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

Anonymous authors
Paper under double-blind review

Abstract

Large language model (LLM)-based agents are rapidly evolving from passive text generators into autonomous systems capable of planning, tool use, retrieval, memory access, environmental interaction, and multi-agent collaboration. While these capabilities expand agent autonomy, they also make agent behavior increasingly difficult to verify, debug, and audit. The difficulty is that agents are evaluated almost entirely by their final answers, even though a correct answer reveals nothing about how an output was produced, what evidence supported each claim, whether tool calls were justified, how memory shaped later decisions, or where execution failures originated. To answer such questions, we look beyond outputs to the execution process itself, and structure it through evidence tracing and execution provenance. Provenance offers a process-level accountability layer for trustworthy LLM agents by modeling the connections among retrieved evidence, tool outputs, memory items, observations, intermediate claims, actions, and final answers. In this survey, we treat execution provenance as the full typed graph of an agent execution and evidence tracing as its projection onto evidence-support relations, giving a single framework that spans retrieval grounding through audit and recovery. We introduce a taxonomy that characterizes agent systems along six dimensions: trace sources, evidence and execution units, provenance relations, tracing granularity and timing, representation forms, and trust functions. We then review existing methods, categorized into seven main research threads: provenance representation, evidence attribution, tool-use provenance, runtime guardrails, provenance-bearing memory, observability, and failure diagnosis. Finally, we connect existing benchmarks, datasets, and metrics to provenance-related capabilities, discuss how evaluation can move beyond final-answer correctness toward process-level accountability, and outline open challenges for unified trace schemas, semantic provenance, provenance-aware safety, realistic execution-trace benchmarks, recovery-oriented evaluation, and privacy-aware audit infrastructure.

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

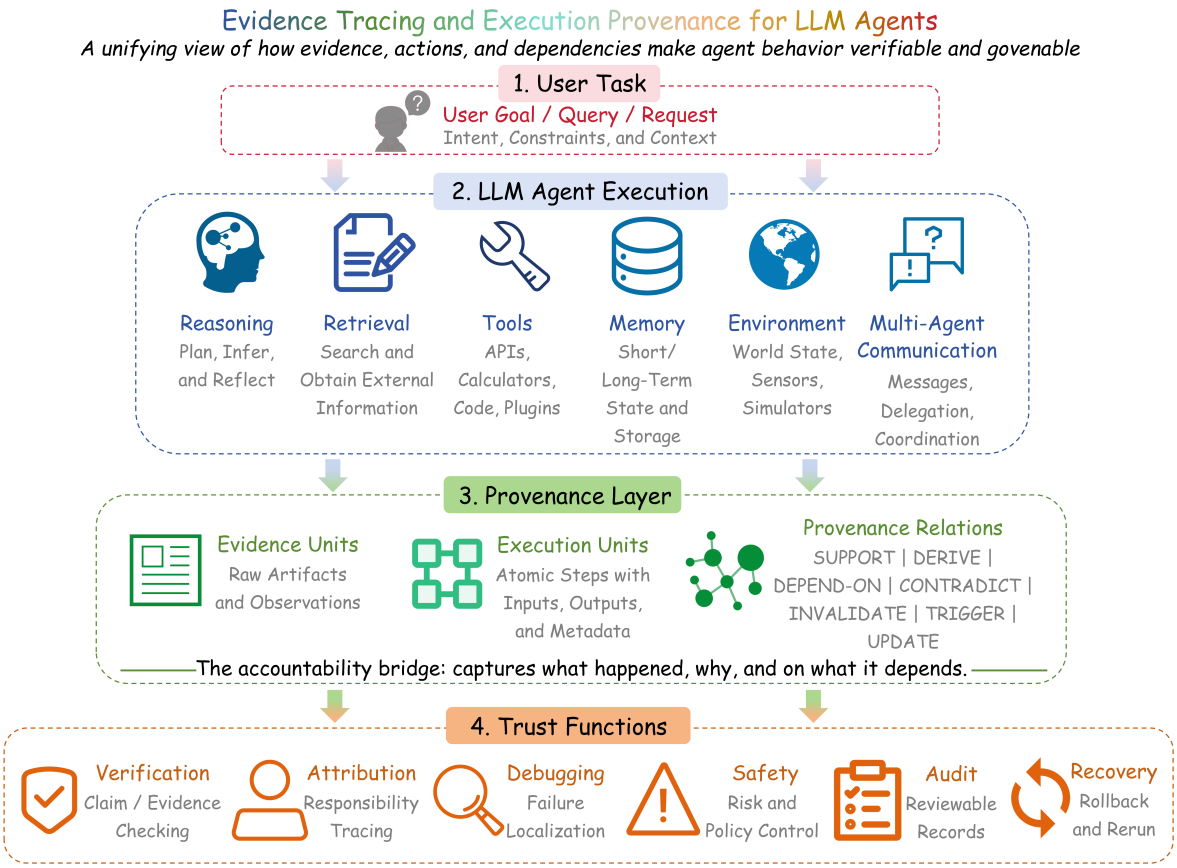

Figure 1: Overview of the proposed provenance framework. A user task initiates a heterogeneous agent execution involving reasoning, retrieval, tool use, memory, environment interaction, and multi-agent communication. The provenance layer records evidence units, execution units, and typed relations produced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

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. (2025), 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. (2026); Zhang et al. (2025a); Kong et al. (2025). These findings suggest that trustworthy agent systems require mechanisms that can record, connect, and reason over the evidence and execution steps underlying agent behavior.

