TRACING AND REFINING INJECTION ATTACKS IN AGENT MEMORY VIA LIFECYCLE RED TEAMING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model agents increasingly suffer from untrusted data that can contaminate memory and pose major security risks. While recent red teaming research explores various memory injection attacks, current evaluations primarily measure final success rates. They overlook the lifecycle of injections and fail to capture the causal chain of attacks, making it impossible to identify the precise stage where an injection fails. Furthermore, current studies often overlook that injected memories might require interactions with each other to take effect. Tracing these hidden dynamics is difficult in standard systems but becomes exceptionally severe in emerging graph memory environments, where injected inputs undergo complex merging and structural evolution. To address these challenges across diverse platforms, we propose a Lifecycle Red Teaming framework to reconstruct the lifecycle path and refine injected memory attacks. Specifically, we first propose LifecycleTracer to reconstruct fragmented lifecycle paths and expose injection interactions. As the foundation, we establish a formal memory lifecycle chain and a validity gating mechanism to standardize evaluations across diverse memory platforms. To pinpoint precise failure stages for individual injections, we incorporate Intra-Lifecycle Replay Attribution to bridge causal gaps within native logs. To uncover complex memory interactions, we also design Cross-Lifecycle Dynamic Attribution to capture interdependencies among multiple injected inputs. Finally, we leverage this feedback to refine injection attacks. Extensive experiments confirm that our framework effectively traces the lifecycle chain and enhances attack performance. Our code is available at `https://anonymous.4open.science/r/LifecycleRed-ICLR`.

1 INTRODUCTION

Large language model agents use memory to retain information across conversations and improve decisions (Zhong et al., 2024; Xu et al., 2025). Past experiences help them act consistently and accumulate knowledge beyond a single interaction (Park et al., 2023; Shinn et al., 2023). Yet memory may absorb untrusted external content that redirects an agent away from the user's intent (Liu et al., 2024c; Zhan et al., 2024). Once stored, an injection can influence later legitimate queries, extending its effect beyond the initial encounter.

Existing attacks expose several ways to compromise memory. AgentPoison optimizes malicious memories for retrieval and later behavior (Chen et al., 2024), while MINJA shows how ordinary queries can induce malicious records (Dong et al., 2025). Other studies investigate barriers to retrieval (Chang et al., 2026) and content distributed across interactions (Rao et al., 2026). Despite exposing vulnerabilities across agent tasks (Ye et al., 2024; Debenedetti et al., 2024; Zhang et al., 2025), existing evaluations often center on final Attack Success Rate (ASR). Yet this outcome alone cannot reveal an injection's path through memory, the stage where its effect changes, or its dependence on other injections.

These limits arise because an injection's lifecycle can be altered by later updates or other injections, where it can form a memory item, change during later updates, appear in retrieval and the model input, and affect the decision. Evidence at one stage does not establish its progress through the next: a stored item may disappear after an update, and a retrieved passage may no longer reveal which input produced it. This distinction matters for graph memories, which can merge entities and revise

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.

relations (Gutiérrez et al., 2024; 2025). Figures 1a and 1b illustrate the difference between writing an injection and its later retrieval, inclusion in context, and appearance in the response. They also show how normal updates can change its availability.

Multiple injections introduce a second question: how do they affect one another? In Figure 1c, removing a part of the tested three- and four-input chains eliminates exact composition, while the five-input chain fails even with every part present. Figure 1d shows that adding an input can raise or lower the target answer rate. These observations call for tracing each input's path and testing effects between inputs under matched conditions. Therefore, we organize this investigation around two core research questions: *Where precisely does an individual injection lose its effect, and under what conditions does that effect depend on interactions with other memory objects?*

To answer these questions, we propose Lifecycle Red Teaming, a framework for diverse memory systems that integrates LifecycleTracer for tracing with LifecycleRefiner for attack refinement. LifecycleTracer captures how injected inputs propagate through memory processing and attributes their effects to specific stages and interactions. It formalizes a memory lifecycle chain with validity criteria to establish a consistent basis for interpreting evidence across vector and graph memory systems. Building on this foundation, Intra-Lifecycle Replay Attribution uses controlled replay to address evidence gaps and locate where an individual injection takes effect or is blocked, while Cross-Lifecycle Dynamic Attribution is designed to examine how adding or removing an injection changes another memory's behavior under fixed conditions. LifecycleRefiner evaluates a fixed set of attack revisions through matched branches and selects among them using component feedback. We compare this selection rule with applying both revision components in every case. Experiments show high agreement with operational lifecycle references, localization of controlled stage changes, and identification of conditional interactions. Fixed revisions substantially improve attack success, and matched branches quantify the selector's gain over fixed joint.

Our contributions are as follows: (i) We propose Lifecycle Red Teaming, a framework that integrates attack tracing and refinement across memory systems. (ii) We develop LifecycleTracer, grounded in a formal memory lifecycle chain and validity gating, with Intra-Lifecycle Replay Attribution and Cross-Lifecycle Dynamic Attribution to reconstruct injection paths, localize failures, and attribute interactions among memories. (iii) We evaluate fixed attack revisions and a component-based selection rule under matched branch costs. (iv) Experiments quantify lifecycle reconstruction, controlled failure localization, interaction attribution, and attack revision under matched candidate evaluations.

