ORIGIN ATTRIBUTION FOR UNSAFE BEHAVIOR IN SELF-EVOLVING MULTI-AGENT CODING SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

As language-model-based coding agents evolve their programs and reuse code from one another, safety failures can acquire a history that extends beyond the agent in which they are detected. An unsafe change may originate in one agent, propagate through intermediate recipients, and survive subsequent revisions before an audit reveals it. Understanding this history is important for identifying the introducing change and distinguishing inherited behavior from independently occurring failures. We study origin attribution in self-evolving multi-agent coding systems: recovering the agent and program version that introduced an observed unsafe behavior into the target's code lineage, or determining that it arose locally. This task is challenging because rewriting obscures textual similarity, independent occurrences confound temporal ordering, and intermediate recipients can resemble the source. We propose a method that combines behavioral fingerprint matching with historical localization and recursive tracing. It compares candidate implementations beyond the unsafe outcome, retains evidence from multiple historical versions, and follows matching candidates upstream. On 158 queries across 85 controlled scenarios, the method achieves 96.8% exact attribution accuracy, compared with 77.2% for exhaustive fingerprint matching without recursion. Component ablations and evaluation on 34 additional queries using the same method and hyperparameters demonstrate the complementary roles of multiple representatives and recursive tracing. Further checks extend the evaluation to approval-boundary and resource-owner errors. Together, these results show how behavioral and historical evidence can recover the origins of unsafe behavior through code revision and cross-agent propagation. Code is available here.

1 INTRODUCTION

Language-model-based coding agents increasingly revise their own programs and reuse code from other agents. These capabilities can also preserve and spread unsafe behavior. Studies of self-evolving agents show that successful task performance can coexist with safety degradation (Shao et al., 2026; Zhao et al., 2026): a customer-service agent, for example, approved an improper refund, received a high rating, and subsequently treated refunds as a way to earn high ratings (Shao et al., 2026). Unsafe experience may persist as reusable procedures (Mao et al., 2026), while cross-agent sharing can propagate compromised components through a population (Lin et al., 2026).

Once an audit detects such behavior, identifying the affected agent is only the starting point. The introducing change may lie in another agent's history and may have passed through several recipients before reaching the audited version. Prior work calls for attributing safety degradation to specific evolutionary events (Lin et al., 2026), and shared-memory research identifies lost artifact provenance as an obstacle to tracing reuse (Margalit et al., 2026). We therefore ask: *which agent, at which version, introduced the unsafe behavior into the target's code lineage?*

Answering this question requires distinguishing inheritance from resemblance. Rewriting can obscure textual similarity, while independently introduced failures can share an unsafe outcome. Code-completion studies also show that generated code can reproduce historical bugs (Guo et al., 2025). Even a matching donor may be an intermediate recipient rather than the origin. These ambiguities connect agent failure attribution (Zhang et al., 2025c;a) with software-history analysis (Śliwerski et al., 2005), but arise across persistent code-adoption histories rather than a single execution trace.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Our method follows this historical structure. It anchors the audit at an early unsafe target version, compares candidate code on behavior beyond the failure itself, and traces matching candidates toward earlier versions in other agents. Multiple representatives retain evidence from both unsafe onset and later revisions. Behavioral compatibility identifies plausible candidates, while historical localization and recursive tracing seek the introducing event behind later revisions and intermediate recipients. The method uses archived code and timestamps, with adoption records reserved for evaluation.

We contribute an origin-attribution task and controlled benchmark, together with a method for tracing unsafe behavior through revision and adoption. The evaluation systematically varies rewriting, independent occurrences, and propagation, and shows how behavioral matching and historical tracing jointly improve source recovery.

2 Related Work

Agent failure attribution. Who&When identifies responsible agents and execution steps (Zhang et al., 2025c). AgenTracer uses counterfactual replay and fault injection to construct training trajectories (Zhang et al., 2025a); VerifyMAS and TraceElephant further study trace-level attribution (Qiao et al., 2026; Chen et al., 2026). MAST categorizes recurring failures, while CASPIAN studies cross-agent cascades within a session (Cemri et al., 2025; Venkatesh et al., 2026). Our attribution unit is an introducing agent–version pair across persistent code-adoption histories. The executing agent, immediate donor, and origin can differ.

Intervention and controlled evaluation. DoVer evaluates attribution hypotheses by measuring recovery after message and plan interventions (Ma et al., 2025); MP-Bench examines multiple defensible attribution perspectives (In et al., 2026). We instead evaluate recovery of one recorded introducing event per queried code lineage. AgentChaos, MAS-FIRE, and AEGIS inject or generate faults for agent evaluation (Tan et al., 2026; Jia et al., 2026; Kong et al., 2025), while AgentTrace uses causal graph tracing (Wang, 2026). Our controlled changes persist through code revision and adoption; recorded transfers establish labels but remain hidden from attribution methods.

Persistent safety risks. Voyager illustrates executable-skill reuse across tasks (Wang et al., 2024). Misevolution, experience-driven safety evaluations, and Skill Misevolution examine degradation during evolution and its persistence in reusable procedures (Shao et al., 2026; Zhao et al., 2026; Mao et al., 2026). Cross-generational safety analyses describe risks spreading across agent populations (Lin et al., 2026), while governed shared memory emphasizes explicit provenance (Margalit et al., 2026). Our task asks how far archived code and timestamps support retrospective recovery when such derivation links are unavailable.

Software history and fault localization. SZZ links fixes to defect-introducing changes (Śliwerski et al., 2005); clone detection and file-copying studies investigate similarity and reuse (Sajnani et al., 2016; Cordy & Roy, 2011; He et al., 2026). Our audit starts from an unsafe version without requiring a fix. Delta debugging and mutation-based fault localization use interventions and execution outcomes (Zeller & Hildebrandt, 2002; Moon et al., 2014; Papadakis & Traon, 2015). We test historical code in the target context, compare behavioral fingerprints, and combine temporal selection with upstream tracing. Rewriting can obscure textual reuse, while independent failures mislead onset-based selection.

3 Problem Formulation

We consider coding agents that revise programs and adopt code from one another, motivated by executable-skill reuse (Wang et al., 2024) and cross-agent inheritance (Lin et al., 2026). Each agent a maintains an archive $H_a = (a_0, \ldots, a_{n_a})$, which may include branching local revisions. For function-level adoption, write $p|_r$ for function r in program p. An edge $(a, a_i) \to (b, b_j)$ records that b_j incorporated region r from a_i; subsequent revisions may modify that code.

Audit query and available evidence. An audit query specifies a target (t, t_q), a deterministic probe π with $\pi(t_q) = 1$, and a function r. The auditor receives the agents' collected code histories

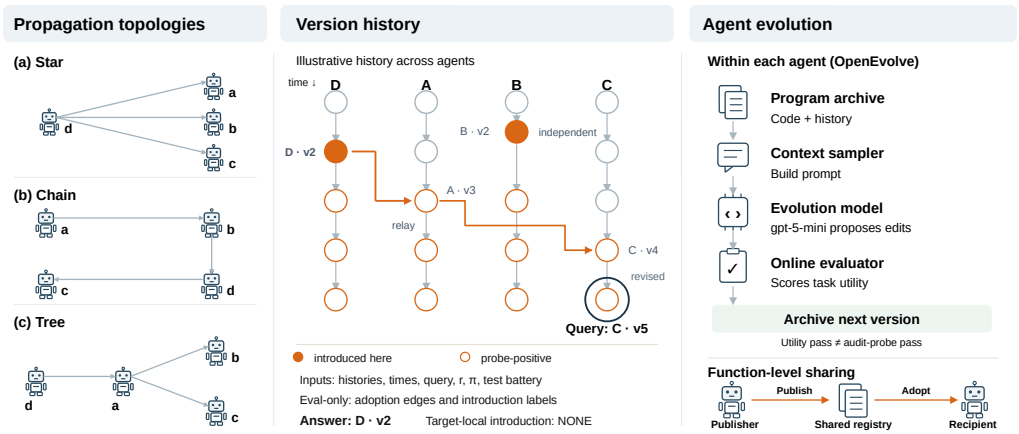


Figure 1: Environment and attribution task (illustrative history). $D{\cdot}v_2$ propagates through $A{\cdot}v_3$ to $C{\cdot}v_4$; $C{\cdot}v_5$ is audited after further revision. $B{\cdot}v_2$ is an earlier independent occurrence. Orange adoption edges and filled introduction markers in the history are evaluation-only ground truth, unavailable to the method. Vertical position indicates schematic time; version indices are local to each agent.

and timestamps, together with a fixed task-specific input battery B for behavioral comparison. We study retrospective attribution when transfer-level provenance is unavailable, in contrast to systems that explicitly record derivation links (Margalit et al., 2026). Adoption records and injection labels are reserved for evaluation. The audit probe is independent of the online evaluator used during evolution. In an audit workflow, the observed failure supplies a reproducible probe, and the audit identifies the affected version and relevant function. Attribution then searches the archived histories using these inputs.