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

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 .

[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).

Execution provenance. A structured and typed representation of an agent run, including evidence units such as retrieved documents, observations, tool outputs, memory items, intermediate claims, and final outputs; execution units such as retrieval calls, tool invocations, parameters, memory accesses, actions, and inter-agent messages; and the causal, procedural, dependency, update, contradiction, and invalidation relations among them.

Evidence tracing. The projection of execution provenance onto evidence-support and influence relations between evidence units and agent claims, decisions, or actions. It is therefore a sub-problem of execution provenance: every evidence-support link belongs to the broader provenance structure, while execution provenance additionally records how tools, memory, observations, actions, and inter-agent messages shape the end-to-end execution.

Together, the two terms define the organizing contrast of this survey: execution provenance is the *process view* of an agent run, while evidence tracing is the *support view* that explains which evidence backs which claims, decisions, and actions.

The need for provenance appears across several parts of current LLM-agent research. Evidence grounding asks whether generated claims are supported by retrieved sources; tool-use safety asks whether actions and arguments are justified; memory research asks how stored information is created, updated, and reused; graph-based methods provide ways to represent dependencies; and observability systems record executions for inspection and debugging. Viewed separately, these topics address different trust problems. Viewed together, they point to a common process-level question: how do evidence, tools, memory, observations, claims, actions, and agents jointly shape an execution outcome? This survey uses evidence tracing and execution provenance to organize these questions under a single framework, summarized in Figure 1 and developed into a six-dimensional taxonomy in Section 2. The taxonomy then guides the review along three provenance-critical threads: representation and evidence attribution, tool-using execution, and provenance-bearing memory, before Section 6 connects benchmarks, datasets, and metrics to process-level evaluation.

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.

Literature selection. We selected papers from LLM agents, RAG attribution, tool-use safety, memory-augmented agents, observability, and trace-based debugging when they explicitly record, represent, constrain, evaluate, or diagnose intermediate evidence or execution artifacts. We prioritize work that introduces trace structures, provenance relations, tool or memory lineage, runtime enforcement, failure localization, or provenance-relevant benchmarks. Rather than exhaustively surveying all RAG, memory, agent, or safety literature, we focus on work that directly contributes to evidence tracing or execution provenance.

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, distinguishing the full execution-provenance graph from its evidence-support projection.
- 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 review existing methods across seven research threads: provenance representation, evidence attribution, tool-use provenance, runtime guardrails, provenance-bearing memory, observability, and failure diagnosis.

- 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 open challenges for provenance-aware LLM agents, including unified trace schemas, semantic provenance, provenance-aware safety, realistic execution-trace benchmarks, recovery-oriented evaluation, and privacy-aware audit infrastructure.

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.

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 visualizes the connected objects and functions in this taxonomy, while representation forms are treated as the encoding layer for these provenance structures. Table 1 gives the full summary used as the backbone for the rest of the paper.

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 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 that generate provenance-relevant artifacts. In LLM agents, these sources extend beyond model outputs to include reasoning, retrieval, tool use, memory, environment interaction, and multi-agent communication. Reasoning traces can explain plans or action choices; retrieval traces provide evidence candidates; tool traces record calls, arguments, outputs, permissions, and errors; memory traces expose cross-session influence; environment traces record observations and state-changing actions; and multi-agent traces capture delegation, critique, and responsibility propagation Yao et al. (2023); Shinn et al. (2023); Qin et al. (2024); Zhou et al. (2024); Yao et al. (2025); Wu et al. (2024a); Li et al. (2023). Later sections examine tool and memory sources in depth, so here we use them only to define the taxonomy boundary.

Evidence and execution units. Trace sources produce two broad kinds of units. *Evidence units* are semantic objects that can support, contradict, invalidate, or contextualize claims and actions, such as documents, passages, observations, tool outputs, memory items, policies, and intermediate claims. *Execution units* are procedural objects that describe what the agent did, including reasoning steps, retrieval calls, tool

