND-ON	$\mathcal{X} \to \mathcal{E} \cup \mathcal{X}$	Dependent step → dependency	$t(s) > t(t)$	Causal (counterfactual: parameterization). Asymmetric; not transitive in general.
TRIGGER	$\mathcal{E} \cup \mathcal{X} \to \mathcal{X}$	Cause → activated step	$t(s) < t(t)$	Causal (counterfactual: occurrence). Asymmetric; not transitive.
UPDATE	$\mathcal{E} \to \mathcal{E}$ (same identified location)	New version → superseded version	$t(s) > t(t)$	Procedural, state succession (instrumented). Asymmetric.
USE	$\mathcal{X} \to \mathcal{E}$	Activity → consumed artifact	$t(s) \geq t(t)$	Procedural, declarative (instrumented; PROV `used`).
GENERATE	$\mathcal{X} \to \mathcal{E}$	Activity → produced artifact	$t(s) \leq t(t)$	Procedural, declarative (instrumented; PROV `wasGeneratedBy`). Functional: each artifact has at most one generator within a run.

Table 12: Necessary and sufficient conditions, worked examples, and boundary cases. The final column names the relation each is most often confused with, together with the test that separates them. s denotes the source node and t the target; $\rho(\cdot)$ denotes node content.

Relation	Holds if and only if	Positive / negative example	Confusable with, and the test
SUPPORT	Taking $\rho(s)$ as given raises the warrant for $\rho(t)$, or for the precondition of step t, above a threshold θ fixed in advance by the annotation protocol. The relation is a property of the two contents, not of the execution; given θ it is adjudicable by a third party who reads only $\rho(s)$ and $\rho(t)$. Because θ is task-specific and not derivable from the graph, a protocol that leaves it unstated will not reproduce, which is why inter-annotator agreement bounds any metric built on this relation (Section 6.4). *Not necessary*: that the agent read s. *Not sufficient*: that the agent cited s, that s was in context, or that s changed the output.	+ Passage "Q3 revenue was $2.1B" supports the claim "Q3 revenue exceeded $2B". − A passage on the same topic that does not entail the claim: relevant and possibly cited, but not supporting (Wu et al., 2025).	DEPEND-ON. Ablate s: if the claim would still be warranted, SUPPORT is a property of content and survives; if the *step* would have been parameterized differently, that is DEPEND-ON. Both may hold; they are not alternatives.
CONTRADICT	$\rho(s)$ and $\rho(t)$ cannot both hold under the task's interpretation, at the same time index.	+ Tool output "balance = 0" contradicts memory item "account is funded". − Two figures at different granularity or different time indices that are jointly satisfiable: no contradiction, although the older figure may be stale.	INVALIDATE. CONTRADICT is a symmetric statement about content and asserts nothing about which endpoint survives.
INVALIDATE	(i) s contradicts t or establishes that a precondition of t no longer holds; (ii) s is at least as authoritative as the basis of t, and $t(s) > t_{\text{est}}(t)$; (iii) after s, t may no longer be used as a warrant. The record of t persists.	+ A 404 response invalidates the planned tool call to that endpoint, whose precondition no longer holds. − A conflicting but less authoritative source: record CONTRADICT only, and leave adjudication to a later step.	CONTRADICT (see above) and UPDATE. INVALIDATE marks the old item unusable; UPDATE installs the new one. They commonly co-occur as two distinct edges.

continued on next page

Table 12 (continued)

Relation	Holds if and only if	Positive / negative example	Confusable with, and the test
DERIVE	$\rho(t)$ was produced by transforming $\rho(s)$: copying, extraction, summarization, translation, or aggregation. Derivation establishes content lineage, not the correctness of the transformation or semantic support. *Not sufficient*: topical similarity, or co-presence in the context window.	+ A memory item derived from a conversation turn; a claim derived from a table row. − A claim produced from parametric knowledge while a document happened to be in context: no DERIVE edge.	SUPPORT. DERIVE says where content came from; SUPPORT says whether content warrants. A claim mis-summarized from a passage is DERIVE *without* SUPPORT, which is precisely the attributable-hallucination case (Min et al., 2023; Ru et al., 2024).
DEPEND-ON	Had t been absent or different, step s would have been issued with different arguments or would otherwise have behaved differently. Whether the step occurred at all is TRIGGER, not DEPEND-ON. Established by replay, ablation, or fault injection (Zhang et al., 2026), not by reading content.	+ A tool call whose argument string was extracted from a retrieved page. − A document present in the context window that provably does not change the arguments under replay: USE but not DEPEND-ON.	USE. On dependencies whose target is an artifact, USE is the observable over-approximation that the harness emits and DEPEND-ON is the counterfactual subset inside it. Dependencies on another *step* have no USE counterpart at all, and can be established only by replay.
TRIGGER	The occurrence of s is what caused step t to be scheduled at all.	+ A tool error triggers replanning; a new observation triggers a diagnostic call. − An input that only changed the arguments of an already-scheduled step.	DEPEND-ON. The test is *whether* versus *how*: TRIGGER governs whether the step happened, DEPEND-ON governs how it was parameterized. Boundary case: a step that would have occurred regardless but was brought forward or delayed by s is *not* triggered by it, since scheduling time is an attribute rather than an occurrence; record the re-timing in the timestamps and leave the edge out.
UPDATE	s and t occupy the same identified location (a store slot, memory key, external state variable, or an externally identified output slot such as the response delivered for a given user turn), and s supersedes t from its timestamp onward. Two artifacts occupy the same location iff they are addressed by the same identifier within the same scope, so a revised answer to one turn updates that turn's response while an answer to a later turn is a different location. Identity of location, not similarity of content, is what the relation asserts.	+ A user-preference memory revised in a later session. − A second, independent memory item about the same topic that does not overwrite the first: two items, no UPDATE.	DERIVE. UPDATE requires identity continuity of the slot; DERIVE does not. A summary written to a *new* key is DERIVE; overwriting the old key is both.
USE	The harness supplied artifact t to activity s. A recorded fact about the execution; asserts no influence and no support.	+ A reasoning step uses all ten passages returned by a retrieval call. − A passage retrieved but filtered out before the step: not used.	DEPEND-ON (see above). Reporting USE as if it were SUPPORT is a common over-claim in citation-style evaluation.
GENERATE	Activity s produced artifact t, whose timestamp equals the completion of s.	+ A tool call generates a tool output; a reasoning step generates a claim. − A corpus document that pre-existed the run: it enters the graph with an external-origin attribute and no generator.	DERIVE. GENERATE connects an activity to an artifact; DERIVE connects two artifacts. Both are typically present: a reasoning step GENERATEs a claim that DERIVEs from a passage.

B.3 Separating Semantic Support from Procedural Influence

The four notions that Table 11 keeps apart are frequently conflated in trace-based evaluation, so we state the discriminating tests directly.

- **Semantic support** (SUPPORT, CONTRADICT, and the first clause of INVALIDATE) is decided by reading the contents of the two endpoints. It is agent-independent: a passage supports a claim whether or not any agent ever retrieved it. When the target of SUPPORT is an execution unit, which carries no content of its own, the content read is the step's *precondition*: the proposition that must

hold for the step to be justified, which the annotation protocol is required to make explicit before the edge can be adjudicated.

