AgentTrails: Towards Trust and Reuse for Agentic Tasks

Eden Wu, Sonia Castelo, Yurong Liu, Cláudio T. Silva and Juliana Freire
New York University
{eden.wu,s.castelo,yurong.liu,csilva,juliana.freire}@nyu.edu

ABSTRACT

LLM-powered agents increasingly tackle complex tasks by invoking tools, querying databases, executing code, and manipulating intermediate artifacts. These agents follow trajectories that are typically stored as chronological logs, obscuring the underlying dataflow – the dependencies between their actions and the artifacts they create and manipulate. This limits developers' ability to understand the agents' trails, compare executions, debug failures, and re-use the computations. We present AgentTrails, a prototype system for agent provenance and sensemaking. AgentTrails converts raw trajectories into structured provenance graphs, where tool calls are modeled as computational actions and inputs and outputs as data artifacts. The system supports the comparison of executions by placing multiple provenance graphs on a shared canvas and constructing a joined quotient graph that aligns recurring tools, artifacts, and dependency structures across trajectories. On top of this representation, AgentTrails supports pattern extraction, downstream analysis, and skill abstraction. We demonstrate AgentTrails on real-world agent trajectories, showing that it reveals hidden dependencies, aligns divergent executions, and surfaces recurring tool-use patterns beyond chronological logs.

VLDB Workshop Reference Format:
Eden Wu, Sonia Castelo, Yurong Liu, Cláudio T. Silva and Juliana Freire. AgentTrails: Towards Trust and Reuse for Agentic Tasks . VLDB 2026 Workshop: DASHSys: Systems for Data-centric Agents with Human-in-the-loop.

1 INTRODUCTION

LLM-powered agents are increasingly used in domains such as software engineering, scientific discovery, and data analysis, where they execute tasks through sequences of tool calls, external queries, code execution, and intermediate artifact manipulation [19, 21]. These executions produce rich trajectories containing messages, tool invocations, responses, generated files, and intermediate results that can be useful for debugging failures, analyzing tool-use behavior, comparing agents and models, and curating data for post-training. Repeated structures across successful executions may further reveal reusable skills or workflows. However, because the trajectories are stored as an unstructured, sequential log, they obscure the workflow the agent orchestrates, making it difficult to understand the underlying logic of the agent's actions.

Understanding Agent Traces: Challenges. Raw agent trajectories are typically recorded as chronological textual logs. While they preserve what happened, they are difficult to use for understanding, comparing, and improving agent behavior:

P1 *Raw traces are long and heterogeneous.* A single trace may contain many turns, tool schemas, structured arguments, responses, and generated artifacts; comparing traces as text is tedious and offers no compact overview of tool usage or execution patterns.
P2 *Chronological order hides workflow structure.* Later tool calls may reuse artifacts produced many steps earlier, combine outputs from multiple calls, or branch from intermediate results. Conversely, adjacent calls may be unrelated. Chronological order alone, therefore, does not reveal dependencies, artifact reuse, or workflow structure [17]. P3 *Agent executions are stochastic.* The same task can produce different tool orders, repeated calls, or divergent branches across runs, agents, or models [1, 3, 7, 20, 22]. This makes it difficult to compare trajectories directly as sequences.

From Traces to Provenance: Challenges. A natural abstraction for agent behavior is execution provenance. However, constructing provenance representations from raw traces introduces two additional challenges:

S1 *Raw traces obscure dependency topology.* Agent trajectories rarely contain explicit dependency edges. Recovering provenance therefore requires identifying which tool inputs depend on which prior outputs from heterogeneous evidence such as artifact identifiers, paths, query strings, and semantic references embedded in responses. Reconstruction must be efficient, auditable, and robust across diverse tools. S2 *Multiple provenance graphs lack common alignment anchors.* Comparing provenance graphs requires more than matching tool names or positions: traces may contain repeated calls, missing steps, or alternative branches, so similar workflow stages appear in different structural contexts. Multi-trace analysis requires abstractions that align activities by workflow role and dependency structure.