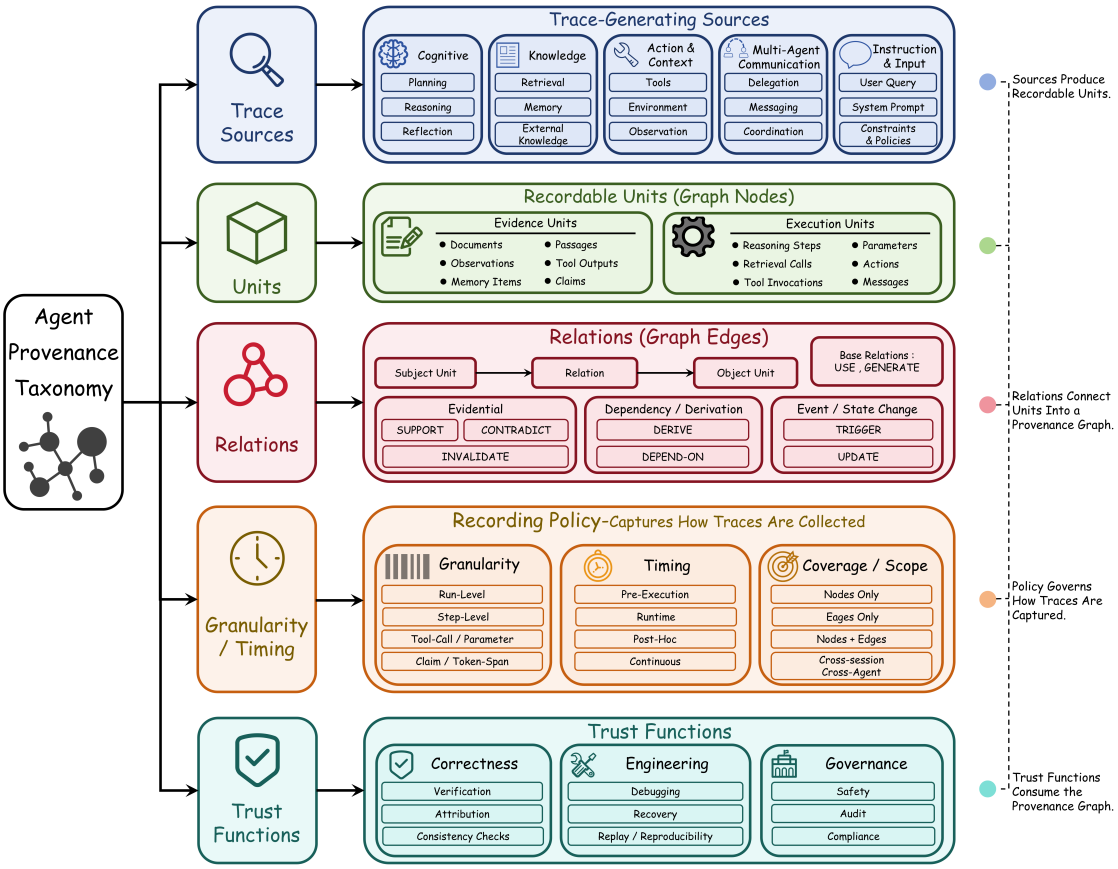

Figure 2: Taxonomy overview of agent provenance. The figure visualizes trace-generating sources, recordable evidence and execution units, typed provenance relations, recording policy, and downstream trust functions; Table 1 gives the full six-dimensional taxonomy, including representation forms.

invocations, generated parameters, memory reads or writes, environment actions, inter-agent messages, and final outputs. The distinction is useful but not absolute: a tool output is both the result of an execution step and possible evidence for a later claim. Provenance-aware agents therefore 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 such as Use, Generate, and Derive, and add agent-specific relations needed for LLM execution: Support, Depend-on, Contradict, Invalidate, Trigger, and Update. These relations separate semantic grounding from procedural dependency. For example, a passage may Support a claim, a tool call may Depend-on generated parameters, a new observation may Invalidate a plan, and a failed action may Trigger recovery. This compact relation set is intentionally minimal; Section 3 discusses how logs, graphs, and runtime structures encode such dependencies in practice.

Relation to classical provenance. The proposed relation set is compatible with classical provenance modeling but extends it for agent-specific semantics. W3C PROV-DM provides a reusable vocabulary for entities, activities, agents, and relations such as use, generation, derivation, revision, and invalidation World Wide Web Consortium (2013). These constructs naturally correspond to agent artifacts, execution steps, model or tool actors, generated outputs, and memory or state updates. However, LLM-agent provenance also requires semantic relations that classical provenance does not directly encode. For example, Support and Contradict depend on the content-level relationship between evidence and claims, while Depend-

Table 1: Taxonomy of evidence tracing and execution provenance in LLM agents. The table summarizes the six dimensions discussed in Section 2; composite dimensions are separated into subrows for clarity, and representative work is illustrative rather than exhaustive.

Dimension	**Categories**	**Description**	**Representative Work**
Trace sources	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. (2024a); AlSayyad et al. (2026); Zong et al. (2025)
Evidence and execution units	*Evidence units*: Documents, passages, observations, tool outputs, memory items, claims, policies, final answers	Semantic objects that support, contradict, invalidate, or contextualize agent claims, actions, and outputs.	ALCE, FActScore, FEVER, RAGChecker Gao et al. (2023); Min et al. (2023); Thorne et al. (2018); Ru et al. (2024)
	Execution units: Reasoning steps, retrieval calls, tool invocations, parameters, tool manifests, permission scopes, memory operations, actions, inter-agent messages	Procedural objects that record what the agent did, in what order, under which context, and across which execution boundary.	AgentOps, AgentTrace, TRAIL, AgenTracer, Aegis Dong et al. (2024); AlSayyad et al. (2026); Deshpande et al. (2025); Zhang et al. (2025a); Kong et al. (2025)
Provenance relations	Support, derive, depend-on, contradict, invalidate, trigger, update, use, generate	Typed links that connect evidence and execution units into provenance structures.	W3C PROV-DM, Agent-Sentry World Wide Web Consortium (2013); Sequeira et al. (2026)
Tracing granularity and timing	*Tracing granularity*: Run-level, step-level, tool-call-level, parameter-level, claim-level, token/span-level	Determines how fine-grained the recorded provenance structure is.	FActScore, SourceCheckup, Agent-Sentry Min et al. (2023); Wu et al. (2024b); Sequeira et al. (2026)
	Tracing timing: Pre-execution, runtime, post-hoc, continuous	Determines when trace collection, verification, enforcement, or diagnosis occurs.	AgentSpec, Agent-Sentry, TRAIL Wang et al. (2025a); Sequeira et al. (2026); Deshpande et al. (2025)
Representation forms	Structured logs, execution graphs, evidence graphs, claim-support graphs, provenance graphs, runtime state	Determines how traces and typed relations are encoded 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, failure attribution, 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. (2025a); Vijayvargiya et al. (2026)

ON, TRIGGER, and INVALIDATE capture procedural influence, recovery, and epistemic failure during agent execution. Thus, our taxonomy reuses classical provenance as a foundation but specializes it for evidence attribution, tool-use safety, memory lineage, and process-level accountability in LLM agents. Appendix A provides the detailed mapping.