- **Content derivation** (DERIVE) is decided by asking whether the target's content is a transformation of the source's. It is agent-dependent and content-level at once, which is why it can hold while SUPPORT fails.
- **Causal dependence** (TRIGGER, DEPEND-ON) is decided counterfactually, by removing or perturbing the source and re-running, unless the harness dispatched the step on an event, in which case the edge is logged directly. It says nothing about whether the content is true.
- **Procedural dependence** (USE, GENERATE, UPDATE) is read directly from what the harness emitted, except that UPDATE is inferred when the store does not version its keys (Table 13). It is the observable layer beneath causal dependence.
- **Temporal succession** is deliberately *not* a relation. Adjacency in a log is an artifact of ordering, not a dependency, so ordering is carried by node timestamps and never by an edge. Temporal adjacency alone does not establish dependence: without an explicit relation, the provenance graph records no dependency between two steps, which means that none has been established, not that none exists.

Two boundary cases deserve explicit treatment because they are the ones on which annotators most often disagree.

Case 1: a tool output changes the final decision without justifying the final claim. Record USE(step, output) because the harness passed it; record DEPEND-ON(action, output) if ablating the output changes the action; record DERIVE only if the answer's content is a transformation of the output's content; record SUPPORT only if a third party reading the two contents judges that the output warrants the claim. The correct annotation is frequently {USE, DEPEND-ON} with neither DERIVE nor SUPPORT, and the resulting graph says exactly the right thing: the decision was influenced by something that did not justify it.

Case 2: an incorrect memory item causally triggers an action. Record TRIGGER(memory item, action), because the relation is causal and holds regardless of the item's truth. Do *not* record SUPPORT merely because the item drove the action: SUPPORT requires a third party to judge that the item's content, taken as given, warrants the action's precondition, and an item that only prompted the step need not do so. When the error is discovered, add CONTRADICT(new evidence, memory item) and INVALIDATE(new evidence, dependent claim). The graph then simultaneously records that the item drove the execution and that it was later shown to be false, which is the distinction the taxonomy exists to preserve.

B.4 Which Relations Are Observable, and Which Must Be Inferred

Provenance relations differ not only in meaning but in how they can be obtained. We therefore annotate every edge with an *origin* attribute. Three values apply to edges of a recorded execution: INSTRUMENTED (emitted by the agent harness or runtime), INFERRED (assigned post hoc by a model or a human annotator), and DECLARED (asserted by the agent itself, for example a self-reported citation). A fourth, DERIVED, is reserved for edges computed from a graph by a defined procedure rather than obtained from the execution at all; the only label carrying it is SUPPORT-CANDIDATE, produced by the projection of Appendix C. Table 13 states which relations admit which origins. The practical consequence is that a provenance graph is heterogeneous in reliability: its procedural skeleton can be logged, but its semantic layer is a judgment, and evaluation must treat the two differently.

Orthogonally to origin, an edge may carry an *assurance* attribute recording whether the trace it came from is merely logged, or additionally protected against post-hoc modification. Among the systems surveyed in Table 3, none reports tamper-evidence, signing, or attestation for its traces, so provenance is trustworthy only to the extent that the party holding the log is (Ojewale et al., 2026; Nian et al., 2026; Zheng et al., 2026). Section 7 treats this as an open problem rather than a solved dimension.

Table 13: Origin of each of the nine provenance relations. SUPPORT-CANDIDATE is omitted, since its origin is always DERIVED.

Relation	Available origins	How it is obtained in practice
USE, GENERATE	Instrumented	Emitted directly by the agent loop, tool router, or memory store; reproducible from logs alone (Dong et al., 2024; AlSayyad et al., 2026; Souza et al., 2025).
UPDATE	Instrumented, Inferred	Instrumented when the store versions its keys; otherwise inferred by matching successive items to the same slot.
TRIGGER	Instrumented, Inferred	Instrumented when the harness dispatches on an event, such as retry-on-error; inferred when the model itself chose to act on an observation.
DEPEND-ON	Inferred	Not directly observable. Obtained by replay, ablation, fault injection, or information-flow tracking; USE over-approximates it only for dependencies on artifacts (Zhang et al., 2026; Costa et al., 2025; Cai et al., 2026).
SUPPORT, CONTRADICT, INVALIDATE	Inferred, Declared	Never directly observable. Require semantic judgment by a human annotator or model judge, and therefore require an annotation protocol and reported inter-annotator agreement (Gao et al., 2023; Min et al., 2023; Thorne et al., 2018; Ru et al., 2024).
DERIVE	Inferred, Declared	Inferred by content comparison, or declared by an agent that reports its own sources for a summary or memory write.

B.5 Derived Predicates: Cited and Omitted

Section 3.2 refers to CITED and OMITTED. These are not additional taxonomy relations; they are predicates derived from the nine above, and defining them requires the *origin* attribute of Section B.4, because a citation is an agent's *assertion* that support holds, which is a different thing from support holding.

Write avail(d) for "artifact d was available to the run", that is, d was generated within the run or supplied to some step: $\text{avail}(d) \equiv \exists x.\, \textsc{Generate}(x, d) \vee \textsc{Use}(x, d) \vee \text{external}(d)$, where external($d$) marks an artifact that pre-existed the run and entered the graph without a generator, such as a corpus document. Distinguish two kinds of SUPPORT edge by origin:

$$\textsc{Cited}(d, c) \;\equiv\; \exists\, e.\; \lambda(e) = \textsc{Support},\; e : d \to c,\; \text{origin}(e) = \textsc{Declared},$$

$$\textsc{Supported}(d, c) \;\equiv\; \exists\, e.\; \lambda(e) = \textsc{Support},\; e : d \to c,\; \text{origin}(e) = \textsc{Inferred}.$$

CITED records that the agent asserted the link; SUPPORTED records that an annotator or judge adjudicated it. The two are independent, and their disagreement is exactly what citation-quality evaluation measures:

$$\text{citation error} \;\equiv\; \textsc{Cited}(d, c) \wedge \neg \textsc{Supported}(d, c), \qquad \text{attribution gap} \;\equiv\; \textsc{Supported}(d, c) \wedge \neg \textsc{Cited}(d, c),$$

$$\textsc{Omitted}(d, a) \;\equiv\; \text{avail}(d) \;\wedge\; \neg \exists\, c \in \text{claims}(a).\; \textsc{Supported}(d, c),$$

where claims(a) denotes the decomposition of answer a into assertive units, which any claim-level protocol must fix. A source that is cited but does not support is therefore a citation error and not a contradiction in terms (Gao et al., 2023; Wu et al., 2025; Martin-Boyle et al., 2026); OMITTED covers a retrieved-but-unused passage as well as a used one, which is the case Section 3.2 describes. Note that this is the one place where the origin attribute is load-bearing rather than descriptive: without it, an agent's self-report and an adjudicated judgment are the same edge, and citation quality cannot be expressed at all.

B.6 Derivation and Adequacy of the Relation Set

Each of the nine relations is admitted only if it passes two tests. The tests also keep the set open to revision: a relation that fails them, or a query that none of them answers, would change it.