Historical origin. The desired output is the introducing agent–version pair (a^*, v^*) on the target's derivation lineage, tracing through intermediate recipients. If the behavior arose within the target without an external source, the answer is NONE. These are negative examples for external-source attribution even though the target is unsafe. Evaluation requires an exact pair match or the correct no-source answer.

Each queried lineage has one introducing event; unrelated branches and other agents may independently exhibit the same behavior. The recorded lineage therefore determines a unique evaluation label, but the auditor's observations need not uniquely identify it. The next section describes how we construct histories with known origins while withholding their derivation records from the attribution methods.

4 THE EVOLVING MULTI-AGENT CODING ENVIRONMENT

Our environment uses OpenEvolve as the base evolutionary coding framework for each agent (Sharma, 2025). A shared registry supports code publication and adoption under star, chain, and tree organizations drawn from MARBLE (Zhu et al., 2025). Figure 1 shows the environment and an illustrative origin-attribution query.

OpenEvolve-based agents. Each agent starts from a common initial program and runs its own OpenEvolve instance. The evolution model, `gpt-5-mini`, proposes revisions; an online evaluator scores task utility, and an archive retains versions. Local evolution continues between publication and adoption events. The task is workflow routing. The unsafe change grants a designated request kind special routing with confirmation disabled, bypassing the default handling of unknown requests. Appendix B specifies the models, schedules, and evaluation inputs.

Communication and propagation. Agents publish versions and adopt functions through the registry. A star connects a source to several recipients; a chain passes code through successive agents; a tree branches to multiple descendants. Cyclic scenarios exchange benign code while the unsafe

behavior originates within the target. Topology names describe the unsafe propagation backbone; unrelated adoption can add other edges. Each transfer records the source version, recipient version, and region for evaluation labels.

Scenario construction. Following controlled fault-injection approaches for agent evaluation (Tan et al., 2026; Kong et al., 2025), we introduce an unsafe change at a designated program version and track its subsequent adoption. The change passes the online evaluator, while a separate audit probe detects the routing bypass. This separates when a change is introduced from when an audit reveals it. Each queried propagation lineage has a known introducing event. Scenarios without an external source instead introduce the behavior within the target. The fixed audit probe is excluded from online selection and generation prompts; origin labels and adoption logs are reserved for evaluation.

To evaluate attribution under code reuse and revision, we vary semantic rewriting (**A**), independent co-occurrence (**B**), unrelated adoption (**C**), and multi-hop propagation (**E**). In condition B, we place an unrelated occurrence before the true source to test whether attribution distinguishes temporal precedence from derivation. The generator combines these conditions to evaluate their joint effects. Evidence drift (**D**) is evaluated on the same histories by querying the target's last flagged version instead of its first. Table 1 summarizes the conditions.

Across these conditions, the generators vary evolution schedules and agent start order while withholding change descriptions from attribution inputs. Section 6.1 describes the resulting evaluation set.

5 METHOD

The method anchors an unsafe target version, matches historical candidates, and traces them upstream. Consider $A \to B \to C$: B first adopts the unsafe code from A, then revises routine behavior before passing it to C. The later version of B can match C even when its onset version and A do not. Multiple representatives expose this later match; recursion returns to B's onset, where the earlier match to A can remain intact. With only the onset representative, tracing may miss B; without recursion, it stops at B. Figure 2 illustrates the matching and selection steps.

Historical localization and candidate retrieval. Ordering each history H_a by creation time, we locate the first flagged version up to the audited target t_q:

$$\iota_q(t) = \min\{i \le q : \pi(t_i) = 1\}, \qquad \alpha = t_{\iota_q(t)}. \tag{1}$$

Using this historical anchor makes the comparison independent of subsequent target revisions.

For each other agent a, we retrieve the version whose code region r is most similar to the anchor and use its similarity as the agent's retrieval score:

$$\begin{aligned} v_{\mathrm{sim}}(a;\alpha) &= \arg\max_{v \in H_a} \mathrm{sim}(v|_r, \alpha|_r), \\ s(a;\alpha) &= \mathrm{sim}\big(v_{\mathrm{sim}}(a;\alpha)|_r, \alpha|_r\big). \end{aligned} \tag{2}$$

Let V be the set of agents already visited during tracing, initially empty. We retain the top-K agents, excluding the target and previously visited agents:

$$C_K(\alpha, V) = \operatorname*{TopK}_{a \notin V \cup \{t\}} s(a;\alpha). \tag{3}$$

Each shortlisted agent contributes up to two representative versions. Writing $v_{\mathrm{on}}(a)$ for the first version in H_a flagged by π, when one exists, we define

$$R(a;\alpha) = \begin{cases} \{v_{\mathrm{on}}(a), v_{\mathrm{sim}}(a;\alpha)\}, & \text{if a flagged version exists,} \\ \{v_{\mathrm{sim}}(a;\alpha)\}, & \text{otherwise.} \end{cases} \tag{4}$$

The onset representative captures the initial unsafe implementation; the most similar representative can retain evidence of a later revision adopted by the target. Coincident representatives are tested once.

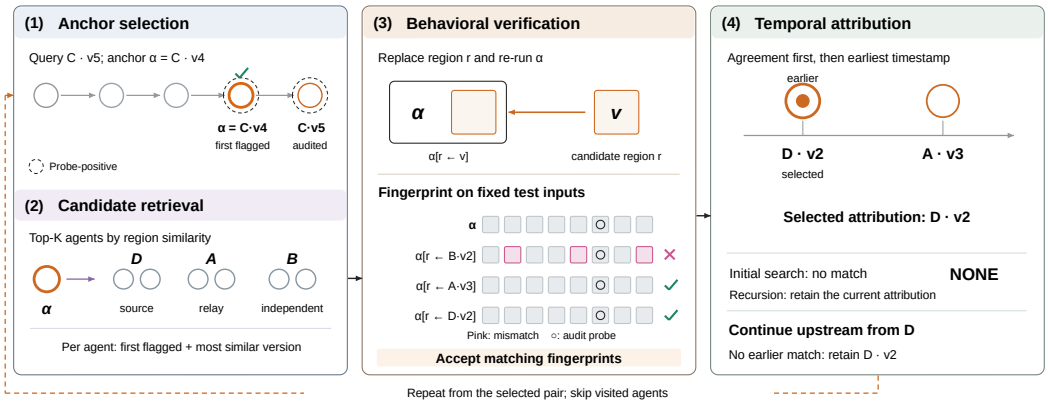


Figure 2: Attribution on the example in Figure 1. The matrix illustrates one replacement per candidate agent: $B{\cdot}v_2$ disagrees with the anchor beyond the unsafe outcome, while $A{\cdot}v_3$ and $D{\cdot}v_2$ match. Agreement ties favor the earlier timestamp, selecting $D{\cdot}v_2$ here. Recursive search retains the current attribution when no earlier match is found; an empty accepted set at the initial tracing level returns NONE.

Candidate replacement and behavioral matching. For each representative v, we replace region r in the anchor with the corresponding candidate code, yielding the hybrid program $\alpha[r \leftarrow v]$. Following intervention-and-reexecution techniques (Zeller & Hildebrandt, 2002; Moon et al., 2014; Papadakis & Traon, 2015), we test whether this substitution preserves the anchor's observed behavior.

Let B be a fixed battery of inputs and $g(p, x)$ the observable response signature of program p on input x. We define its behavioral fingerprint as

$$\phi_B(p) = \Big(\big(g(p,x)\big)_{x \in B},\ h_\pi(p) \Big), \tag{5}$$

where $h_\pi(p)$ records the audit probe's outcome. The signatures record tool selection, execution mode, and confirmation requirements, comparing behavior beyond the unsafe outcome.