Our Approach: AgentTrails. We present AgentTrails, a prototype visual analytics system for tool-calling agent provenance and sensemaking. AgentTrails provides three coordinated levels of analysis. First, to address P1 and provide a holistic entry point, AgentTrails offers an UpSet-style overview of tool usage across traces. The view shows which traces invoke which tools, how often tools are called, and how tool usage relates to trace-level metrics such as score, cost, latency, or token count. This enables users to filter, group, and select traces before inspecting detailed provenance. Second, to address P2 and S1, AgentTrails converts a selected raw trajectory into a structured provenance graph. Tool calls are represented as computational actions, while their inputs, outputs, intermediate artifacts, and returned values are represented as data artifacts. The graph is constructed from trace-derived evidence such as arguments, responses, artifact identifiers, paths, filenames, URLs, and reused semantic values. Third, to address P3 and S2, AgentTrails supports multi-trace comparison through a joined quotient graph. It abstracts tool calls into activity capsules, clusters similar

Proceedings of the VLDB Endowment. ISSN 2150-8097.

capsules across traces, and aligns recurring tools, artifacts, and dependency structures while preserving trace-specific branches.

AgentTrails includes an LLM-based analysis agent that inspects the reconstructed provenance graphs and joined quotient graphs to summarize observed patterns and explain selected branches. The agent supports lightweight downstream sensemaking, while the core workflow remains grounded in trace-derived structures.

This paper makes three preliminary contributions: i) A provenance framing for tool-calling agent sensemaking, focused on recovering dependency topology from chronological traces. ii) A trace-agnostic approach that extracts and infers dependencies from structural and semantic evidence. iii) A joined provenance abstraction for aligning multiple traces and exposing shared workflows, divergent branches, and low-support behaviors.

2 RELATED WORKS

Workflows and Provenance. Provenance has been studied in the context of scientific workflow systems to capture workflow evolution, execution context, and data lineage with the goal of supporting reproducibility and reuse [6]. Approaches have been proposed that use visualization to make sense of provenance, including summaries of workflow collections [8] and to compare machine learning pipelines [12]. These systems assume workflows are explicitly defined or recoverable from controlled environments, while tool-calling agent trajectories arrive as chronological logs with implicit dependencies and stochastic executions. AgentTrails addresses this gap by focusing on provenance reconstruction and multi-trace analysis of heterogeneous agent trajectories.

Agent Trajectory Analysis, Provenance, and Reuse. Recent work has explored *agent trajectories* as objects of analysis. PROV-AGENT models provenance for agentic workflows by instrumenting the running framework, one graph per run [17]; CHIEF, TRAIL, and AgenTracer transform flat logs into structured representations for debugging [4, 18, 23]; and AgentLens, Agent Trajectory Explorer, SeaView, and Graphectory support visualization and process-level analysis of agent behavior [2, 5, 9, 10]. These efforts largely focus on single-run inspection or domain-specific settings. In contrast, AgentTrails reconstructs provenance post-hoc from raw logs without instrumentation and—uniquely—aligns multiple executions through a joined graph, supporting downstream pattern extraction and sensemaking. On the reuse front, ExpeL, Reflexion, and Trace2Skill use prior trajectories for workflow induction, reflection, or skill extraction [11, 16, 24]. AgentTrails is complementary: rather than reusing trajectories as flat sequences, it exposes their internal dependency structure, enabling identification of reusable motifs while pruning redundant tool calls.

3 SYSTEM OVERVIEW

AgentTrails targets raw agent traces stored as long, heterogeneous event logs. To address P1, it normalizes each trace into a sequence of actions while preserving evidence fields such as arguments, responses, artifact identifiers, paths, URLs, returned objects, and reused semantic values. This design does not assume a fixed agent architecture or tool schema, allowing AgentTrails to support traces from heterogeneous frameworks and domains.

Trace Overview with Tool Coverage. The first stage provides a global, tool-agnostic overview of selected traces. For a trace $T_r = \langle c_{r,1}, \ldots, c_{r,n_r} \rangle$, where $c_{r,i}$ denotes the i-th tool call of trace r together with its arguments and response, and tool set $\mathcal{U}$, AgentTrails builds a coverage matrix where each entry $M_{r,u} = |\{i : \text{tool}(c_{r,i}) = u\}|$ counts how many times tool u is invoked in trace r. Rows encode traces, columns encode tools, and cell values encode repeated calls. The matrix is shown as an UpSet-style overview (Fig. 4A), with a top bar chart for total tool frequency or Shapley-style tool impact, and a side rail for trace-level metrics such as score, cost, latency, or token count. This view does not infer dependencies; it supports holistic filtering, grouping, and trace selection before graph-level analysis.