- **Necessity.** Some trust-function query of Section 2.1 (verification, attribution, debugging, safety enforcement, audit, or recovery) becomes unanswerable if the relation is removed or merged into another.

- **Distinctness.** The relation carries a decision test that no other relation's test subsumes. Because the four families of Section B.3 are decided by different kinds of evidence (content comparison, content transformation, counterfactual ablation, and harness records), two relations in different families can never be merged without discarding one of those tests.

Table 14 applies both tests relation by relation. The third column is the operative one: it names what a reader would no longer be able to express, and the fourth names surveyed work that already draws the distinction operationally, so that necessity rests on existing practice.

Alternative decompositions we considered and rejected. Four alternatives are natural. Three of them lose a distinction the trust functions need or add cost without letting the graph answer a new query. The second, an attribute encoding of relation kind, loses no information, and we decline it as a matter of schema design rather than expressiveness.

1. **W3C PROV-DM unextended.** Its relations record how artifacts were produced and never compare content, so SUPPORT and CONTRADICT have no counterpart at all (Section 2.1). Claim–evidence grounding, the concern of an entire literature reviewed in Section 3, would be outside the model.
2. **A single attributed INFLUENCE edge**, with kind recorded as an attribute. Provided the attribute is mandatory and ranges over the same nine kinds, this encoding carries the same information as explicit labels: the projection π of Section C can be restated as a filter on the attribute value rather than on λ, and P3 (that procedural influence has *no* image under π) holds in the same form. The choice is therefore one of schema design, and we prefer explicit relation labels for three practical reasons. First, the signature constraints of Table 11 attach directly to a label, whereas under the attribute encoding they become conditional constraints on attribute values that a validator must enforce separately. Second, a label cannot be omitted, whereas an attribute can be dropped or defaulted by a consumer that does not recognize it; when traces pass between tools, that is the most likely way for "does this justify the claim" and "did this cause the step" to be silently conflated. Third, labeled edges correspond directly to edge types in property-graph stores, and the PROV-compatible ones to the typed relations of W3C PROV-DM (Table 9), so π reduces to selection by edge type in standard query languages. An implementation that adopts a single edge type with a mandatory, validated kind attribute is equivalent for the purposes of this survey.
3. **A finer decomposition**, splitting SUPPORT into entailment, paraphrase, and partial support, or USE into reading and parameterizing. Each split is a *refinement* of a single decision test rather than a new one, and the resulting distinctions are already carried by edge attributes and by the threshold θ of Table 12. Splitting them raises annotation burden and lowers inter-annotator agreement without letting the graph answer a query it could not answer before.
4. **Merging TRIGGER into DEPEND-ON**, on the view that both are counterfactual. They are, but over different questions (occurrence versus parameterization), and runtime enforcement acts on the first while information-flow tracking acts on the second, so the merge would erase the boundary between the two guardrail families surveyed in Section 4.

Candidate relations considered and excluded. Six further candidates were considered while constructing the set and excluded, each on one of the two tests. FOLLOWS, a temporal-succession edge, fails distinctness: order is already carried by node timestamps, and adjacency establishes no semantic, causal, or procedural dependence (Section B.3). CITED and OMITTED fail necessity: both are predicates derived from the nine, and Section B.5 defines them in those terms rather than as edges of their own. RETRIEVED-FROM fails distinctness: a retrieval is an execution unit, so the edge it would add is already USE on the query and GENERATE on the returned passage. RESPONSIBLE-FOR and AUTHORIZED-BY fail necessity only under the current corpus, in which delegation and policy are expressed indirectly; they are the two candidates we regard as open, and the next paragraph states the test they would have to pass.

Scope of the relation set. The set is not proved minimal in any formal sense, and minimality is not a property a relation vocabulary can have absolutely. It is minimal *relative* to two things we state: the six

Table 14: Derivation of the relation set. Each relation is admitted because removing or merging it makes a trust-function query unanswerable, and because at least one surveyed system or benchmark already acts on the distinction it draws.

Relation	**Query it answers**	**What is lost if dropped or merged**	**Already operational in**
SUPPORT	Is this claim warranted by the evidence available? (verification)	Claim-level verification collapses into retrieval logging. No PROV relation compares content, so this cannot be recovered from the base model.	ALCE, FActScore, SourceCheckup, RAGChecker (Gao et al., 2023; Min et al., 2023; Wu et al., 2025; Ru et al., 2024)
CONTRADICT	Which two artifacts cannot both be relied on? (verification, safety)	Treated as negative SUPPORT, conflict between two non-assertive artifacts, such as two memory items or two retrieved passages, becomes unrepresentable, since SUPPORT requires an assertive target.	FEVER, RAGTruth, memory-conflict work (Thorne et al., 2018; Niu et al., 2024; Zhong et al., 2024)
INVALIDATE	Which recorded item ceased to be usable, and from when? (recovery, audit)	A stale or poisoned item can be deleted but not *marked*: recovery loses its audit trail. Distinguished from CONTRADICT, which is symmetric, by being directed and time-stamped: it marks which item ceased to be usable, and from when, while leaving that item in the graph.	A-MemGuard, memory-poisoning defenses (Wei et al., 2026; Chen et al., 2024)
DERIVE	Is this content a transformation of that content? (attribution)	Merged into SUPPORT, the attributable hallucination, a claim mis-summarized from a real passage and hence derived but unsupported, becomes inexpressible, and it is precisely the case attribution benchmarks target.	ALCE, RAGTruth, NeuroTaint (Gao et al., 2023; Niu et al., 2024; Cai et al., 2026)
DEPEND-ON	Would this step have been parameterized differently without that artifact? (debugging, safety)	Merged into USE, availability is mistaken for dependence: every artifact the harness supplied counts as a cause, and ablation-verified influence becomes indistinguishable from mere presence.	NeuroTaint, Fides, AgenTracer (Cai et al., 2026; Costa et al., 2025; Zhang et al., 2026)
TRIGGER	Would this step have occurred at all? (debugging, safety)	Merged into DEPEND-ON, *whether* a step happened collapses into *how* it was parameterized, but a guardrail that blocks a call acts on the former and a taint check on the latter.	AgentSpec, Agent-Sentry, CaMeL (Wang et al., 2026; Sequeira et al., 2026; Debenedetti et al., 2026)
USE	What did the harness supply to this step? (audit)	The loggable skeleton of the graph disappears, and every procedural edge becomes inferred rather than instrumented, which is the distinction Section B.4 rests on.	OpenTelemetry, AgentOps, AgentTrace, PROV-AGENT (OpenTelemetry, 2026; Dong et al., 2024; AlSayyad et al., 2026; Souza et al., 2025)
GENERATE	Which step produced this artifact? (audit, attribution)	Artifacts lose their origin, so no chain runs from a tool call to a downstream claim, and evidence cannot be traced back past the point at which it entered the run.	W3C PROV-DM, AgentOps, PROV-AGENT (World Wide Web Consortium, 2013; Dong et al., 2024; Souza et al., 2025)
UPDATE	Which version of this location superseded which? (memory, recovery)	Merged into DERIVE, the same-location constraint is dropped, so memory lineage cannot be separated from ordinary content copying and versioned stores lose their history.	MemGPT, A-MEM, Mem0 (Packer et al., 2023; Xu et al., 2025; Chhikara et al., 2025)

trust functions of Section 2.1, and the corpus of Section 1.1. A different set of trust functions would admit a different set: provenance for cost accounting or for licensing compliance would need relations we do not have. The vocabulary therefore stays open: a tenth relation is warranted when it answers a trust-function query that no combination of the nine answers, *and* its decision test is not a refinement of one already listed in Table 12. Two candidates we regard as open are an explicit authorization relation linking an action to the

policy that permitted it, and a responsibility relation for multi-agent delegation; both are currently expressed indirectly, and Section 7 treats them as such.