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. (2024b); Costa et al. (2025); Cai et al. (2026); Wang et al. (2025a); 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. (2025); 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 2 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 Positioning Existing Systems in the Taxonomy

Table 2 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, NeuroTaint, and Agent-Sentry operate at runtime over tool calls, parameters, permissions, or information-flow labels, 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 dimensions well: they may record traces, verify claims, enforce tool-use policies, audit executions, or diagnose failures. However, no existing line of work simultaneously spans 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 2: Representative systems mapped to the proposed taxonomy. Rows are grouped by primary trust function to show how existing work covers trace sources, granularity, timing, and representation forms. The mapping highlights a coverage gap: current systems often support tracing, attribution, safety enforcement, debugging, or audit separately, but rarely integrate fine-grained provenance representation with multiple trust functions.

Primary trust	System	Trace source(s)	Granularit	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. (2024a)	Multi-agent, tool	Step	Runtime	Message log
	MemGPT Packer et al. (2023)	Memory	Step	Continuous	Tiered memory
	A-MEM Xu et al. (2026)	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. (2024b)	Retrieval	Claim	Post-hoc	Source–support
	RAGChecker Ru et al. (2024)	Retrieval	Claim	Post-hoc	Claim–evidence
Safety	CaMeL Debenedetti et al. (2025)	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./toke	Runtime	Taint labels
	AgentSpec Wang et al. (2025a)	Tool, env	Tool-call	Pre/Runtime	Policy rules
	AgentBound Bühler et al. (2025)	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. (2026)	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.

Figure 3 provides a running example for this section. It instantiates the taxonomy by showing how evidence acquisition, claim construction, tool execution, and conflict-driven update or recovery can be represented as a typed provenance graph rather than as a flat chronological log.

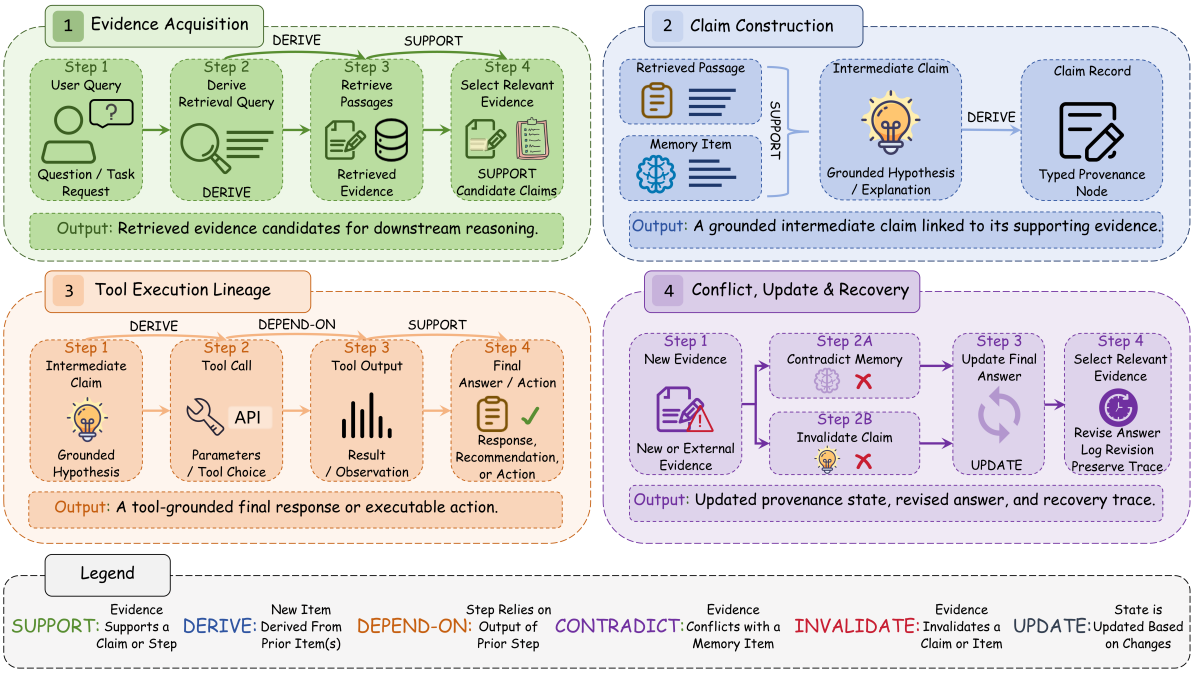

Figure 3: Running example of a provenance graph for an LLM-agent execution. The graph connects user queries, retrieved passages, intermediate claims, tool calls, tool outputs, memory items, new evidence, and final actions through typed relations such as Support, Derive, Depend-on, Contradict, Invalidate, and Update.