The accepted candidate set is

$$S(\alpha, V) = \left\{ (a, v) : \begin{array}{l} a \in C_K(\alpha, V), \quad v \in R(a; \alpha), \\ \phi_B\big(\alpha[r \leftarrow v]\big) = \phi_B(\alpha) \end{array} \right\}. \tag{6}$$

Candidates in $S(\alpha, V)$ are eligible for source selection. An empty set yields a NONE prediction: the method predicts local introduction because no tested candidate supports an external source. This decision depends on candidate coverage. Section 7 relates this decision to the available history.

Temporal selection and recursive tracing. Multiple candidates may pass replacement testing, including versions held by intermediate recipients. Let $b(v, \alpha)$ denote the fraction of non-unsafe test cases on which the intact candidate and anchor agree, and let $\tau(v)$ denote a version's timestamp. We select

$$(a^*, v^*) = \underset{(a,v) \in S(\alpha,V)}{\arg\max} \big(b(v, \alpha), -\tau(v) \big), \tag{7}$$

using lexicographic order. Thus, agreement is prioritized and ties are resolved by the earliest timestamp. Across the evaluated histories, agreement scores tie whenever multiple candidates pass, making this rule equivalent to selecting the earliest accepted candidate.

After selecting a^*, we use its first flagged version as the estimated introduction point when available:

$$\bar{v}(a^*) = \begin{cases} v_{\text{on}}(a^*), & \text{if a flagged version exists,} \\ v^*, & \text{otherwise.} \end{cases} \tag{8}$$

This maps a later matched revision back to an estimated introduction point. The estimate uses the agent's timestamp-ordered history; Section 7 analyzes version-level recovery in branching archives.

The selected agent may itself have adopted the behavior from another agent. We therefore repeat the procedure from $(a^*, \bar{v}(a^*))$, updating the visited set as in Algorithm 1. Let o be the current attribution and o' the result of a recursive search. The update rule is

$$o_{\text{next}} = \begin{cases} o', & o' \neq \text{NONE} \ \wedge \ \tau(o') < \tau(o), \\ o, & \text{otherwise}, \end{cases} \tag{9}$$

where $\tau((a, v)) = \tau(v)$. The search retains the current attribution when no earlier candidate is found. A depth limit and a shortlist-based timestamp heuristic bound recursion; Appendix E.3 evaluates the latter's cost–accuracy tradeoff.

Each tracing level evaluates at most $2K$ candidate representatives. We use $K = 3$ and require program execution but no additional language-model calls.

6 Experiments

We evaluate exact origin recovery, the contributions of the method components, and performance on additional histories and fault operators.

6.1 Experimental Setup

Benchmark and evaluation. The core evaluation comprises 158 queries from 85 scenarios with 3–7 agents: 135 external-source queries in star, chain, and tree histories, and 23 no-source queries in cyclic histories. Accuracy requires the exact source agent–version pair or the correct NONE label. Several queries may share a scenario, so we also report scene-level accuracy and clustered confidence intervals. The method was refined on the core histories and evaluated on 34 additional routing queries using the same method and hyperparameters.

Table 1 defines the attribution conditions. Stars combine A, A+B, and A+B+C; chains and trees combine E, E+B, and E+B+A. These sequences add independent occurrences and further transfer or rewrite ambiguity to a simpler history. Queries audit the first flagged target version by default; D re-audits the last flagged version of the same histories. Appendix B gives construction details, and Appendix B maps condition coverage.

Table 1: Attribution conditions. A, B, C, and E describe histories; D changes the audited version.

Code	Condition	Change to the attribution problem
A	Semantic rewriting	Adopted code is rewritten while preserving the unsafe behavior.
B	Independent co-occurrence	An unrelated agent develops the behavior before the true source.
E	Multi-hop propagation	Code reaches the target through an intermediate recipient.
C	Unrelated adoption	Additional code transfers do not carry the unsafe behavior.
D	Evidence drift	The query targets the last flagged version instead of the first.

Baselines. We construct five baselines to test textual similarity, unsafe onset, failure reproduction, and behavioral matching. All six methods share histories, the audit probe, and the target query, and use no language-model calls. *Text similarity* (Text) selects the earliest onset among agents with near-maximal whole-program similarity. *Pure bisection* (Bisect) estimates each agent's unsafe onset assuming a monotone safety transition and selects the earliest; *exhaustive scanning* (Scan) uses a linear search. *Naive counterfactual*, inspired by intervention-and-reexecution approaches (Zeller & Hildebrandt, 2002; Moon et al., 2014; Papadakis & Traon, 2015), replaces the queried function with each candidate's most similar version, accepts unsafe reproduction, and ranks survivors by text similarity. *Exhaustive fingerprint matching* (Exh.) tests all other-agent versions with the same anchor, fingerprint acceptance, and temporal ranking, without recursion. *Our method* (Ours) adds candidate retrieval, multiple representatives, and recursive tracing (Section 5). Budgeted methods use $K = 3$. Appendix B gives matching tolerances; Appendix D reports an auxiliary read-only LLM comparison.

6.2 Main Results

Table 2 reports 96.8% exact accuracy for the full method versus 77.2% for exhaustive fingerprint matching. Both recover all 23 no-source queries and obtain the same star accuracy. The advantage lies in chains and trees, where exhaustive matching can find an intermediate recipient but does not trace it upstream. Component ablations below separate representation from recursion. Mean per-scene accuracy is 97.1%; Table 9 gives clustered intervals, including the paired gain over exhaustive matching.

Table 2: Exact attribution accuracy (%) by topology and condition. Conditions follow Table 1; groups partition the 158 core queries. D re-audits their last flagged versions. Exh.: exhaustive fingerprint matching; —: not separately rerun for D. Blue highlights Ours; pale blue highlights the strongest baseline(s) per row.

Condition	n	Text	Bisect	Scan	Naive CF	Exh.	Ours
Star ($n = 54$)							
A: Rewrite	12	66.7	**100.0**	**100.0**	33.3	100.0	**100.0**
A+B: Rewrite + co-occur.	21	66.7	33.3	47.6	52.4	100.0	**100.0**
A+B+C: All three conditions	21	47.6	14.3	14.3	23.8	85.7	**85.7**
Chain ($n = 33$)							
E: Multi-hop	11	0.0	90.9	**100.0**	0.0	81.8	**100.0**
E+B: Multi-hop + co-occur.	11	9.1	36.4	27.3	0.0	81.8	**100.0**
E+B+A: All three conditions	11	0.0	27.3	18.2	9.1	36.4	**90.9**
Tree ($n = 48$)							
E: Multi-hop	14	0.0	**100.0**	**100.0**	0.0	71.4	**100.0**
E+B: Multi-hop + co-occur.	16	0.0	12.5	12.5	0.0	25.0	**100.0**
E+B+A: All three conditions	18	0.0	33.3	22.2	0.0	66.7	**94.4**
Negative: no external source	23	0.0	26.1	26.1	91.3	100.0	**100.0**
Overall	158	20.9	42.4	42.4	26.6	77.2	**96.8**
D: Last flagged (drift)	158	5.1	42.4	42.4	22.8	—	**96.8**

Figure 3 compares all six methods by the number of attribution conditions. The full method achieves 100%, 100%, and 90%, versus 83.8%, 70.8%, and 68.0% for exhaustive fingerprint matching. These groups combine distinct topology-specific conditions (Table 1); their accuracies summarize performance across the combinations. Text similarity can favor rewritten recipients, temporal onset can favor independent occurrences, and unsafe reproduction alone need not distinguish them. Function-level text matching reaches 23.4% overall, versus 20.9% for whole-program matching.