C The Evidence-Tracing Projection

Section 1 describes evidence tracing as the projection of execution provenance onto its evidence-support relations, together with the content-derivation links that connect them. This section states precisely what that means. One consequence is worth flagging at the outset: $\pi(G)$ is not a subgraph of G. It retains a subset of G's edges, but it also *adds* a layer of explicitly labeled SUPPORT-CANDIDATE edges recording content flow that no semantic edge has yet adjudicated. Evidence tracing is a sub-problem of execution provenance in the sense that it answers a strictly weaker question, not in the sense that its graph is contained in the larger one.

C.1 Definition

An **execution provenance graph** of a single agent run is a typed attributed directed multigraph

$$G \;=\; (V,\; E,\; \tau,\; \lambda,\; \alpha),$$

where V is the set of nodes; $\tau : V \to \mathcal{T}$ assigns each node a type from the vocabulary of Table 10, inducing the partition $V = \mathcal{E} \uplus \mathcal{X} \uplus \mathcal{A}$ into evidence units, execution units, and actors; E is an abstract set of edges with endpoint maps $\mathrm{src}, \mathrm{tgt} : E \to V$, so that one ordered pair of nodes may carry several edges (a passage can both DERIVE and SUPPORT the same claim, a tool call can both USE and DEPEND-ON the same parameter, and, crucially, two edges may share a label and differ only in *origin*, which is how a citation the agent DECLARED and an annotator subsequently INFERRED is recorded as two SUPPORT edges rather than one); $\lambda : E \to \mathcal{R}^+$ labels each edge, where $\mathcal{R}^+ = \mathcal{R} \cup \{\text{SUPPORT-CANDIDATE}\}$ extends the nine relations with the derived label introduced below, subject to the signature constraints of Table 11. An execution provenance graph produced by instrumentation or annotation carries no SUPPORT-CANDIDATE edge; the label exists so that π's output lies in the same class as its input, and it has signature $\mathcal{E} \to \mathcal{C}$, is asymmetric, and carries a temporal requirement inherited from the DERIVE, USE and GENERATE edges of its witness path; and α assigns attributes to nodes and edges, including timestamps, actor identity, content or content hash, trust and taint labels, and the *origin* and *assurance* values of Section B.4.

Partition the relation vocabulary into

$$\mathcal{R}_{\mathrm{sem}} = \{\text{SUPPORT}, \text{CONTRADICT}, \text{INVALIDATE}\}, \qquad \mathcal{R}_{\mathrm{der}} = \{\text{DERIVE}\},$$
$$\mathcal{R}_{\mathrm{proc}} = \{\text{USE}, \text{GENERATE}, \text{DEPEND-ON}, \text{TRIGGER}, \text{UPDATE}\}.$$

Here $\mathcal{R}_{\mathrm{proc}}$ groups the *causal* and *procedural* families of Appendix B.3 together, because the projection treats them alike; the two remain distinct relation families everywhere else in the paper.

The **evidence-tracing projection** is the map π sending G to the evidence view $\pi(G) = (V', E', \tau', \lambda', \alpha')$. All five clauses below are evaluated against the *input* graph G; in particular clause 3 reads procedural edges of G before clause 2 removes them, so clause 3 does not depend on the output of clause 2; clauses 4 and 5 are then computed from the edge sets that clauses 1 and 3 produce.

1. **Edges retained verbatim.** $E_{\mathrm{keep}} = \{\, e \in E \;:\; \lambda(e) \in \mathcal{R}_{\mathrm{sem}} \cup \mathcal{R}_{\mathrm{der}} \cup \{\text{SUPPORT-CANDIDATE}\} \,\}$. The last disjunct is vacuous on an instrumented graph; together with clause 3's crossing requirement it is what makes π idempotent.

2. **Edges discarded.** No edge with $\lambda(e) \in \mathcal{R}_{\mathrm{proc}}$ appears in E'.

3. **Contracted candidate paths.** Call a path of G a *content-flow path* when every one of its edges is a DERIVE edge traversed forwards, a GENERATE edge traversed forwards from an activity to the artifact it produced, or a USE edge traversed backwards from a consumed artifact to the activity that consumed it, *and at least one of its edges is a USE or GENERATE edge*, so that the path passes

through the execution layer. For $d \in \mathcal{E}$ and $c \in \mathcal{C}$, set E_{cand}, a set of fresh edge identities disjoint from E, to contain a derived edge SUPPORT-CANDIDATE(d, c) whenever G contains a content-flow path from d to c *and* no SUPPORT or INVALIDATE edge already runs from d to c (a CONTRADICT edge between them does not block, since contradicting is not adjudicating support). A candidate exists to flag content flow that has not yet received a support judgment, so a pair that already carries one, whether inferred or declared, needs none; the edge's origin records which. Such an edge records that d *could have* contributed content to c; it is an availability statement, and it is deliberately *not* a SUPPORT edge. Requiring a content-flow path to cross the execution layer is the second half of what makes π idempotent: a chain of DERIVE edges alone is already in the evidence view and must generate no candidate.

4. **Nodes retained.** $V' = \mathcal{E} \cap \big(\text{endpoints of } E_{\text{keep}} \cup E_{\text{cand}}\big)$, together with any execution unit that is the target of a retained SUPPORT or INVALIDATE edge, since justifying an action, and recording that its precondition no longer holds, are both evidence-tracing questions. No retained edge is therefore left dangling.

5. **Attributes retained.** Content, content hash, timestamps, trust labels, origin, assurance, and, on SUPPORT-CANDIDATE edges, a witness identifier are kept. Scoping the witness by *label* rather than by which clause admitted the edge is what keeps π idempotent on attributes as well as on structure: on a second application such an edge arrives through clause 1, not clause 3, and must not lose its witness. Execution-specific attributes are dropped: actor identity, latency, permission scopes, retry counts, and tool manifests. The witness identifier names one content-flow path of G that licenses the edge; when several paths license the same edge the witness names one of them, so recovery of G from $\pi(G)$ is partial by design. A contracted edge takes origin = DERIVED, the fourth origin value of Section B.4, since it is computed from G rather than logged, judged, or asserted.

The components of the output are then $E' = E_{\text{keep}} \cup E_{\text{cand}}$, with τ' and λ' the restrictions of τ and λ to V' and E', and α' the restriction of α with the execution-specific attributes of clause 5 removed, extended on E_{cand} by $\lambda'(e) =$ SUPPORT-CANDIDATE and the attributes just listed.