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-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., 2025a), whereas Aegis uses positive–negative trajectory pairs to identify faulty agents and error modes (Kong et al., 2025). Thus, 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

Table 3: 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.

work shows that a cited source may be relevant without supporting the specific claim (Wu et al., 2024b); and claim–evidence mapping explicitly distinguishes supported claims, unsupported claims, and omitted evidence (Martin-Boyle et al., 2026).

For provenance representation, these correspond to distinct edge types rather than a single citation relation. A source may be Cited but not Support a claim; 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. 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 form two complementary layers of 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 3 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 provenance-critical questions: 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 provenance-critical objects: 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, demonstrating why parameter-level lineage is needed 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. (2025). 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. (2025). 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 origin of value 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. (2025a). AgentBound provides capability-constrained access control for MCP servers, limiting the resources available to each server Bühler et al. (2025). 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. (2025), while provenance and runtime rules determine whether a particular action should proceed, be restricted, require confirmation, or be blocked Wang et al. (2025a); 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. (2025a). Agent-Sentry uses execution provenance to verify whether

Table 4: 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.

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 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. (2025)	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. (2025a); AgentBound Bühler et al. (2025); MCP-SafetyBench Zong et al. (2025)	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

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, 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 4 summarizes representative systems according to their main focus, provenance object, enforcement timing, and 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, 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. Instead of producing immediate external actions, evidence is retained, transformed, retrieved, and reused 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., 2025b; 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., 2026).

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; 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

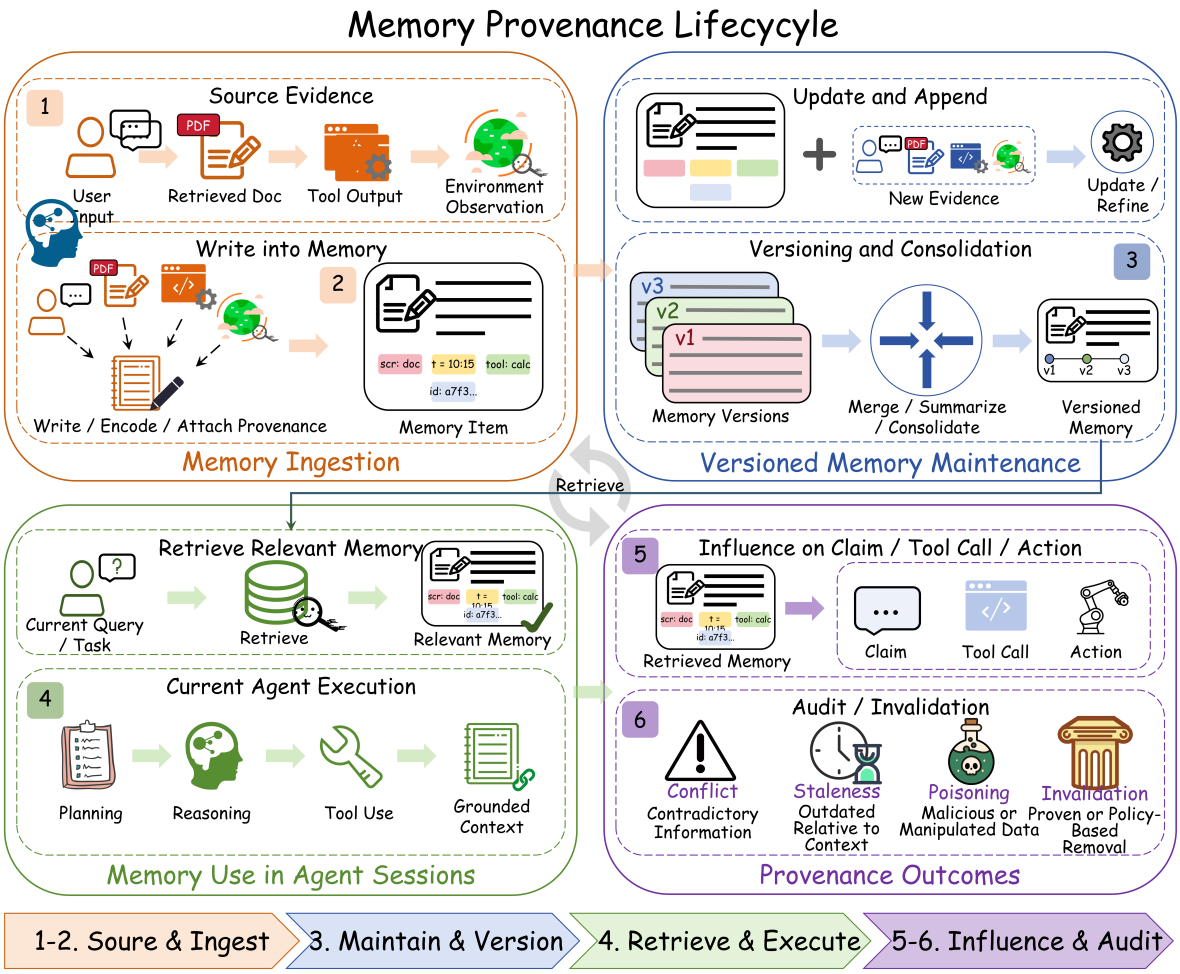

Figure 4: Memory as provenance-bearing evidence across agent sessions. Memory items are written from source evidence, maintained through updates and consolidation, retrieved into later agent executions, used to influence claims and actions, and audited for conflicts, staleness, poisoning, and invalidation.

and 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., 2026).