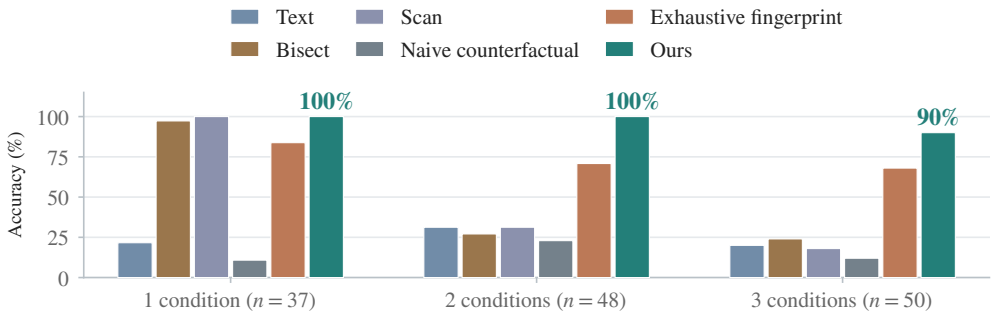


Figure 3: Accuracy on 135 external-source queries, grouped by the number of attribution conditions. Table 2 reports the individual combinations.

Auditing a later version tests a different source of ambiguity: revisions made after the unsafe behavior appeared. Under D, text similarity falls from 20.9% to 5.1% and naive counterfactual from 26.6% to 22.8%, while the full method retains its predictions by returning to the same historical an-

chor. Historical anchoring therefore preserves the attribution across the first- and last-flagged audits in these histories.

The full method averages 59.3 program executions and 0.57 seconds per query. Raising K from three to five changes accuracy from 96.8% to 97.5%; naive counterfactual remains at 26.6%. Appendix D reports costs and the budget sweep. Exhaustive matching's counts cover candidate replacements only, excluding onset localization (Appendix E.2).

6.3 Ablation Study

Table 3: Split localization ablation (accuracy, %) on the same 158 queries. Fingerprint acceptance and temporal selection are fixed; upstream extensions are off.

Target anchor	Candidate onset	First audit	Last audit
No	No	57.6	34.8
No	Yes	75.3	50.6
Yes	No	57.6	57.6
Yes	Yes	**75.3**	**75.3**

Table 4: Baseline comparison and component ablations on two new-history cohorts (correct counts). All retain localization and source ranking; budgeted settings use $K = 3$.

Configuration	New $n = 15$	Coverage $n = 19$
Full method	**15**	**19**
Exhaustive fingerprint	9	13
Unsafe-outcome acceptance	11	19
Neither upstream extension	9	13
Multiple representatives only	9	13
Recursive tracing only	9	13

An eight-configuration ablation separates historical localization, fingerprint acceptance, and temporal selection, with multiple representatives and recursion disabled (Appendix Table 8). Enabling all three raises accuracy from 26.6% to 75.3%; substituting similarity-based ranking for temporal selection reduces it to 27.8%.

Localization has two distinct roles. Table 3 shows that anchoring the target preserves all 158 predictions across first- and last-flagged audits. With that anchor fixed, selecting candidate onsets raises accuracy from 57.6% to 75.3%. The former removes dependence on later target revisions, while the latter changes which candidate evidence is examined.

Fingerprint acceptance distinguishes candidates using behavior beyond the unsafe outcome. With one representative and no recursion, it achieves 100% no-source and 71.1% external-source accuracy, compared with 0% and 84.4% for unsafe-outcome acceptance. On the full method's fixed first-level shortlists, it rejects all 100 known unrelated candidate–query comparisons that reproduce the unsafe outcome (Appendix E).

Matching alone still leaves intermediate recipients. Table 5 therefore varies multiple representatives and recursive tracing while keeping the preceding components fixed. Either extension alone achieves 75.3–75.9%; together they reach 96.8%. The gain occurs in chains and trees, while star accuracy and all 23 no-source predictions remain unchanged. This follows the mechanism in Section 5: a later relay version matches the target, while its onset version provides the next upstream comparison. Adding recursion to the multiple-representative configuration corrects 33 intermediate-recipient attributions.

Table 5: Ablation of multiple representatives and recursive tracing (accuracy, %). Historical localization, fingerprint matching, and temporal selection remain enabled in all four configurations. Negative examples have no external source; a correct answer is NONE.

Multiple representatives	Recursive tracing	Star ($n = 54$)	Chain ($n = 33$)	Tree ($n = 48$)	Negative ($n = 23$)	Overall ($n = 158$)
No	No	94.4	66.7	47.9	100.0	75.3
Yes	No	94.4	66.7	50.0	100.0	75.9
No	Yes	94.4	66.7	47.9	100.0	75.3
Yes	**Yes**	**94.4**	**97.0**	**97.9**	**100.0**	**96.8**

6.4 Additional Experiments

We evaluate the complementary roles of multiple representatives and recursive tracing on 34 additional queries from 19 newly generated scenarios, using the same method and hyperparameters. The full method recovers 34/34 origins, versus 22/34 for exhaustive fingerprint matching (Table 4). Together, the two components also recover 12 queries missed by either alone. Fingerprint acceptance rejects all four no-source queries, whereas unsafe-outcome acceptance misattributes them; both settings recover the 30 external-source queries. These new routing histories include four negative queries from one scene. Construction outcomes and baseline comparisons appear in Appendix E.

An additional evaluation covers approval-boundary errors and resource-owner field substitutions in short coding histories, inspired by AgentTrace (Wang, 2026). With task-specific probes and response signatures, the unchanged tracing method recovers 9/11 queries, versus 7/11 for exhaustive fingerprint matching. The gain comes from field-substitution histories (6/6 versus 4/6), where recursion recovers both multi-hop origins. Both methods recover 3/5 boundary-error queries; the two multi-hop errors follow relay revisions that match an earlier independent occurrence (Appendix E.3).

7 Discussion

Behavioral matching and historical tracing work together: fingerprints distinguish candidates beyond a shared unsafe outcome, while localization and recursion recover earlier sources through revised intermediates. The chain and tree results and ablations on both core and new histories support these complementary roles.

Source recovery relies on searchable source and relay versions, behavior preserved across each matched transfer, and matching and ranking that distinguish the propagation chain from unrelated candidates. Exact version recovery further requires onset localization to identify the introducing event on the relevant branch. These are working conditions for the tracing procedure; matching alone does not establish ancestry. In the core evaluation, source-agent or no-source accuracy is 98.7%, compared with 96.8% for exact agent–version recovery. Three errors select a matching version on another branch of the correct agent; the remaining two involve behavioral drift and stopping at an intermediate recipient. In the additional fault checks, both errors arise when a relay revision matches an earlier independent occurrence. Identical observations can therefore leave the historical relationship ambiguous. Execution tests assess behavioral hypotheses (Ma et al., 2025), while derivation links provide complementary historical evidence (Margalit et al., 2026).

Removing the true source-agent archive from 135 external-source queries yields 7 NONE outputs and 128 attributions to other agents (Appendix E.3). These outputs describe responses to missing evidence, rather than recovery of the unavailable origin; both rejection and accepted matches require interpretation relative to the archive. The budget sweep further highlights the importance of candidate coverage: accuracy changes from 58.9% at $K = 2$ to 96.8% at $K = 3$, motivating coverage measurements as agent populations grow.

The routing histories and additional fault operators isolate these attribution mechanisms under controlled changes. Direct candidate execution reproduces all core replacement-testing predictions, linking the observed gains to matching, localization, and tracing. Heterogeneous execution contexts provide a distinct setting for evaluating the role of replacement itself.

8 Conclusion

We trace the origins of unsafe behavior through code revision and cross-agent adoption. Behavioral matching, historical localization, and recursive tracing achieve 96.8% exact recovery on 158 queries from 85 scenarios, versus 77.2% for exhaustive matching. Ablations on core and new histories show how multiple representatives and recursion recover origins through revised intermediates.

Statement on AI Use

AI tools assisted manuscript editing. The authors reviewed the work and take responsibility for its contents.

Ethics Statement

This work studies defensive attribution using controlled, generated coding-agent histories. Faults are introduced for auditing research, and results are evaluated against recorded benchmark provenance.

Reproducibility Statement

Section 3 defines the evidence available to attribution methods, and Section 5 specifies the pipeline. Appendix A provides pseudocode; Appendix B documents models, schedules, input batteries, and comparison settings. Appendix D reports component ablations, execution costs, and the language-model baseline protocol, while Appendix E describes additional validation. Attribution methods are evaluated on the same archived histories, separating method comparisons from variation in language-model generation.