C.2 Properties

P1 (Information loss; π is not injective). Many distinct execution graphs have the same evidence view. Two runs that reach the same claims from the same evidence through different tool calls, different orderings, or different agents are indistinguishable under π. Consequently **evidence tracing cannot recover execution provenance**: the support view is strictly weaker than the process view, which is why a survey of one is incomplete without the other.

P2 (Groundedness). Every edge of $\pi(G)$ is licensed by G: each retained edge *is* an edge of G, and each contracted edge is witnessed by at least one content-flow path of G. In particular π asserts no SUPPORT edge that is not already asserted in G; the edges it adds carry the distinct label SUPPORT-CANDIDATE precisely so that availability is never read as support.

P3 (Not all influence is evidence). No edge labeled in $\mathcal{R}_{\text{proc}}$ survives π. The consequential case is the two *influence* relations, DEPEND-ON and TRIGGER: this is the formal content of the distinction between *supporting* a conclusion and merely *influencing* it, since influence without semantic support is present in G and absent from $\pi(G)$ by construction. USE, GENERATE, and UPDATE are also discarded, but they are bookkeeping rather than influence claims, and USE and GENERATE survive indirectly as witnesses of contracted edges. The exclusion is deliberate: the evidence view addresses evidential justification, while the process view records execution history and dependencies, including causal influence where established. A system that reports only $\pi(G)$ cannot answer safety questions that require procedural or causal influence information, and a system that reports only $\mathcal{R}_{\text{proc}}$ cannot answer grounding questions.

P4 (Idempotence). $\pi(\pi(G)) = \pi(G)$. Clause 1 retains $\mathcal{R}_{\text{sem}}$, $\mathcal{R}_{\text{der}}$ and SUPPORT-CANDIDATE edges, which is every label present in an evidence view, so nothing is lost; clause 2 removes nothing, since no $\mathcal{R}_{\text{proc}}$

Table 15: The projection applied to the running example of Figure 3. Nodes: q user query, r retrieval call, p retrieved passage, m memory item, z_1 claim-forming reasoning step, c intermediate claim, g generated tool parameter, k tool call, o tool output, z_2 answer-forming reasoning step, a final answer, n new external evidence, w memory write, a' revised answer.

Edge in G	**In $\pi(G)$?**	**Why**
$\textsc{Trigger}(q, r)$	Dropped	Procedural: governs whether retrieval happened.
$\textsc{Generate}(r, p)$	Dropped	Activity-to-artifact bookkeeping. It carries no edge into $\pi(G)$, but may appear inside the witness path of a contracted edge.
$\textsc{Use}(z_1, p)$, $\textsc{Use}(z_1, m)$	Dropped	Availability, not support.
$\textsc{Generate}(z_1, c)$	Dropped	Activity-to-artifact.
$\textsc{Derive}(p, c)$	**Kept**	Content of the claim is a transformation of the passage.
$\textsc{Support}(p, c)$	**Kept**	Passage content warrants the claim.
$\textsc{Support}(m, c)$	**Kept**	Memory content warrants the claim; later invalidated, see below.
$\textsc{Derive}(c, g)$	**Kept**	Tool argument derived from the claim's content.
$\textsc{Trigger}(c, k)$	Dropped	Governs whether the tool was called.
$\textsc{Use}(k, g)$	Dropped	The call consumed the parameter; recorded, not judged.
$\textsc{Depend-on}(k, g)$	Dropped	Governs how the call was parameterized. This edge is essential for the tool-use safety question and invisible in the evidence view.
$\textsc{Generate}(k, o)$	Dropped	Activity-to-artifact.
$\textsc{Use}(z_2, o)$, $\textsc{Generate}(z_2, a)$	Dropped	Answer construction: procedural bookkeeping.
$\textsc{Support}(o, a)$	**Kept**	Tool output warrants the final answer.
$\textsc{Contradict}(n, m)$	**Kept**	New evidence conflicts with the memory item.
$\textsc{Invalidate}(n, c)$	**Kept**	New evidence removes the claim's warrant.
$\textsc{Trigger}(n, w)$, $\textsc{Update}(a', a)$	Dropped	Procedural recovery and state succession.
$\textsc{Support-candidate}(p, a)$	**Added**	Witness: $p \xrightarrow{\textsc{Derive}} c \xrightarrow{\textsc{Derive}} g \xleftarrow{\textsc{Use}} k \xrightarrow{\textsc{Generate}} o \xleftarrow{\textsc{Use}} z_2 \xrightarrow{\textsc{Generate}} a$. The passage is a possible content contributor to the answer; whether it *supports* the answer is a separate judgment.
$\textsc{Support-candidate}(m, a)$	**Added**	Same shape, witnessed through $m \to z_1 \to c \to \cdots \to a$. Read together with $\textsc{Invalidate}(n, c)$, this is the edge that tells an auditor a discredited memory item may have reached the answer.
$\textsc{Support-candidate}(c, a)$, $\textsc{Support-candidate}(g, a)$	**Added**	The claim and the tool parameter derived from it are also content contributors to the answer.
$\textsc{Support-candidate}(p, c)$, $\textsc{Support-candidate}(m, c)$, $\textsc{Support-candidate}(o, a)$	Not added	Blocked by clause 3: $\textsc{Support}(p, c)$, $\textsc{Support}(m, c)$ and $\textsc{Support}(o, a)$ already adjudicate these pairs.

edge survives the first application; and clause 3 adds nothing, since a content-flow path requires a Use or Generate edge and the evidence view contains none. This last point is why clause 3 requires the path to cross the execution layer: without that requirement a bare chain of Derive edges would generate a fresh candidate on every application and π would not converge.

We use *projection* in this structural sense: π is a total, idempotent, information-losing map from typed provenance graphs to typed evidence graphs that preserves the semantic sub-structure. We do not claim it is a projection in a categorical sense, and no result in this survey depends on stronger properties.

C.3 Worked Example

Table 15 applies the definition to the running example of Figure 3. We list the twenty edges on which the projection acts. The generator of the recovery-phase artifact a', and the memory item that the write w produces, are elided; note that this is a simplification of the example rather than of the definition, since supplying a' with a generator and an input would add further candidate edges into a'. Of those twenty the evidence view retains seven, discards thirteen, and adds four contracted candidates.

The example makes the division of labour concrete. The evidence view alone shows an auditor that the final answer may have drawn on a claim supported partly by a memory item that was later contradicted and whose

warrant was invalidated, an attribution failure. It does not show that the call was triggered by that claim, or that its parameterization depended on the argument derived from it, which is the fact a safety analyst needs. Neither view subsumes the other, and this is exactly why the survey treats execution provenance as the full graph and evidence tracing as its projection.

D Benchmark Catalog and Coverage Details

This appendix derives every cell of the coverage heatmap from the benchmark catalog of Table 8, and states the annotation protocol under which that catalog was coded. The main text uses Figure 5 to emphasize coverage gaps; Table 16 shows the computation itself, so that any rating can be reproduced and checked against the catalog. The per-benchmark entries are not repeated here: the catalog is Table 8 in Section 6, and this appendix adds only what the main text does not carry: the derivation and the protocol.