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., 2025b; Wei et al., 2025; 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., 2026; Chhikara et al., 2025), most memory systems still 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., 2025b). Defense work such as A-MemGuard suggests that memory should support checking and correction, not only input-time filtering (Wei et al., 2025).

Table 5 summarizes this gap. Across representative memory systems and security frameworks, write and retrieval support are common, but conflict handling, staleness detection, contamination tracking, 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. A provenance-bearing memory system must support both.

Table 5: Agent memory through a provenance lens. Existing memory systems support long-term recall, updates, and adaptive behavior, but provenance-specific capabilities such as source attribution, conflict handling, staleness detection, contamination tracking, and evidence-aware verification remain limited. Symbols: ✓ = Yes; ▲ = Partial; ❍ = Limited; ✗ = No; ★ = Attack-focused.

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., 2026)	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	✓	✓	▲	▲	✓

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 few evaluate 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 existing benchmark families tend to specialize in one or two dimensions, such as evidence grounding, tool use, memory, multi-agent interaction, or safety attacks. 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., 2024b). 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 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—but evaluate it only at the answer or context level, not per claim.

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

Legend: **S** Strong coverage **P** Partial coverage **L** Limited or absent coverage. *Rating criteria:* a capability is rated **S** when it is a primary evaluation target with dedicated labels or metrics, **P** when it is supported only indirectly or for a subset of cases, and **L** when it is largely absent.

Key gap. Existing benchmarks cover important but isolated aspects of agent provenance. No benchmark family provides strong end-to-end coverage across evidence labels, tool calls, memory, multi-agent communication, safety attacks, provenance relations, and recovery.

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 full-stack evaluation of evidence tracing and execution provenance remains underdeveloped.

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., 2024b; 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., 2024b). 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., 2025a), and Aegis generates positive–negative trajectory pairs with faulty-agent and error-mode labels (Kong et al., 2025). Together, these works show that provenance can be evaluated not only by trace completeness, but also by its ability to support failure localization and attribution.

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., 2025). 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., 2025a). AgentBound extends this line of work to execution boundaries and access control in MCP-style agent-tool ecosystems (Bühler et al., 2025). 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., 2026; Chhikara et al., 2025; Maharana et al., 2024; Kim 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., 2026); 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 (Kim 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., 2025b; 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 rarely evaluate 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; Kim et al., 2026; Zhang et al., 2025b; 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., 2024a; 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

Table 6: 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, and we mark them as such to avoid conflating them with mature measures. ✓: 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., 2024b).
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, intervention precision.	✓	Tool-use safety, prompt injection, and information-flow control (Ruan et al., 2024; Zhan et al., 2024; Debenedetti et al., 2024; 2025; Costa et al., 2025; Zong et al., 2025).
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., 2026; AlSayyad et al., 2026; Zhang et al., 2025a; Kong et al., 2025).

error, and how responsibility is distributed across agents. MAST analyzes failures from specification errors, inter-agent misalignment, and weak task verification (Cemri et al., 2026), 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., 2025a; Kong et al., 2025). 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., 2024b; Ruan et al., 2024; Debenedetti et al., 2024; Maharana et al., 2024; Deshpande et al., 2025; Cemri et al., 2026). 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 6 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.

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., 2026), we organize these metrics by their evaluation function. Evidence attribution metrics assess claim grounding; execution-provenance metrics 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.

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., 2024b). 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., 2025). Memory and multi-agent benchmarks evaluate recall, personalization, coordination, failure modes, or error attribution (Maharana et al., 2024; Kim et al., 2026; Deshpande et al., 2025; Cemri et al., 2026; Zhang et al., 2025a; Kong et al., 2025). Although recent work has improved realistic tool-safety evaluation and multi-agent failure attribution, existing benchmarks still rarely evaluate 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., 2026). Few benchmarks jointly capture user requests, retrieved evidence, tool calls and outputs, memory reads and writes, inter-agent messages, intermediate claims, final responses, and state changes. Even fewer annotate relations such as Support, Contradict, Depend-on, Update, and Invalidate, 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. Existing benchmarks commonly test whether an agent succeeds, fails, hallucinates, or takes an unsafe action, but rarely evaluate 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., 2026). 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; Kim 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 7 summarizes representative benchmarks and datasets according to their task type, trace availability, evidence labels, tool calls, memory, multi-agent setting, safety attacks, and main metrics.

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 they rarely explain how evidence, tools, memory, observations, messages, and actions jointly shape an agent's final answer or external

Table 7: Benchmarks for evidence tracing and execution provenance. Existing benchmarks cover different parts of the provenance problem, including citation quality, RAG faithfulness, agent execution traces, tool-use safety, memory evaluation, and multi-agent failure analysis. However, few benchmarks jointly provide evidence labels, tool calls, memory operations, multi-agent communication, safety perturbations, and provenance-relation annotations. Symbols: ✓ = Yes; • = Partial; ✗ = No; T = Task-dependent; S = State-based.

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., 2024b)	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., 2026)	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., 2025)	MCP execution boundary	✓	✗	✓	✗	✗	✓	Access-control violations, policy enforcement
LOCOMO (Maharana et al., 2024)	Long-term conversational memory	•	•	✗	✓	✗	✗	Long-range QA, event summarization, memory recall
MemoryArena (Kim et al., 2026)	Multi-session agent memory	✓	•	T	✓	✗	✗	Multi-session task success, memory use

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., 2026). 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., 2024b; Schreieder et al., 2025). 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., 2024b).