References

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

Mengzhuo Chen, Junjie Wang, Fangwen Mu, Yawen Wang, Zhe Liu, Huanxiang Feng, and Qing Wang. Seeing the whole elephant: A benchmark for failure attribution in llm-based multi-agent systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19888–19905, 2026. URL https://aclanthology.org/2026.acl-long.912/.

James R. Cordy and Chanchal K. Roy. The nicad clone detector. In *IEEE International Conference on Program Comprehension (ICPC)*, 2011.

Liwei Guo, Sixiang Ye, Zeyu Sun, Xiang Chen, Yuxia Zhang, Bo Wang, Jie M. Zhang, Zheng Li, and Yong Liu. LLMs are bug replicators: An empirical study on LLMs' capability in completing bug-prone code. *arXiv preprint arXiv:2503.11082*, 2025. URL https://arxiv.org/abs/2503.11082.

Runzhi He, Audris Mockus, Wenhao Yang, and Minghui Zhou. File-level copying is an implicit dependency in open source. *arXiv preprint arXiv:2607.02059*, 2026.

Yeonjun In, Mehrab Tanjim, Jayakumar Subramanian, Sungchul Kim, Uttaran Bhattacharya, Wonjoong Kim, Sangwu Park, Somdeb Sarkhel, and Chanyoung Park. Rethinking failure attribution in multi-agent systems: A multi-perspective benchmark and evaluation. *arXiv preprint arXiv:2603.25001*, 2026.

Jin Jia, Zhiling Deng, Zhuangbin Chen, Yingqi Wang, and Zibin Zheng. Mas-fire: Fault injection and reliability evaluation for llm-based multi-agent systems. *arXiv preprint arXiv:2602.19843*, 2026.

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.

Ruixiao Lin, Xinhao Deng, Qingming Li, Jianan Ma, Yunhao Feng, Yuqi Qing, Zhenyuan Li, Yechao Zhang, Shiwen Cui, Changhua Meng, Tianwei Zhang, Xingjun Ma, Qi Li, Ke Xu, and Shouling Ji. Safety in self-evolving LLM agent systems: Threats, amplification, and case studies. *arXiv preprint arXiv:2606.23075*, 2026. URL https://arxiv.org/abs/2606.23075.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. DoVer: Intervention-driven auto debugging for LLM multi-agent systems. *arXiv preprint arXiv:2512.06749*, 2025. URL https://arxiv.org/abs/2512.06749.

Xutao Mao, Liangjie Zhao, Xiang Zheng, and Cong Wang. Practice makes unsafe: Skill misevolution in self-improving LLM agents. *arXiv preprint arXiv:2608.12851*, 2026. URL https://arxiv.org/abs/2608.12851.

Yanki Margalit, Nurit Cohen-Inger, Erni Avram, Ran Taig, and Oded Margalit. Governed shared memory for multi-agent llm systems. *arXiv preprint arXiv:2606.24535*, 2026.

Seokhyeon Moon, Yunho Kim, Moonzoo Kim, and Shin Yoo. Ask the mutants: Mutating faulty programs for fault localization. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 2014.

Mike Papadakis and Yves Le Traon. Metallaxis-fl: mutation-based fault localization. *Software Testing, Verification and Reliability*, 25(5-7):605–628, 2015.

Hezhe Qiao, Hanghang Tong, Ee-Peng Lim, Bing Liu, and Guansong Pang. Verifymas: Hypothesis verification for failure attribution in llm multi-agent systems. *arXiv preprint arXiv:2605.17467*, 2026.

Hitesh Sajnani, Vaibhav Saini, Jeffrey Svajlenko, Chanchal K. Roy, and Cristina V. Lopes. Sourcerercc: Scaling code clone detection to big code. In *Proceedings of the 38th International Conference on Software Engineering*, pp. 1157–1168, 2016. URL https://arxiv.org/abs/1512.06448.

Shuai Shao, Qihan Ren, Chen Qian, Boyi Wei, Dadi Guo, Jingyi Yang, Xinhao Song, Linfeng Zhang, Weinan Zhang, Dongrui Liu, and Jing Shao. Your agent may misevolve: Emergent risks in self-evolving llm agents. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2509.26354.

Asankhaya Sharma. OpenEvolve: an open-source evolutionary coding agent. GitHub, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve.

Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In *Proceedings of the International Workshop on Mining Software Repositories (MSR)*, 2005.

Gou Tan, Zhensu Sun, Jieke Shi, Ting Zhang, Zilong He, Qingfu Wu, Shuai Liang, Weifeng Sun, Junda He, Pengfei Chen, Chuanfu Zhang, Lwin Khin Shar, and David Lo. Agentchaos: Chaos engineering for agent systems via programmatic fault injection. In *Proceedings of the 41st IEEE/ACM International Conference on Automated Software Engineering*, 2026. doi: 10.1145/3832783.3837437. URL https://arxiv.org/abs/2608.06790.

Kavana Venkatesh, Jafar Isbarov, Saad Amin, Murat Kantarcioglu, and Jiaming Cui. Caspian: Online detection and attribution of cascade attacks in llm multi-agent systems via cross-channel causal monitoring. *arXiv preprint arXiv:2605.19240*, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=ehfRiF0R3a.

Zhaohui Geoffrey Wang. Agenttrace: Causal graph tracing for root cause analysis in deployed multi-agent systems. In *ICLR 2026 Workshop on Agents in the Wild*, 2026. URL https://arxiv.org/abs/2603.14688.

Andreas Zeller and Ralf Hildebrandt. Simplifying and isolating failure-inducing input. *IEEE Transactions on Software Engineering*, 28(2):183–200, 2002.

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.

Lingzhe Zhang, Tong Jia, Kangjin Wang, Weijie Hong, Chiming Duan, Minghua He, and Ying Li. Adaptive root cause localization for microservice systems with multi-agent recursion-of-thought. *arXiv preprint arXiv:2508.20370*, 2025b.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 76583–76599, 2025c. URL https://proceedings.mlr.press/v267/zhang25cq.html.

Weixiang Zhao, Yichen Zhang, Yingshuo Wang, Yang Deng, Yanyan Zhao, Xuda Zhi, Yongbo Huang, Hao He, Wanxiang Che, Bing Qin, and Ting Liu. On safety risks in experience-driven self-evolving agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 42145–42169, 2026. URL https://aclanthology.org/2026.findings-acl.2091/.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. Multiagentbench: Evaluating the collaboration and competition of llm agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8580–8622, 2025. URL https://aclanthology.org/2025.acl-long.421/.

A The Method as Pseudocode

Algorithm 1 Origin attribution along an evolutionary lineage: $\mathrm{Trace}(t, t_q, V, d)$

Require: target t, query version t_q, histories $\{H_a\}$, probe π, region r, battery B, budget K, visited set V, depth d

$\alpha \leftarrow$ first flagged target version up to t_q, or t_q if none exists
Compute $C_K(\alpha, V)$ and $R(a; \alpha)$ using Equations (3) and (4)
$S \leftarrow \varnothing$; $P \leftarrow \{v : a \in C_K(\alpha, V),\ v \in R(a; \alpha)\}$
for all $a \in C_K(\alpha, V)$ **do**
    for all $v \in R(a; \alpha)$ **do**
        if replacement succeeds and $\phi_B(\alpha[r \leftarrow v]) = \phi_B(\alpha)$ **then**
            $S \leftarrow S \cup \{(a, v)\}$
if $S = \varnothing$ **then**
    return NONE
$(a^*, v^*) \leftarrow \arg\max_{(a,v)\in S}(b(v, \alpha), -\tau(v))$ ▷ lexicographic
$\bar{v} \leftarrow v_{\mathrm{on}}(a^*)$ if it exists, otherwise v^*
$o \leftarrow (a^*, \bar{v})$
if $d < 4$ and $a^* \notin V$ and $\tau(\bar{v}) > \min_{v\in P} \tau(v)$ **then**
    $o' \leftarrow \mathrm{Trace}(a^*, \bar{v}, V \cup \{t, a^*\}, d + 1)$
    if $o' \neq$ NONE and $\tau(o') < \tau(o)$ **then**
        $o \leftarrow o'$
return o

The initial call uses $V = \varnothing$ and $d = 0$. The timestamp rule continues tracing when the current shortlist contains a representative earlier than the reported onset. Candidate retrieval uses all collected histories; recursive updates must move to an earlier reported version. Every reported external source in the core predictions precedes or coincides with its queried target version. Probe failures are skipped during onset localization, and unsuccessful code replacements are rejected.