D.1 Derivation of the Coverage Heatmap

Figure 5 is derived from Table 8 rather than assigned by judgment. For the five capabilities that appear as columns in the catalog (evidence labels, tool calls, memory, multi-agent structure, and safety attacks), a family's rating is the maximum over its member benchmarks under the mapping Yes $\rightarrow$ **S**, Partial / Task-dependent / State-based $\rightarrow$ **P**, No $\rightarrow$ **L**.

The maximum is the right aggregator because each cell states an *existential* claim (does *any* benchmark in this family evaluate this capability?), and it has two consequences a reader should keep in view. First, the heatmap is an optimistic upper bound: a family rated **S** typically contains members rated **L** for the same capability. Second, families differ in size, from two members to nine, so a maximum is easier to attain in a large family; each cell of Table 16 therefore also reports how many members attain the rating. The **L** cells are unaffected by family size, and they are the ones the survey's argument rests on: an **L** means *no* member covers the capability, which cannot be an artifact of family size.

The catalog's remaining column, trace availability, has no heatmap counterpart. It is a precondition for evaluating any of the seven capabilities rather than a capability in its own right, so we report it per benchmark in Table 8 and do not aggregate it.

Two capabilities have no catalog column and are coded directly from the released artifacts by a countable criterion. *Provenance relations* is rated by how many of the nine relations of Section B.2 a benchmark releases as ground-truth labels: **S** for three or more, **P** for one or two, **L** for none. Citation benchmarks label SUPPORT and nothing else; taint benchmarks label source-to-sink influence, which is evidence for DEPEND-ON and nothing else; trace-debugging benchmarks label the responsible step, which is evidence for TRIGGER and nothing else. Relations recoverable from a call log but never labeled (USE and GENERATE) do not count, because an unlabeled relation supplies no ground truth. *Recovery* is rated **S** only when a benchmark scores repair outcomes; it is **P** when the benchmark scores something adjacent to repair, namely superseding stale memory, localizing the failure that a repair would target, or blocking an unsafe action before it executes, and **L** otherwise. Under these criteria no family reaches **S** in either column, which are therefore the sharpest gaps the heatmap records.

Table 16 gives the resulting derivation. The determining member is the benchmark that attains the family maximum; where several members tie we name them, up to four, and otherwise indicate the count.

Three observations follow from the derivation that are not visible in the heatmap itself. First, *five of the six families reach* **S** *on exactly one capability* and are **P** or **L** everywhere else. For four of the five that capability is the one the family was built for; the exception is the trace-debugging family, whose only **S** is in the multi-agent column and is inherited from MAST, which belongs to both families. Second, the remaining family, provenance/security, reaches **S** on three capabilities and is the only family with no **L** cell; but it is also the largest at nine members, two of its three **S** ratings are unanimous while the third rests on a single member, and it never reaches **S** on evidence labels or recovery. Third, and most consequentially, *no family reaches* **S** *on provenance relations or on recovery.* Every rating above **L** in those two columns is carried by a benchmark labeling exactly one relation (SUPPORT in citation benchmarks, DEPEND-ON in taint benchmarks,

Table 16: Cell-by-cell derivation of Figure 5 from Table 8. For the five catalog columns, each entry gives the rating, the number of family members attaining it out of the family size, and the determining member; the provenance-relations and recovery columns, coded from released relation labels and scored repairs (Section D.1), give the rating and its basis. Entries marked "—" have no member above **L**.

Family	Evidence labels	Tool calls	Memory	Multi-agent	Safety attacks	Prov. relations	Recovery
RAG *(7)*	**S** 4/7 (FActScore, ALCE, SourceCheckup, RAGTruth)	**L** 0/7 (—)	**L** 0/7 (—)	**L** 0/7 (—)	**L** 0/7 (—)	**P** (SUPPORT only)	**L** (—)
Tool-use *(4)*	**L** 0/4 (—)	**S** 3/4 (WebArena, ToolBench, τ-bench)	**P** 2/4 (AgentBench task-dep.; τ-bench state-based)	**L** 0/4 (—)	**P** 1/4 (τ-bench policy checks)	**L** (none labeled)	**L** (—)
Memory *(2)*	**P** 2/2 (LoCoMo, MemoryArena)	**P** 1/2 (MemoryArena task-dep.)	**S** 2/2 (LoCoMo, MemoryArena)	**L** 0/2 (—)	**L** 0/2 (—)	**P** (session-level SUPPORT references)	**P** (superseding stale items; repair unscored)
Multi-agent *(2)*	**P** 1/2 (AgenTracer/Aegis)	**P** 2/2 (MAST, AgenTracer task-dep.)	**P** 2/2 (MAST, AgenTracer task-dep.)	**S** 2/2 (MAST, AgenTracer/Aegis)	**L** 0/2 (—)	**P** (responsible step, evidencing TRIGGER)	**L** (—)
Trace-debugging *(2)*	**P** 1/2 (TRAIL)	**P** 2/2 (TRAIL, MAST)	**P** 2/2 (TRAIL, MAST)	**S** 1/2 (MAST)	**L** 0/2 (—)	**P** (error location on trace steps)	**P** (localization only; repair unscored)
Provenance / security *(9)*	**P** 3/9 (OpenAgentSafety, AgentSentry, NeuroTaint)	**S** 9/9 (all members)	**S** 1/9 (NeuroTaint/TaintBench)	**P** 4/9 (OpenAgentSafety, MCP-SafetyBench, NeuroTaint, AgentSpec)	**S** 9/9 (all members)	**P** (source-to-sink influence, evidencing DEPEND-ON)	**P** (blocking, not repair)

TRIGGER in trace-debugging benchmarks), or by an artifact adjacent to repair rather than repair itself. No benchmark in the catalog releases the typed relation vocabulary as ground truth, and none scores recovery. This is the concrete form of the gap that Section 7 argues future benchmarks should close.

D.2 Annotation Protocol for the Coverage Ratings

Three rating schemes appear in this survey: the **S**/**P**/**L** coverage ratings of Figure 5, the memory-capability symbols of Table 6, and the benchmark-capability symbols of Table 8. They share one annotation protocol, stated here so that any cell can be re-derived and checked against the artifact. The heatmap is not coded directly: it is computed from the catalog by the rule of Section D.1, so the protocol below governs only the per-artifact tables from which it is computed, and correcting a catalog cell propagates to the heatmap automatically.

Unit of coding and evidence admitted. The unit is a single (artifact, capability) pair: one benchmark or system, one column. Evidence is admitted in a fixed order of precedence, and the first level that decides the cell ends the search: (i) the released dataset, repository, or evaluation harness; (ii) the artifact's own paper, including its appendices; (iii) its documentation. The ordering is what makes the ratings checkable, and it has one key consequence: *a capability claimed in prose but with no released label or trace never yields the top rating.* We code what an artifact makes available to a third party, not what it reports about itself. Where an artifact was not publicly released at the time of coding, it is coded from its paper alone and can attain at most the middle rating; this is the one case in which the middle rating rests on a description rather than on a release.