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., 2025b; 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., 2024a; 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., 2026). 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., 2025a; Kong et al., 2025). 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; 2025; Costa et al., 2025; Sequeira et al., 2026; Bühler et al., 2025).

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; 2025; Costa et al., 2025; Cai et al., 2026; Wang et al., 2025a; Sequeira et al., 2026; Bühler et al., 2025). 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. RAG benchmarks provide evidence labels but usually omit tools and memory; tool-use safety benchmarks provide attacks and tool calls but often lack claim-level evidence labels; memory benchmarks evaluate recall or personalization but rarely evaluate lineage, conflict handling, or downstream influence; and multi-agent benchmarks analyze failures but often lack 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., 2026; Maharana et al., 2024; Kim et al., 2026). However, no benchmark family 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., 2026). AgenTracer studies responsible-agent attribution from multi-agent trajectories (Zhang et al., 2025a). Aegis further examines responsible-step attribution through contrastive or counterfactual multi-agent traces (Kong et al., 2025). OpenAgentSafety emphasizes realistic tool-use safety risks in open agent settings (Vijayvargiya et al., 2026). MCPSafetyBench extends safety evaluation to protocol-level agent-tool environments (Zong et al., 2025). 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.

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, 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, yet they rarely provide fine-grained claim-level support, semantic influence tracking, memory lineage, cross-agent provenance propagation, or robust tracing under adversarial conditions. 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, Edward Berman, 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=dHng2OOJjr.

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, Parth 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. *arXiv preprint arXiv:2512.15163*, 2025.

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, et al. Why do multi-agent llm systems fail? *Advances in Neural Information Processing Systems*, 38, 2026.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. AgenTracer: Who is inducing failure in the LLM agentic systems? *arXiv preprint arXiv:2509.03312*, 2025a.

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. *arXiv preprint arXiv:2509.14295*, 2025.

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/.

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.

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*, 2024a.

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.

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/.

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.

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.

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.

Kevin Wu, Eric Wu, Ally Cassasola, Angela Zhang, Kevin Wei, Teresa Nguyen, Sith Riantawan, Patricia Shi Riantawan, Daniel E. Ho, and James Zou. How well do llms cite relevant medical references? an evaluation framework and analyses. *arXiv preprint arXiv:2402.02008*, 2024b. URL https://arxiv.org/abs/2402.02008.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. Agentspec: Customizable runtime enforcement for safe and reliable llm agents. *arXiv preprint arXiv:2503.18666*, 2025a. URL https://arxiv.org/abs/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.

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.

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.

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. *arXiv preprint arXiv:2503.18813*, 2025. URL https://arxiv.org/abs/2503.18813.

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.

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.

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, 2026.

Christoph Bühler, Matteo Biagiola, Luca Di Grazia, and Guido Salvaneschi. Securing ai agent execution. *arXiv preprint arXiv:2510.21236*, 2025. URL https://arxiv.org/abs/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*, Findings of ACL, pages 10471–10506. Association for Computational Linguistics, 2024. URL https://doi.org/10.18653/v1/2024.findings-acl.624.

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.

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, and Yang Liu. Prompt injection attack against LLM-integrated applications. *arXiv preprint arXiv:2306.05499*, 2023. URL https://arxiv.org/abs/2306.05499.

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.

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.

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.

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.

Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, 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*, 2025b. doi:10.1145/3748302.

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.

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.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

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, Zeyang Sha, Jingying Wang, Yuhang Liu, Zhehao Huang, and Xiaolin Huang. InjecMEM: Memory injection attack on LLM agent memory systems. OpenReview preprint, 2026. URL https://openreview.net/forum?id=QVX6hcJ2um. Submitted to ICLR 2026.

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 2025b. 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, 2025. 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.

Hyunjin Kim et al. Memoryarena: Benchmarking agent memory in interdependent multi-session agentic tasks. *arXiv preprint arXiv:2602.16313*, 2026. URL https://arxiv.org/abs/2602.16313.

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

Tobias Schreieder, Tim Schopf, and Michael Färber. Attribution, citation, and quotation: A survey of evidence-based text generation with large language models. *arXiv preprint arXiv:2508.15396*, 2025. URL https://arxiv.org/abs/2508.15396.

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 8 makes the relationship to the W3C PROV data model explicit. The upper block aligns agent provenance with PROV entities, activities, agents, and core relations, showing that much of agent execution can reuse a standard provenance vocabulary. The lower block lists 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 Representative Systems Mapped to the Taxonomy

Table 9 places representative systems within the taxonomy dimensions introduced in Section 2. The table is provided as supplementary material rather than as part of the main taxonomy section, because it functions as a system-level comparison rather than as a definition of the taxonomy itself. Three patterns stand out. First, *trace-source* systems such as ReAct, MemGPT, and AutoGen produce rich execution units but implement no trust function on their own; they are substrates that provenance mechanisms build upon. Second, *attribution* systems such as ALCE, FActScore, and SourceCheckup operate post-hoc at claim granularity over retrieval, and rarely touch tools, memory, or actions. Third, *safety* systems such as CaMeL, FIDES, NeuroTaint, and Agent-Sentry operate at runtime on tool parameters, but provide little claim-level evidence support. No single system spans trace sources, fine granularity, runtime timing, an explicit representation, and multiple trust functions at once, which motivates a unified provenance layer.