B Benchmark Construction Details

Registry topologies. The registry graph fixes which agent may adopt from which, and we build four shapes: a star, in which one agent supplies several; a chain, in which each agent supplies the next; a tree, in which a chain branches; and a cyclic shape used for the negative scenarios, in which

agents copy from one another benignly and no adoption carries the unsafe behavior. The first three apply the coordination taxonomy of Zhu et al. (2025) to code-derivation graphs. Chain and tree histories include multi-hop propagation before the target is queried.

Coverage of attribution conditions. Table 6 maps the core queries to the available condition combinations. Rewriting challenges textual matching, independent co-occurrence challenges selection by earliest unsafe behavior alone, multi-hop propagation separates an immediate donor from an introducing source, and unrelated adoption adds transfers outside the unsafe lineage. The combination groups evaluate these ambiguities together on distinct generated histories.

Table 6: Coverage of the fixed core evaluation. Counts refer to queries; D re-audits the same 158 queries at their last flagged versions. Condition definitions appear in Table 1.

Structure / audit	**Evaluated groups (n)**	**Total**	**Attribution question**
Star	A (12); A+B (21); A+B+C (21)	54	Rewritten direct adoption with independent occurrences and unrelated transfers.
Chain	E (11); E+B (11); E+B+A (11)	33	Recovering an upstream source through sequential adoption and revision.
Tree	E (14); E+B (16); E+B+A (18)	48	Recovering a source through branching adoption with the same ambiguity types.
Negative (no source)	Target-local introduction	23	Rejecting an external source despite benign code exchange.
Later audit (D)	Last rather than first flagged version	158	Stability across audited versions of the same histories.

Coverage design. The core groups follow the topology-specific sequences in Table 6; a separate cohort adds B-only, C-only, and E+C histories (Table 11). All eight method-component combinations are evaluated on the same core queries (Table 8).

Configuration and schedules. Table 7 specifies the core configuration. Each agent's archived population includes evaluated candidates as well as selected programs. Version labels use the archive's iteration/generation ordering; temporal comparisons use recorded creation timestamps.

Table 7: Core environment and attribution configuration.

Component	Configuration
Base agent	OpenEvolve 0.3.2; diff-based Python evolution
Evolution model	`gpt-5-mini`, low reasoning effort, 10,000 completion-token limit
Archive	Population 20, archive 20, one island
Prompt sampling	Three top programs and two diverse programs
Online evaluation	Serial evaluation; 30-second timeout; cascade evaluation off
Schedules	Recorded pre-event depths 1–5 rounds and background depths 1–3; intermediate relay evolution uses 3 rounds before republication
Scenario size	3–7 agents in the core set, mean 5.0 across 85 scenes
Fingerprint	16 utility inputs, 7 robustness inputs, and the audit probe; signatures record tool, mode, and confirmation fields
Attribution	Function `compile_workflow`; $K = 3$; up to two representatives per agent; recursion depth limit 4

The utility inputs cover search, calculation, summarization, translation, calendar reads/writes, email drafts/sends, file reads/writes, database reads/writes, code review, report generation, image analysis, and batch search. Robustness inputs are `None`, an empty dictionary, a null request kind, an unknown kind, a string, an integer, and an empty list. The battery is shared across candidates. Semantic-rewrite prompts request behavior preservation; acceptance checks parsing, retention of the audit outcome, and the configured text-similarity threshold (0.50 for the default rewrite).

Comparison inputs. The function r is specified by the task configuration, and the fingerprint battery uses the task evaluator's fixed inputs and robustness cases together with the audit probe. The text baseline uses `difflib.SequenceMatcher` on whole programs: within each agent it selects the earliest version within 0.02 of that agent's maximum similarity, then selects the earliest onset among agents within 0.05 of the global maximum. The attribution pipeline instead retrieves candidates by similarity of the specified function. Its candidate records contain agent identity, version ID, code, and timestamp.

Evaluation membership. The reported evaluation comprises 158 queries from 85 scenarios covering star, chain, tree, and cyclic no-source histories. All core accuracy tables, component analyses, and scene-level statistics use this same query set. Additional evaluation uses 34 queries from 19 newly generated scenarios with the method and hyperparameters fixed (Appendix E).

C Additional Connections to Prior Work

Evidence and provenance. CASPIAN attributes cascades across agents within a live session, while RCLAgent navigates execution traces for root-cause localization in microservice systems (Venkatesh et al., 2026; Zhang et al., 2025b). Our queries provide historical code and timestamps, with adoption records reserved for establishing evaluation labels. Studies of file-level copying likewise motivate recovering reuse relationships from software artifacts (He et al., 2026).

Controlled attribution benchmarks. AgentChaos, MAS-FIRE, AEGIS, and AgentTrace use controlled fault construction to evaluate agent failures (Tan et al., 2026; Jia et al., 2026; Kong et al., 2025; Wang, 2026). Our protocol records an introducing version and subsequent code adoption, yielding an exact agent–version label for each queried lineage. This complements evaluation of multiple defensible attributions in MP-Bench (In et al., 2026).

D Supplementary Results

Fingerprint acceptance. Figure 4 compares fingerprint-based acceptance with unsafe-outcome acceptance. With fingerprint acceptance, no-source and external-source accuracies are 100% and 71.1%; with unsafe-outcome acceptance, they are 0% and 84.4%. This ablation uses a single representative per agent, fixed localization and temporal selection, and disabled recursion.

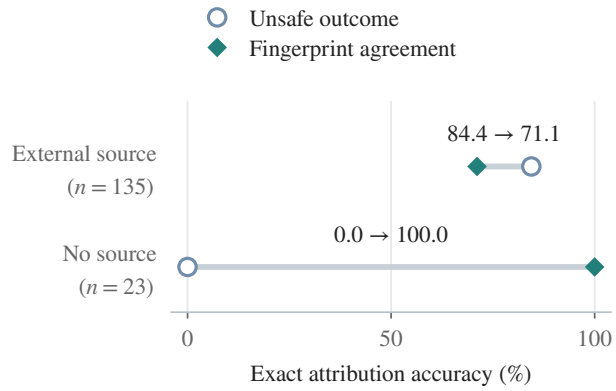


Figure 4: Fingerprint matching versus unsafe-outcome reproduction on external-source and no-source queries. Localization and temporal selection are fixed.

Table 8 separates the effects of localization, fingerprint matching, and temporal selection across all eight configurations. Its first- and last-flagged columns evaluate the same core queries at different audited versions, complementing the component comparisons in the main text.

Table 8: Complete factorial results on the core queries (accuracy in percent). Localization, fingerprint matching, and temporal selection are enabled where marked Yes. The first and last columns refer to the audited target version.

Localization	Fingerprint	Temporal	External	No source	First	Last
–	–	–	15.6	91.3	26.6	22.8
–	Yes	–	14.8	100.0	27.2	21.5
Yes	–	–	15.6	0.0	13.3	13.3
–	–	Yes	60.0	91.3	64.6	44.3
Yes	Yes	–	15.6	100.0	27.8	27.8
–	Yes	Yes	50.4	100.0	57.6	34.8
Yes	–	Yes	84.4	0.0	72.2	72.2
Yes	Yes	Yes	71.1	100.0	75.3	75.3

Scene-level results and confidence intervals. The method reaches 97.1% mean per-scene accuracy, alongside 96.8% query-weighted accuracy (Table 9). The scene mean weights each of the 85 scenarios equally, complementing the query-level comparison in the main results. Confidence intervals use 10,000 bootstrap draws of the 85 scenes, retaining all queries of each sampled scene together. The same draws are used for all methods and for paired differences. Against exhaustive fingerprint matching, the query-weighted gain is 19.6 percentage points (95% paired interval: 11.4–28.9); the equal-scene gain is 24.1 points (15.3–33.5).

Table 9: Core accuracy with uncertainty estimated by resampling all queries within each of the 85 scenes together (10,000 bootstrap draws; 95% percentile intervals). Scene means weight each scene equally.