Provenance Graph Construction. To address P2 and S1, AgentTrails converts a selected chronological trace into a provenance graph that makes producer-consumer dependencies explicit (Fig. 1). For a trace T, the graph is $G_T = (A_T \cup E_T,\ R_T)$, where A_T contains tool-call activities and E_T contains recovered entities such as inputs, outputs, artifacts, and returned values. The edge set $R_T \subseteq (A_T \times E_T) \cup (E_T \times A_T)$ keeps the graph bipartite: *generatedBy* edges are always emitted from each activity to its outputs, while *usedBy* and weaker *informedBy* edges—the reconstruction targets—link entities to later activities that consume them. The key difficulty is that raw traces rarely provide dependency edges directly. AgentTrails therefore treats graph construction as evidence-based reconstruction. It first creates a deterministic skeleton from exact evidence: output entities are extracted from response metadata such as identifiers, paths, filenames, URLs, names, or returned objects. A dependency is added when a later call explicitly references an earlier entity:

$$e_j \rightarrow a_i \quad \text{if} \quad K(e_j) \cap \text{refs}(c_i.\text{args}) \neq \emptyset,\ j < i,$$

where $K(e_j)$ denotes the recovered keys for entity e_j—identifying strings from its producing response, e.g., a storage id, filename, or URL. This step captures high-precision artifact reuse while enforcing temporal validity, keeping the graph acyclic: agent loops surface as repeated activities, which the joined graph aggregates with per-trace multiplicity.

AgentTrails also extracts weaker semantic evidence from shared values, query terms, table columns, domain objects, or tokens that appear in earlier responses and later arguments. These matches are stored as dependency candidates rather than asserted as ground truth. A constrained LLM refinement step receives the draft graph and candidate relations, then returns graph patch operations. Only patches that preserve valid node references, temporal order, and the activity/entity schema are accepted. This design keeps the provenance graph auditable: exact edges, semantic candidates, and LLM-refined edits remain distinguishable.

Two questions remain for scaling this methodology: provenance quality needs gold dependency benchmarks, and asking LLMs to infer dependencies directly from full traces does not scale. AgentTrails therefore retrieves likely candidates first—via indexed artifact keys, value sketches, schema-aware blocking, and semantic matching—and uses an LLM only to refine or explain ambiguous ones. As a first step, we hand-annotated 10 traces with 234 gold dependency edges; preliminary results are promising and a full staged evaluation is underway.

Multi-Trace Provenance Graph. To address P3 and S2, AgentTrails constructs a joined provenance graph over selected traces.

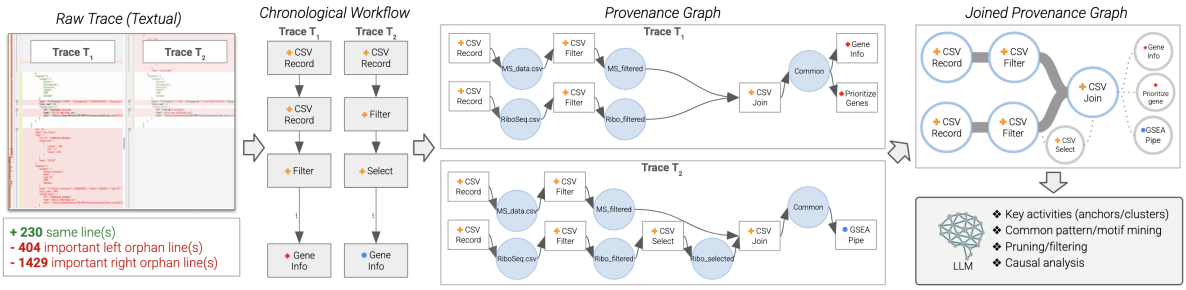

Figure 1: AgentTrails abstracts raw traces into chronological workflows that expose the sequence of actions, then reconstructs a provenance graph that represents actions as activities and artifacts as entities. For multi-trace analysis, AgentTrails aligns similar activities across provenance graphs into a single joined provenance graph, enabling downstream tasks.