Decision procedure. Each cell is decided by the following four steps in order, so that the same evidence always yields the same rating:

1. *Ground truth?* Does the artifact release per-instance labels for the capability and report a metric computed over them? If yes, the cell takes the top rating (✔ / ✓ / **S**) and coding stops.
2. *Present but unlabeled?* Does the capability appear in the released traces or the system's behavior without ground-truth labels, or is it labeled for only a subset of instances, only at run level, or only for some memory types, or labeled for all instances with no metric computed over the labels? If yes, the cell takes the middle rating (• / ▲ / **P**).
3. *Structural qualifier?* Three values are not degrees of coverage but statements about *how* the capability is realized, and are applied before the negative rating: T when coverage varies across the artifact's

own task suite rather than across instances, and S when the capability exists as external environment state rather than as an agent-side artifact. ★ is the corresponding value for a work that studies a capability adversarially rather than providing it.

4. *Otherwise* the cell takes the negative rating (✗ / ❍ or ✗ / **L**). ❍ marks a system that exposes the information from which the capability could be built but does not implement it.

Where the middle and negative ratings divide. The step that decides most contested cells is the boundary between the middle rating (• / ▲ / **P**) and the negative one (✗ / ❍ or ✗ / **L**), and it needs separate treatment because the two labels are easy to misread as degrees of the same thing. The middle rating is a claim that the capability is *present in what the artifact releases* but is not ground truth: the traces contain it unlabeled, or it is labeled for only some instances, or realized in a form the column does not name. The negative rating is a claim that it is *not there to be found*: for a benchmark, neither labels nor recoverable traces are released; for a system, the capability is not implemented. The operational test is therefore not "how much coverage is there" but "could a third party extract this from the release at all". One consequence is that **L** is the falsifiable direction: a single released file showing the capability overturns an **L**, whereas overturning a **P** requires showing that a metric is computed over labels, which is the step 1 test. This is why Section D.1 treats the **L** cells as the signal unaffected by family size on which the survey's argument rests, and why ❍ in Table 6 is worded as "exposes the information but does not implement it" rather than as weak support.

Two worked cells. Applying the procedure to two cells that could plausibly have been coded either way shows how the steps decide them.

ALCE, evidence labels → ✓. Step 1 asks for per-instance ground truth *and* a metric over it. ALCE releases per-claim citation annotations and reports citation precision and recall computed from them (Gao et al., 2023), so both halves hold and coding stops at step 1. Had it released the annotations without scoring them, step 1 would fail and step 2 would give •, since the distinction is the metric, not the labels.

τ-bench, memory → S. The benchmark carries information across turns, so a coder might record ✓ or •. Step 1 fails: nothing about memory is labeled per instance. Step 2 is the tempting one, but what persists is the environment database the tools read and write, not an agent-side memory item (Yao et al., 2025); the capability exists, yet not as the artifact the column is about. Step 3 therefore applies and the qualifier S records both facts at once: present, but realized as external state. It maps to **P** in the heatmap, so the family rating is unaffected; what the qualifier preserves is *why*, which a bare **P** would discard.

Scope and known limits of the protocol. The protocol codes the *presence* of a capability, never its quality: a benchmark with weak evidence labels and one with strong labels are both ✓, and the survey makes no claim about which is better. Three limits follow. First, coding was performed by the authors jointly, with disagreements resolved by discussion against the released artifact; there was no independent second coder and we therefore report no inter-coder agreement, which is the same limitation recorded for the corpus in Section 1.1. Second, the ratings are a snapshot: they reflect the artifact versions available at the time of writing, and a subsequent release can change a cell. Third, step 1 turns on whether a metric is *computed over* the labels, which is the step at which we expect residual disagreement, and it is the step to check first when a cell is in doubt.

The protocol's main benefit is that every rating becomes a claim about a specific released artifact, which a single counterexample, such as a labeled file or a metric in a results table, can overturn.

E Additional Details for Open Problems

This appendix expands the open problems discussed in Section 7. It does not restate the main text: the main text makes the argument for each open problem and cites the work that motivates it, while this appendix carries in table form only the artifact the argument implies: a requirements checklist an implementer or benchmark designer can work from. Where a requirement here corresponds to a sentence of Section 7, the table row is the operational version of it, and a reader following the argument alone can skip this appendix

Table 17: Design requirements for unified LLM-agent provenance schemas.

Requirement	What should be represented	Why it matters
Trace sources	User instructions, model outputs, retrieval calls, tool calls, memory operations, environment observations, inter-agent messages, and final responses.	Enables end-to-end reconstruction of agent execution.
Evidence units	User queries and instructions, documents, passages, observations, tool outputs, memory items, generated tool parameters, policies, tool manifests, inter-agent messages, intermediate claims, and final answers.	Supports claim-level verification and attribution.
Execution units	Reasoning steps, retrieval calls, tool invocations, memory reads and writes, environment actions, and delegation steps.	Supports debugging, safety enforcement, and audit.
Typed relations	Support, Contradict, Depend-on, Derive, Trigger, Update, Invalidate, Use, and Generate.	Turns raw logs into provenance graphs that expose influence and responsibility.
Trust metadata	Source trust, permission scope, confidentiality/integrity labels, timestamps, actor identity, and policy constraints.	Enables runtime enforcement, privacy-aware audit, and governance.
Replay and recovery fields	Checkpoints, rollback targets, invalidation events, repair steps, human approvals, and state-change records.	Supports intervention and recovery, not only post-hoc explanation.

without loss. The main text presents the research agenda; this appendix provides more detailed design requirements, representative foundations, and evaluation dimensions.

E.1 Trace Schema Requirements

A unified LLM-agent provenance schema should combine operational execution records with semantic evidence relations. Table 17 summarizes the main schema requirements.

Existing standards and frameworks provide partial foundations. W3C PROV-DM defines general provenance concepts such as entities, activities, agents, generation, usage, derivation, and invalidation (World Wide Web Consortium, 2013). OpenTelemetry provides distributed tracing abstractions, while OpenLineage provides data workflow lineage (OpenTelemetry, 2026; OpenLineage Contributors, 2026). AgentOps and AgentTrace move closer to LLM-agent observability by recording structured agent artifacts and execution context (Dong et al., 2024; AlSayyad et al., 2026). PROV-AGENT adapts provenance modeling to agentic workflows (Souza et al., 2025). However, LLM-agent provenance requires additional semantic units and relations, especially generated claims, evidence support, contradiction, memory invalidation, tool-argument influence, and inter-agent responsibility.

E.2 Detailed Agenda for Semantic and Claim-Level Provenance

Claim-level provenance should distinguish at least five conditions that are often conflated in current systems: citation presence, source relevance, claim support, contradiction, and evidence omission. Only support and contradiction are taxonomy relations; citation presence and omission are the derived predicates of Appendix B.5, and source relevance is weaker than any of them. ALCE, FActScore, RAGTruth, RAGChecker, SourceCheckup, and related attribution surveys provide foundations for this direction (Gao et al., 2023; Min et al., 2023; Niu et al., 2024; Ru et al., 2024; Wu et al., 2025; Schreieder et al., 2026). However, agentic settings introduce additional challenges because evidence may be transformed by reasoning, retrieval, tools, memory, reflection, or inter-agent communication before it influences the final answer or action.