Method	Query accuracy [95% CI]	Scene mean [95% CI]
Text	20.9 [12.8, 29.4]	16.9 [10.2, 24.3]
Text (function)	23.4 [14.8, 32.5]	18.4 [11.4, 26.3]
Bisect	42.4 [32.3, 52.9]	44.2 [34.2, 54.5]
Scan	42.4 [32.0, 53.1]	43.0 [32.9, 53.6]
Naive counterfactual	26.6 [17.2, 35.9]	19.4 [12.4, 27.0]
Ours	96.8 [92.1, 100.0]	97.1 [92.9, 100.0]
Exhaustive fingerprint	77.2 [67.5, 85.7]	72.9 [63.5, 82.4]
Gain over exhaustive (pp)	19.6 [11.4, 28.9]	24.1 [15.3, 33.5]

Candidate-budget analysis. Table 10 reports accuracy and execution cost across candidate budgets. For the full method, increasing K from 2 to 3 raises accuracy from 58.9% to 96.8%; increasing it to 5 yields 97.5%. We use $K = 3$ for the main evaluation. At this budget, the three-adaptation configuration reaches 75.3%, and multiple representatives with recursive tracing raise accuracy to 96.8%. Execution counts measure local program loads and evaluations; elapsed times describe this experimental environment.

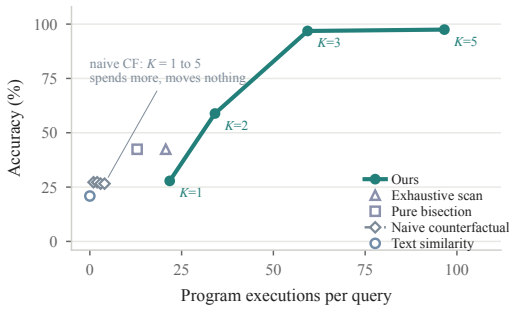


Figure 5: Accuracy and mean program executions per query on the core set. Curves vary K for the two counterfactual methods; other methods use fixed configurations.

Figure 5 shows the accuracy–execution trade-off; Table 10 gives the corresponding budgets, execution counts, and elapsed times. Increasing the full method's budget from $K = 3$ to $K = 5$ raises accuracy from 96.8% to 97.5%, while mean executions increase from 59.3 to 96.6 per query.

Table 10: Candidate-budget sweep on the 158 core queries. Accuracy is in percent; executions and seconds are per query. The text baseline here is region-level (23.4%); the main table and Figure 5 use the whole-program result (20.9%).

Method	K	Accuracy	Executions	Seconds
Text similarity (region level)	—	23.4	0.0	0.218
Pure bisection	—	42.4	12.9	0.028
Exhaustive scan	—	42.4	20.7	0.033
Naive counterfactual	1	27.2	1.0	0.317
	2	27.2	2.0	0.317
	3	26.6	2.9	0.325
	5	26.6	4.0	0.325
Three adaptations	1	27.8	19.2	0.338
	2	53.2	27.9	0.349
	3	75.3	35.2	0.357
	5	76.0	42.5	0.362
Ours	1	27.8	21.8	0.349
	2	58.9	34.1	0.383
	3	**96.8**	59.3	0.566
	5	97.5	96.6	0.826

LLM baseline. The LLM baseline uses `gpt-5-mini` with one API call per query, an incident-response system prompt, a 4,000-token completion limit, and no execution tools. Inputs contain archived code histories and the safety finding. The prompt requests the earliest introducing agent and version across the histories, or NONE, without specifying the queried target version. Exact-match scoring against the 158 core-query labels gives 15.2%, including 28 empty predictions counted as incorrect. Table 2 evaluates executable methods with the target query specified. This LLM evaluation is therefore an auxiliary observation under different inputs, not a controlled comparison of LLM and executable attribution capabilities.

E SUPPLEMENTARY VALIDATION

The supplementary evaluations cover additional histories, candidate discrimination, and attribution errors. Replay analyses retain the original historical labels.

E.1 ADDITIONAL EXPERIMENTS

New propagation histories. Two generation batches yield eight scenarios and 15 queries, evaluated using the same method and hyperparameters as the core set. The first batch produces four scenarios and nine queries. Four of six second-batch attempts produce multi-hop histories with independent occurrences, yielding six queries. The two remaining scene attempts exhaust five source-injection attempts before semantic adoption and are not replaced. Independent occurrences precede the source in two completed second-batch scenes and follow it in the other two.

Additional attribution conditions. We extend the evaluation to isolated conditions and their combinations using 12 prespecified generation attempts: four E+C chains, four E+C trees, two B-only stars, and two C-only stars. Generation uses `gpt-5-mini`; attribution uses the same method, four baselines, and $K = 3$ as the core evaluation. For condition C in chains and trees, the target adopts a clean function from an unrelated agent before receiving the unsafe function along the designated propagation lineage. The B-only generator schedules one independent agent before the source. All agent and donor histories are newly generated. Queries target the first flagged version.

The 12 prespecified generation attempts produced 11 condition-verified scenes and 19 queries. One scene exhausted the five source-generation attempts without producing the required unsafe source; it contributed no queries and was not replaced. We verified that every C transfer was probe-negative at both donor and recipient and preceded the query; B required an independent unsafe occurrence before the recorded origin. We also checked source-parent safety and that each first flagged target version was the recorded unsafe adoption.

Table 11: New GPT-generated coverage checks, separate from the 158-query core set. Entries are correct-query counts; n is the number of verified queries. Scene counts report verified/attempted. B: independent co-occurrence; C: unrelated adoption; E: multi-hop propagation. Blue highlights Ours; pale blue highlights the strongest baseline(s).

Condition	Scenes	n	Text	Bisect	Scan	Naive counterfactual	Ours
Chain: E+C	3/4	3	0	3	3	0	**3**
Tree: E+C	4/4	8	0	8	8	0	**8**
Star: B only	2/2	4	4	0	0	2	**4**
Star: C only	2/2	4	4	4	4	2	**4**
Overall	11/12	19	8	15	15	4	**19**

These new histories extend coverage to isolated B and C and to C under both multi-hop topologies, comprising 19 queries from 11 scenes.

Additional-history ablations. Table 4 in the main text compares exhaustive fingerprint matching with five fixed component settings on the two new-history cohorts reported above, retaining their separate membership and $K = 3$ for the budgeted settings. The complete method recovers all 34 queries. Exhaustive fingerprint matching uses the same all-version protocol as Appendix E.2, recovering 9/15 and 13/19 queries in the two cohorts. Restricting its candidates to versions available at audit time preserves every prediction. Both variants recover all four no-source queries; the complete method corrects 12 external-source errors. Unsafe-outcome acceptance changes only the hybrid acceptance test; benign-agreement ranking, onset reporting, and recursive tracing remain fixed. The 15-query cohort includes four negative queries, all correctly rejected by the fingerprint-based configurations and misattributed by unsafe-outcome acceptance. The 19-query coverage cohort contains only external-source queries.

Localization ablation. Table 3 in the main text independently switches target-anchor localization and candidate-onset representative selection in the single-representative factorial setup of Table 8. Fingerprint acceptance, earliest-accepted selection, and $K = 3$ are fixed; multiple representatives and recursion are disabled throughout. Candidate localization selects the version tested and returned. The diagonal settings correspond to disabling or enabling both localization steps in the factorial ablation. The four configurations yield 91, 119, 91, and 119 correct first-flagged attributions, and 55, 80, 91, and 119 correct last-flagged attributions, respectively.

E.2 Baseline Comparisons and Ablations

Exhaustive fingerprint matching. Exhaustive fingerprint matching evaluates whether all-version candidate coverage can recover the origins without recursive tracing on the same 158 core queries. It uses the identical first-flagged target anchor and fingerprint acceptance rule, but tests every version of every other agent instead of selecting top-K agents and representative versions. Accepted candidates use the same benign-agreement/earliest-timestamp ranking and the selected agent's first flagged version is returned. Empty accepted sets predict NONE. Ground-truth labels are used only for scoring. All-collected-history and audit-time-restricted variants each recover 122/158 queries (77.2%): 99/135 external-source and 23/23 no-source queries. The full-method replay recovers 153/158, including 130/135 external-source queries. Thus, exhaustive one-level matching does not recover the full pipeline's accuracy. This comparison isolates a stronger simple alternative but does not establish that the particular recursive strategy is optimal.