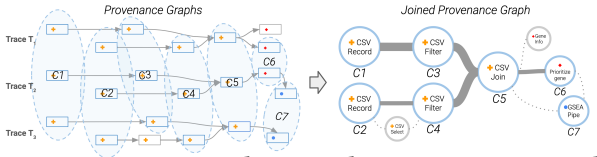

Figure 2: AgentTrails aligns similar activities across multiple provenance graphs into clusters (left), then merges each cluster into a joined anchor (right).

This stage targets the alignment problem: different traces may solve the same task with different tool orders, repeated calls, missing steps, or alternative branches. Thus, comparing traces by raw sequence or exact tool name is insufficient.

As shown in Fig. 2, AgentTrails first abstracts each activity into an activity capsule, which compactly describes the activity's tool identity, input/output structure, local graph context, and upstream root lineage. Capsules are then clustered across traces using a weighted similarity over these features:

$$S(\kappa_i, \kappa_j) = \sum_m w_m \, s_m(\kappa_i, \kappa_j),$$

where κ_i, κ_j are capsules and each $s_m \in [0, 1]$ compares one feature—tool tokens, input/output structure, graph context, root lineage, or evidence keywords—via Jaccard or cosine similarity, with weights w_m summing to one. Each cluster becomes a joined activity node. Original provenance edges are then remapped based on the cluster assignments: if two original edges connect activities assigned to the same source and target clusters, they are aggregated into one joined edge.

Joined nodes and edges store supporting traces, support counts, per-trace multiplicity, representative members, and optional score summaries. High-support structures reveal recurring workflow motifs, while low-support structures expose trace-specific branches, redundant operations, or anomalous paths. The joined graph therefore summarizes a set of runs at the provenance level rather than as an aggregate tool sequence.

This abstraction raises a follow-up question: joined provenance construction can be designed in many ways. Prior work on workflow analogies suggests that graphs may be aligned by operators, data dependencies, execution roles, or higher-level workflow intent [14]. AgentTrails takes a first step by clustering activity capsules into joined anchors, but future work should explore alternative abstractions and scalable candidate alignment through blocking, retrieval, and role-aware graph features.

Interactive Visualization, Filtering, and Copilot. AgentTrails provides coordinated views for overview, inspection, and comparison. The UpSet view supports trace selection, metadata grouping, score comparison, and tool-usage analysis. The single-trace view exposes recovered activities, entities, arguments, responses, and edge evidence. The joined view compares multiple traces, using node size for activity support, edge width for dependency support, and color to preserve trace membership. Users can fade or prune nodes and edges by support or score, enabling them to identify dominant workflows, low-support anomalies, and differences between high- and low-scoring traces.

In addition to these views, AgentTrails includes a provenance copilot for lightweight interactive sensemaking. Given the current selection, the copilot can inspect graph structures, retrieve tool inputs and outputs, expand joined nodes to their trace members, and summarize visible patterns. The copilot does not define provenance; it helps users interpret trace-derived, auditable graphs.

4 USAGE SCENARIOS

Inspecting a SciAgentGym Physics Trace. We first demonstrate AgentTrails on a SciAgentGym task. SciAgentGym evaluates multi-step scientific tool use by LLM agents and provides more than 1,780 tools across Physics, Chemistry, Materials Science, Life Science, and Astronomy [15]. We analyze task 27 from Physics, which asks for the hyperfine transition frequency of a hypothetical ground-state hydrogen atom with electron spin $\frac{3}{2}$. The raw trace contains structured tool calls, numerical outputs, and generated visualizations, making the scientific workflow difficult to verify from text or chronological order alone.

After loading the trace into AgentTrails, the reconstructed provenance graph exposes a clear fan-in/fan-out computation pattern (Fig. 3). Independent quantities-physical constants, electron and proton g-factors, and the wavefunction value at the nucleus-converge into the calculation of the hyperfine constant A. This intermediate result then branches into two energy-shift calculations for the $F = 1$ and $F = 2$ states. These shifts, together with the angular-momentum states, support the final transition-frequency calculation and the visualization outputs. This dependency structure is not apparent from the raw sequential trace, where related calls may be separated and where the scientific role of each intermediate output is buried in tool responses.