Table 8: Explicit correspondence between the W3C PROV data model (World Wide Web Consortium, 2013) and the agent-provenance model used in this survey. The upper block maps PROV constructs to agent counterparts; the lower block lists agent-specific semantic relations and units that PROV does not express.

W3C PROV construct	**Agent-provenance counterpart**	**Notes and agent-specific extension**
PROV-aligned core (reusable vocabulary)		
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.
`used`	Depend-on, Use	An action or tool call depends on a parameter, tool output, or memory item; agent provenance adds trust/taint labels on the used 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.
`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 or external-state updates across turns or sessions.
Agent-only (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.

C Additional Details for Open Problems

This appendix expands the open problems discussed in Section 7. The main text presents the research agenda; this appendix provides more detailed design requirements, representative foundations, and evaluation dimensions.

C.1 Trace Schema Requirements

A unified LLM-agent provenance schema should combine operational execution records with semantic evidence relations. Table 10 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.

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

Claim-level provenance should distinguish at least five relations that are often conflated in current systems: citation presence, source relevance, claim support, contradiction, and evidence omission. ALCE, FActScore,

Table 9: Representative agent systems expanded by *primary trust function*. The remaining columns map each system onto selected taxonomy dimensions of Section 2: trace source, granularity, timing, and representation. "Substrate" marks systems that generate traces but do not themselves implement a trust function; "(eval)" marks benchmarks that probe a capability rather than enforce it.

Primary trust	System	Trace source(s)	Granularit	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 (learned)	—
	ToolLLM Qin et al. (2024)	Tool	Tool-call, param	Runtime	API-call path
	AutoGen Wu et al. (2024a)	Multi-agent, tool	Step	Runtime	Message log
	MemGPT Packer et al. (2023)	Memory	Step	Continuous	Tiered memory
	A-MEM Xu et al. (2026)	Memory	Step	Continuous	Linked memory
Attribution	ALCE Gao et al. (2023)	Retrieval	Claim	Post-hoc	Citation links
	SourceCheckup Wu et al. (2024b)	Retrieval	Claim	Post-hoc	Source–support
Verification	FActScore Min et al. (2023)	Retrieval	Claim (atomic)	Post-hoc	Claim–evidence
	RAGAS Es et al. (2024)	Retrieval	Answer	Post-hoc	—
Verification (eval)	τ-bench Yao et al. (2025)	Tool, env	Tool-call	Post-hoc	State log
Safety	CaMeL Debenedetti et al. (2025)	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/toke	Runtime	Taint labels
	AgentSpec Wang et al. (2025a)	Tool, env	Tool-call	Pre/Runtime	Policy rules
	AgentBound Bühler et al. (2025)	Tool	Tool-call	Pre/Runtime	Permissions
Safety (eval)	ToolEmu Ruan et al. (2024)	Tool	Tool-call	Runtime (emul.)	Action log
	InjecAgent Zhan et al. (2024)	Tool, env	Tool-call	Runtime	—
	AgentDojo Debenedetti et al. (2024)	Tool, env	Tool-call	Runtime	—
Safety, Audit	Agent-Sentry Sequeira et al. (2026)	Tool	Parameter	Runtime	Provenance graph
Recovery	Reflexion Shinn et al. (2023)	Reasoning, memory	Step	Continuous	Episodic memory
Debugging	RAGChecker Ru et al. (2024)	Retrieval	Claim	Post-hoc	Claim–evidence
	TRAIL Deshpande et al. (2025)	All sources	Step	Post-hoc	Annotated trace
	MAST Cemri et al. (2026)	Multi-agent	Step	Post-hoc	Failure taxonomy
Debugging (eval)	WebArena Zhou et al. (2024)	Env, tool	Step	Post-hoc	Action/state log
Audit, Debugging	AgentOps / AgentTrace Dong et al. (2024); AlSayyad et al. (2026)	All sources	Step	Post-hoc/Cont.	Structured log

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., 2024b; Schreieder et al., 2025). 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 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).

Table 10: 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	Documents, passages, tool outputs, observations, memory items, policies, intermediate claims, and final claims.	Supports claim-level verification and attribution.
Execution units	Reasoning steps, retrieval operations, tool invocations, parameter-generation steps, memory reads/writes, environment actions, and message exchanges.	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.

Table 11: 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?

C.3 Memory and Multi-Agent Provenance Checklist

Table 11 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., 2024a; Li et al., 2023; Cemri et al., 2026; Zhang et al., 2025a; Kong et al., 2025).

C.4 Runtime Safety and Recovery Requirements

Runtime provenance should support both enforcement and repair. Table 12 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 12: Runtime safety and recovery requirements for provenance-aware agents.

Capability	Provenance requirement	Representative foundations
Prompt-injection 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., 2025; 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., 2025a; Bühler et al., 2025)
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.

Table 13: 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 Support, Contradict, Depend-on, Trigger, Update, Invalidate, unsafe influence, and recovery intervention.
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, asking 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.

C.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 13 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., 2024b; 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., 2025; Maharana et al., 2024; Kim et al., 2026; Deshpande et al., 2025; Cemri et al., 2026; Zhang et al., 2025a; Kong et al., 2025). 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.