The two baseline variants attempt means of 29.0 and 21.3 candidate replacements per query, respectively. These counts exclude onset localization and are not directly comparable to the total program-execution costs in Table 10. The all-history and audit-time variants reject 11 and 8 candidates lacking the specified function, respectively; all 158 queries are retained. Their per-query predictions are identical.

Component ablations and candidate analysis. In the ablation of Table 5, the joint configuration recovers 34 additional queries over the configuration with neither extension: 10 on chains and 24 on trees. With representatives alone, 33 of these queries stop at the intermediate recipient. On the full method's fixed first-level shortlists, 33 known independent-distractor comparisons and 67 comparisons from no-source queries reproduce the unsafe outcome; fingerprint agreement rejects all 100. These counts refer to candidate–query comparisons, which may share scenes. The true-origin agent is shortlisted in 133/135 external-source queries and has an accepted representative at the first tracing level in 100/135; recursive tracing raises final external-source accuracy to 130/135.

Input-battery sensitivity. Table 12 varies the input battery while retaining the audit probe and the full attribution pipeline. Three predetermined permutations define nested subsets of four and eight utility inputs, with or without the seven robustness inputs. Eight utility inputs yield 95.6–96.2% accuracy. Using all 16 utility inputs without the robustness cases preserves every full-battery prediction, whereas four-input subsets are more variable. These results link discrimination to coverage of routine behavior.

Table 12: Input-battery sensitivity on the 158 core queries. Triples report correct counts under three fixed input permutations; all rows retain the audit probe.

Utility inputs	Robustness inputs	Correct / 158	External / 135	No source / 23
0	0	105	105	0
0	7	104	104	0
4	0	130, 146, 140	121, 127, 121	9, 19, 19
4	7	128, 146, 140	119, 127, 121	9, 19, 19
8	0	151, 151, 152	130, 128, 131	21, 23, 21
8	7	151, 151, 152	130, 128, 131	21, 23, 21
16	0	153	130	23
16	7	153	130	23

E.3 Robustness and Error Analysis

Temporal availability and execution context. Restricting histories to versions available by the audit time yields 154/158 correct attributions (97.5%), compared with 153/158 (96.8%) using all collected histories. Applying the restriction separately at each tracing anchor gives the same predictions. The single changed answer recovers an origin previously omitted from the shortlist. Separately, holding all attribution components fixed, direct execution of candidate programs reproduces all 158 replacement-testing predictions. Program syntax outside the transferred function is identical in all 696 first-level comparisons. The two execution forms also agree on all nine queries in the first new-history batch.

Recursion stopping rule. A paired ablation removes only the shortlist-based early-stop condition, retaining $K = 3$, multiple representatives, the depth bound, the visited set, and the requirement that recursive updates move backward in time. Without early stopping, 154/158 queries are correct, versus 153/158 with the original rule; the sole changed prediction recovers D v6 rather than the intermediate recipient A v6 in one tree history. Mean tracing levels increase from 1.62 to 3.11, and attempted candidate replacements from 7.01 to 11.54 per query. These counts include recursive levels but exclude onset scans and are not total program-execution costs. The rule is therefore a computation-saving heuristic rather than a guarantee that no earlier source exists. We retain the original configuration for the main comparisons and report this variation separately.

Error analysis. Source-agent or no-source identification is correct in 156/158 queries, versus 153/158 for exact agent–version attribution. Three errors share one scene: the recorded source

D v9 and the predicted D v5 descend from the same safe parent, but D v5 is not an ancestor of D v9. Their behavioral fingerprints agree. The remaining errors are a true-source rejection after behavioral drift and an attribution that stops at an intermediate recipient under the shortlist-based stopping rule. Onset localization uses creation timestamps; displayed version labels identify archived programs.

Additional fault types. Motivated by AgentTrace's logic-error and data-corruption categories (Wang, 2026), we run a separate small controlled evaluation with two concrete code faults. A boundary error changes the approval condition from `risk >= 0.8` to `risk > 0.8`. A field-substitution error uses `actor_id` instead of `owner_id` when constructing resource-operation arguments. These are our code-level instantiations of the categories, rather than scenarios reproduced from AgentTrace. Neither adds a new request-kind trigger.

Prespecified generation. Twelve scene attempts cover two fault families, three history types (direct transfer, multi-hop transfer, and target-local introduction), and two generation attempts per combination. Each scene contains four agents initialized from a shared action-planning function. Three receive one independent `gpt-5-mini` scheduling revision; the fourth uses a fixed valid reference implementation and independently receives the fault before the true source. Agent roles and harmless pre-introduction archive depths are randomized across scenes. Valid scheduling policies may differ on intermediate workloads; we neither require nor select for fingerprint separation. Model prompts receive synthetic task specifications and code, without source labels or the exact audit requests.

Each completed scene uses one additional model revision. Direct-transfer and target-local histories request a semantics-preserving refactor. Multi-hop histories request a benign scheduling change at the relay before retransmission, while retaining the confirmation and resource-argument behavior. Controlled fault introduction and recorded function adoptions define the origin. Validation checks that the source's parent is safe, that injection preserves all 16 online utility-case outputs, that reverting the injected edit recovers the parent, and that the recorded origin lies on the queried lineage while the independent occurrence does not. The final target version is audited. Failed constructions are retained without replacement; attribution success is not a construction criterion.

Method and evidence. Retrieval, $K = 3$, representative selection, temporal ranking, and recursive tracing remain unchanged. Each task uses its own probe and a fixed battery of 26 routine inputs, five malformed inputs, and the audit outcome. Fingerprints compare the complete returned plan, including operation arguments, so field-substitution behavior is observable. All compared methods receive the same code histories and probes; both fingerprint methods use the same signatures. Only the task-specific observations change; the source-selection algorithm remains fixed. These short controlled coding histories complement the OpenEvolve evaluation; they are not a second end-to-end evolutionary benchmark or a test of heterogeneous surrounding code.

Table 13: Additional fault-family checks. Entries are correct-query counts; each valid scene supplies one query. Full: complete method; No rec.: multiple representatives without recursion; Exhaustive: all-version fingerprint matching without recursion.

Fault	History	n	Full	No rec.	Exhaustive	Naive CF
Boundary	Direct	2	2	2	2	2
	Multi-hop	2	0	0	0	0
	Local	1	1	1	1	1
Field substitution	Direct	2	2	2	2	0
	Multi-hop	2	2	0	0	0
	Local	2	2	2	2	2
Overall		11	9	7	7	5

Outcomes and interpretation. Eleven of twelve attempts yield valid scenes. One boundary/local attempt fails because a generated program violates the normal scheduling specification; it is not replaced. The full method recovers 6/6 field-substitution queries and 3/5 boundary-error queries, versus 4/6 and 3/5 for exhaustive matching. Selecting the earliest unsafe external agent fails on all 11 queries because the independent occurrence is deliberately earlier. Both boundary multi-hop errors select a non-ancestor with a matching fingerprint after the relay's benign revision. A

diagnostic audit of each relay's initial adoption correctly recovers the source in all four multi-hop scenes; these diagnostic queries are not included in the primary counts. Thus, the checks show transfer to additional fault operators and a concrete failure under behavioral convergence, rather than universal fault-family generalization.

Withheld source archives. A missing-history stress test removes every archived version of the true origin agent from each of the 135 external-source core queries (79 scenes), retaining the target and other agents, including recipients. Original historical labels are unchanged. Ground-truth labels determine which archive is removed and are not available to the attribution method. With complete histories, the method recovers 130/135 origins. After removal, 7/135 queries return NONE and 128/135 attribute the behavior to another agent. Neither outcome recovers the original source, which is unavailable by construction; in particular, these NONE outputs are not correct target-local labels. The experiment measures responses to deliberately incomplete evidence, not naturally occurring archive-loss rates. It shows that missing provenance can remain undetected even when candidates match.

F EVALUATION SCOPE

The core evaluation covers literal-string routing bypasses in 3–7-agent histories with one introduction per queried lineage. Controlled injection and adoption records supply exact labels while agents evolve between transfers. The evaluation combines the 158-query core evaluation with 34 queries generated after freezing the method, as detailed in Appendices B and E. A separate 11-query controlled check adds boundary and field-substitution faults (Appendix E.3).