Comparing the reconstructed graph with the benchmark's expected tool use shows that the trace covers the required scientific

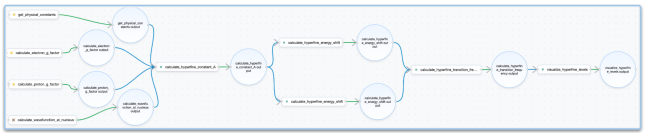

Figure 3: SciAgentGym usage scenario. AgentTrails exposes the dependency structure hidden in a raw physics trace.

Figure 4: Discovera usage scenario. AgentTrails helps users move from trace-level overview (A), to single-trace provenance inspection (B), to joined multi-trace comparison (C).

steps. More importantly, the provenance graph reveals how those steps are connected: which upstream quantities feed the hyperfine constant, how A is reused in multiple downstream computations, and how the final frequency is grounded in earlier tool results. This scenario demonstrates how AgentTrails supports single-trace validation and scientific workflow inspection, rather than benchmarking the underlying model.

Discovera Gene-Set Exploration. We also demonstrate AgentTrails on Discovera traces for Signature-to-Mechanisms analysis. Discovera is a workflow-aligned scientific agent in which the LLM orchestrates deterministic tools, stores intermediate artifacts, and grounds reasoning in tool outputs [13]. We analyze task `s2m_task_016`. In the UpSet overview, we group runs by task ID and rank traces by score (Fig. 4A). The top trace uses fewer tool calls than several alternatives, motivating provenance-level inspection.

Opening the top trace reveals a compact workflow: two Wang 2025 Cancer Cell datasets are ingested with `csv_record`, filtered by `FDR < 0.05`, projected through `csv_select`, joined with `csv_join`, refined to remove global differentially expressed genes, and analyzed with `ora_pipe`. The provenance graph exposes this as a coherent path from data ingestion to enrichment (Fig. 4B).

We then select the next two ranked traces and construct a joined provenance graph. Filtering the joined view by score highlights the high-scoring execution as a direct, well-supported dataflow, while lower-scoring traces contain branches and detours away from the core data-processing path (Fig. 4C). Through node support, edge support, and trace membership encodings, AgentTrails helps distinguish shared workflow structure from trace-specific deviations: the lower-scoring runs introduce extra calls off the dominant provenance path—concrete candidates for pruning or prompt fixes.

5 CONCLUSION

AgentTrails is preliminary work toward a principled infrastructure for agent provenance and sensemaking. We presented a provenance framing for tool-calling agent traces, evidence-based dependency-graph reconstruction from chronological logs, and a joined quotient graph for aligning and comparing executions. Usage scenarios on SciAgentGym and Discovera traces demonstrate that the system surfaces dependency structure and workflow patterns that are not recoverable from raw sequential logs. Several important questions remain open: provenance quality lacks ground-truth benchmarks; the capsule similarity function and its weights require principled tuning and evaluation; and scalability to very long traces or large trace collections has not been assessed. We view AgentTrails as a foundation for future work on agent debugging, workflow reuse, and skill extraction grounded in auditable, trace-derived provenance.

ACKNOWLEDGMENTS

This work was supported in part by DARPA ASKEM (HR0011262087), ARPA-H BDF, and NSF (OAC-2411221). The views, opinions, and findings expressed are those of the authors and should not be interpreted as representing the views or policies of these agencies.