Future work should therefore evaluate not only whether a final claim is supported, but also how the supporting information traveled through the execution. Important cases include: a claim derived from multiple retrieved passages; a tool output that contradicts a memory item; a memory item summarized from a previous conversation; a policy constraint that justifies or blocks an action; and an inter-agent message that propagates

Table 18: Checklist for memory and multi-agent provenance.

Setting	Provenance fields	Open question
Memory write	Source, timestamp, writer, evidence basis, confidence, and policy context.	Was the memory created from trusted evidence or from unsupported/injected content?
Memory update	Previous version, update source, update reason, conflict status, and invalidation marker.	Does the update revise, contradict, or invalidate earlier memory?
Memory retrieval	Retrieval query, retrieved memory items, ranking score, context of reuse, and downstream consumer.	Why was this memory reused, and how did it affect the current execution?
Memory deletion or invalidation	Deletion trigger, invalidating evidence, affected downstream claims/actions, and recovery steps.	Can stale or poisoned memory be removed without breaking auditability?
Inter-agent communication	Sender, receiver, role, message content, cited evidence, delegated task, and verification status.	Which agent introduced, verified, reused, or amplified a claim?
Multi-agent failure	Responsible agent, responsible step, error mode, propagated message, and counterfactual trajectory.	Where did the failure originate, and how did it propagate across agents?

Table 19: Runtime safety and recovery requirements for provenance-aware agents.

Capability	Provenance requirement	Representative foundations
Prompt-injection and tool-risk detection	Identify whether untrusted external content influences privileged instructions, tool calls, or memory updates.	InjecAgent, AgentDojo, ToolEmu (Zhan et al., 2024; Debenedetti et al., 2024; Ruan et al., 2024)
Information-flow control	Track confidentiality and integrity labels across inputs, tools, memory, and outputs.	CaMeL, Fides (Debenedetti et al., 2026; Costa et al., 2025)
Semantic taint tracking	Trace influence through summarization, paraphrasing, reasoning, and memory consolidation.	NeuroTaint (Cai et al., 2026)
Argument verification	Check whether sensitive tool arguments derive from trusted sources or explicit user intent.	Agent-Sentry (Sequeira et al., 2026)
Execution bounding	Enforce tool permissions, access boundaries, and policy constraints at runtime.	AgentSpec, AgentBound (Wang et al., 2026; Bühler et al., 2026)
Recovery	Invalidate stale memory, quarantine contaminated evidence, retry tool calls, request approval, roll back state changes, or replay from safe checkpoints.	Open problem building on provenance enforcement and trace debugging.

an unsupported assumption. Semantic taint and causal influence methods such as NeuroTaint suggest one path toward tracing transformed influence, but robust semantic provenance remains open (Cai et al., 2026).

E.3 Memory and Multi-Agent Provenance Checklist

Table 18 lists provenance fields that future memory and multi-agent systems should expose.

Memory systems should not treat long-term memory as a generic context store. Memory items should carry lineage, validity, trust, and downstream-use metadata, especially when they affect tool calls, personalization, or long-horizon decisions. Memory poisoning and memory-injection attacks highlight the need to trace how compromised memory influences later executions (Chen et al., 2024; Dong et al., 2025). Multi-agent systems require a parallel form of message-level provenance. In workflows built from delegation, critique, verification, and role specialization, provenance should identify not only which final answer was wrong, but which agent introduced the unsupported claim, which agent failed to verify it, and which message propagated it (Wu et al., 2024; Li et al., 2023; Cemri et al., 2025; Zhang et al., 2026; Kong et al., 2026).

E.4 Runtime Safety and Recovery Requirements

Runtime provenance should support both enforcement and repair. Table 19 summarizes the distinction.

Most current systems focus on detection, blocking, or policy enforcement. A stronger provenance-aware runtime should support intervention. For example, if a tool argument is derived from untrusted web content, the system should not only block the action but also identify the contaminated source, remove the derived parameter, request confirmation, and prevent the same contamination from being written into memory. If a

Table 20: Requirements for realistic provenance benchmarks and governance.

Requirement	Current limitation	Future benchmark target
Full execution traces	Existing benchmarks often capture only answers, tool calls, attacks, memory retrieval, or failure labels.	Capture retrieval, tools, memory, environment states, inter-agent messages, claims, actions, and state changes.
Relation annotations	Provenance relations are rarely labeled directly.	Annotate the nine typed relations, and separately flag unsafe influence and recovery intervention, which are properties of an execution rather than relations.
Cross-component coverage	RAG, tool-use, memory, and multi-agent benchmarks usually evaluate isolated capabilities.	Evaluate how evidence, tools, memory, messages, and state changes jointly affect final outputs and actions.
Recovery evaluation	Benchmarks usually test success, failure, hallucination, or unsafe action.	Test whether agents can repair traces by invalidating memory, quarantining evidence, retrying tools, requesting approval, or rolling back unsafe actions.
Privacy-aware tracing	Traces may expose confidential documents, credentials, API arguments, memory items, and internal reasoning artifacts.	Require minimization, anonymization, access control, encryption, retention policies, and selective disclosure.
Audit and governance	Trace collection alone does not guarantee accountability.	Evaluate whether traces are inspectable, policy-compliant, comparable, and usable for audit.

memory item is later contradicted by new evidence, the system should invalidate the memory and identify downstream claims or actions that depended on it. These recovery operations require provenance graphs that preserve dependencies across evidence, tools, memory, state changes, and actions.

E.5 Benchmark and Governance Requirements

Future benchmarks should evaluate provenance as a first-class object rather than as an implicit by-product of task success. Table 20 summarizes the main benchmark and governance requirements.

Existing benchmark families provide partial foundations. RAG and attribution benchmarks support evidence grounding; tool-use and safety benchmarks expose tool calls, unsafe actions, and prompt-injection risks; memory benchmarks evaluate recall and personalization; multi-agent benchmarks evaluate coordination and failure modes; and trace-debugging benchmarks support failure localization (Gao et al., 2023; Ru et al., 2024; Wu et al., 2025; Liu et al., 2024; Zhou et al., 2024; Ruan et al., 2024; Debenedetti et al., 2024; Yao et al., 2025; Vijayvargiya et al., 2026; Zong et al., 2026; Maharana et al., 2024; He et al., 2026; Deshpande et al., 2025; Cemri et al., 2025; Zhang et al., 2026; Kong et al., 2026). The remaining gap is end-to-end evaluation of provenance chains that connect evidence and tool outputs to memory updates, intermediate claims, final answers, external state changes, and recovery actions.

Privacy and governance should be evaluated together with provenance quality. Provenance traces can contain sensitive user information, confidential documents, credentials, API arguments, internal reasoning artifacts, and long-term memory items. Observability systems such as AgentOps and AgentTrace provide foundations for trace collection (Dong et al., 2024; AlSayyad et al., 2026), while Fides highlights confidentiality and integrity constraints (Costa et al., 2025). Future systems should additionally support trace minimization, access control, anonymization, retention policies, encryption, and selective disclosure so that provenance infrastructure does not itself become a privacy or compliance risk.