REFERENCES

[1] Bjarni Haukur Bjarnason, André Silva, and Martin Monperrus. 2026. On Randomness in Agentic Evals. In *ICLR 2026 Workshop on Agents in the Wild.*
[2] Timothy Bula, Saurabh Pujar, Luca Buratti, et al. 2025. SeaView: Software Engineering Agent Visual Interface for Enhanced Workflow. arXiv:2504.08696
[3] Yanda Chen, Joe Benton, Ansh Radhakrishnan, et al. 2025. Reasoning Models Don't Always Say What They Think. arXiv:2505.05410
[4] Darshan Deshpande, Varun Gangal, Hersh Mehta, et al. 2025. TRAIL: Trace Reasoning and Agentic Issue Localization. arXiv:2505.08638
[5] Michael Desmond, Ja Young Lee, Ibrahim Ibrahim, et al. 2025. Agent Trajectory Explorer: Visualizing and Providing Feedback on Agent Trajectories. *Proc. AAAI Conf. Artif. Intell.* 39, 28 (2025). https://doi.org/10.1609/aaai.v39i28.35350
[6] Juliana Freire, David Koop, Emanuele Santos, and Cláudio T. Silva. 2008. Provenance for Computational Tasks: A Survey. *Computing in Science and Engineering* 10, 3 (2008), 11–21.
[7] Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, et al. 2024. AI Agents That Matter. *Trans. Mach. Learn. Res.* 2025 (2024).
[8] David Koop, Juliana Freire, and Cláudio T. Silva. 2013. Visual summaries for graph collections. In *IEEE PacificVis.* https://doi.org/10.1109/PacificVis.2013.6596128
[9] Shuyang Liu, Yang Chen, Rahul Krishna, et al. 2026. Process-Centric Analysis of Agentic Software Systems. *Proc. ACM Program. Lang.* 10, OOPSLA1 (2026). https://doi.org/10.1145/3798271
[10] Jiaying Lu, Bo Pan, et al. 2025. AgentLens: Visual Analysis for Agent Behaviors in LLM-Based Autonomous Systems. *IEEE Trans. Vis. Comput. Graph.* 31, 8 (2025).
[11] Jingwei Ni, Yihao Liu, Xinpeng Liu, et al. 2026. Trace2Skill: Distill Trajectory-Local Lessons into Transferable Agent Skills. arXiv:2603.25158
[12] Jorge Piazentin Ono, Sonia Castelo, Roque López, et al. 2020. PipelineProfiler: A Visual Analytics Tool for the Exploration of AutoML Pipelines. *IEEE Trans. Vis. Comput. Graph.* 27 (2020). https://api.semanticscholar.org/CorpusID:218470098
[13] Daniela Pinto Veizaga, Aécio Santos, Eden Wu, et al. 2026. Discovera: A Workflow-Aligned AI Agent for Signature-to-Mechanisms Analysis. NE Agents Day 2026 Workshop Submission, Submission 18.
[14] Carlos Scheidegger, Huy Vo, David Koop, et al. 2007. Querying and Creating Visualizations by Analogy. *IEEE Trans. Vis. Comput. Graph.* 13, 6 (2007). https://doi.org/10.1109/TVCG.2007.70584
[15] Yujiong Shen, Yajie Yang, Zhiheng Xi, Binze Hu, et al. 2026. SciAgentGym: Benchmarking Multi-Step Scientific Tool-Use in LLM Agents. In *ICML.*
[16] Noah Shinn, Federico Cassano, Ashwin Gopinath, et al. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. In *NeurIPS.* arXiv:2303.11366
[17] Renan Souza, Amal Gueroudji, Stephen DeWitt, et al. 2025. PROV-AGENT: Unified Provenance for Tracking AI Agent Interactions in Agentic Workflows. In *2025 IEEE International Conference on eScience (eScience).* 467–473.
[18] Yawen Wang, Wenjie Wu, Junjie Wang, et al. 2026. From Flat Logs to Causal Graphs: Hierarchical Failure Attribution for LLM-based Multi-Agent Systems. arXiv:2602.23701
[19] Jiaqi Wei, Yuejin Yang, Xiang Zhang, et al. 2025. From AI for Science to Agentic Science: A Survey on Autonomous Scientific Discovery. arXiv:2508.14111
[20] Shunyu Yao, Noah Shinn, Pedram Razavi, et al. 2025. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. In *ICLR.*
[21] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, et al. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *ICLR.*
[22] Jiayi Yuan, Hao Li, Xinheng Ding, et al. 2025. Understanding and Mitigating Numerical Sources of Nondeterminism in LLM Inference. In *NeurIPS.*
[23] Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, et al. 2026. AgenTracer: Who Is Inducing Failure in the LLM Agentic Systems?. In *ICLR.*
[24] Andrew Zhao, Daniel Huang, Quentin Xu, et al. 2024. ExpeL: LLM agents are experiential learners. In *AAAI.* https://doi.org/10.1609/aaai.v38i17.29936