2 Related Work

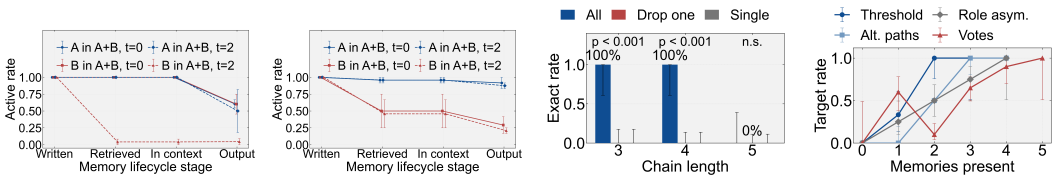
2.1 Memory for Language Agents

Agent memory has evolved from retaining observations for later action (Park et al., 2023; Shinn et al., 2023) to consolidating and updating records (Zhong et al., 2024; Xu et al., 2025). Its representations range from RAPTOR's summary trees (Sarthi et al., 2024) to graph retrieval in HippoRAG and G-Retriever (Gutiérrez et al., 2024; 2025; He et al., 2024); Graphiti and Mem0 provide further agent memory architectures (Rasmussen et al., 2025; Chhikara et al., 2025). Self-RAG controls retrieval during generation (Asai et al., 2024). Benchmarks span general agent capabilities in AgentBench and long-term conversational memory in LoCoMo and LongMemEval (Liu et al., 2024a; Maharana et al., 2024; Wu et al., 2025). These benchmarks measure memory utility, not an injected record's path through writing, retrieval, context inclusion, and decisions.

2.2 Injection Attacks and Defenses

Prompt-injection benchmarks expose attacks through untrusted content (Zhan et al., 2024; Debenedetti et al., 2024; Zhang et al., 2025), while PoisonedRAG corrupts retrieved knowledge (Zou et al., 2025). Memory poisoning persists across turns: AgentPoison targets retrieval, MINJA induces malicious writes through queries, and Sleeper plants false memories for later use (Chen et al., 2024; Dong et al., 2025; Pulipaka et al., 2026). Other work studies payload access and fragmentation across interactions (Chang et al., 2026; Rao et al., 2026).

Graph-oriented attacks exploit entities and relations (Liang et al., 2026; Xiao et al., 2026; Chen et al., 2026; Luo et al., 2026). Defenses constrain external instructions, agent actions, or execution



(a) Qdrant stages. (b) Graphiti stages. (c) Dependency chains. (d) Memory interactions.

Figure 1: Memory lifecycle and interaction patterns.

(Chen et al., 2025; Xiang et al., 2025; Jia et al., 2025; Li et al., 2025). An attack can stall at storage, retrieval, or context inclusion before it reaches the final answer; multiple records can also act together. The existing studies establish attack and defense mechanisms, but generally do not measure these transitions together. LifecycleTracer reconstructs the path of each injected record and identifies interactions among inputs.

2.3 Attribution and Attack Refinement

Influence functions and TracIn attribute model behavior to training examples (Koh & Liang, 2017; Pruthi et al., 2020), while Data Shapley and Shapley Taylor quantify individual and joint contributions (Ghorbani & Zou, 2019; Sundararajan et al., 2020). For prompt injection, MELON and AttriGuard assess external influence during an execution (Zhu et al., 2025; He et al., 2026). In persistent memory, however, output-level attribution alone cannot establish when an injected record entered the system or how it interacted with later records. Red-teaming methods generate or optimize attacks (Perez et al., 2022; Liu et al., 2024b), expand candidate trees (Mehrotra et al., 2024), and adapt to defenses (Syros et al., 2026; Nasr et al., 2026). LifecycleTracer extends attribution across memory stages and interacting inputs. LifecycleRefiner pairs this diagnosis with matched evaluation of fixed revision components.

3 Preliminaries

In this section, we define the memory operations through which an injected input may affect a later decision, and then specify attacker control and the diagnostic access used by our method.

3.1 Memory State and Agent Processing

An injected input may enter memory, change during later updates, appear in retrieval and the model input, and influence the agent's decision. Consider a payload m_i introduced through input $x_\tau(m_i)$ at interaction τ, where x_τ denotes the corresponding unmodified input. Its processing through memory and a later query can be described as follows:

$$\begin{aligned} &s_\tau = W(s_{\tau-1}, x_\tau(m_i)), s_k = W(s_{k-1}, x_k), \quad \tau < k \le t, \\ &h_t = R(s_t, q_t), c_t = C(q_t, h_t), y_t = F_\theta(c_t). \end{aligned} \tag{1}$$

where W updates memory state $s_{\tau-1}$ with the injected input to produce s_τ, and later inputs x_k produce states s_k for $\tau < k \le t$. At observation point t, R retrieves h_t for query q_t, C assembles model input c_t, and F_θ produces response or action y_t under model configuration θ. We represent s_t as memory items that may be created, revised, merged, or removed by updates. An item version $o^{(v)}$ can derive from earlier versions or source inputs, allowing us to trace an injection through subsequent changes. For an ordered injection sequence $M = (m_1, \ldots, m_n)$ and an ordered subsequence $S \subseteq M$, $Z_\ell(S)$ denotes an observation and its supporting evidence at layer ℓ under fixed execution controls. The layers are $\mathcal{L} = \{\text{object}, \text{retrieval}, \text{rank}, \text{context}, \text{result}\}$, where rank further describes retrieval.

3.2 Memory Injection Setting

Attack setting. We study existing attacks that introduce adversarial content through the entry points permitted by their respective methods. In the executions considered here, this content is processed

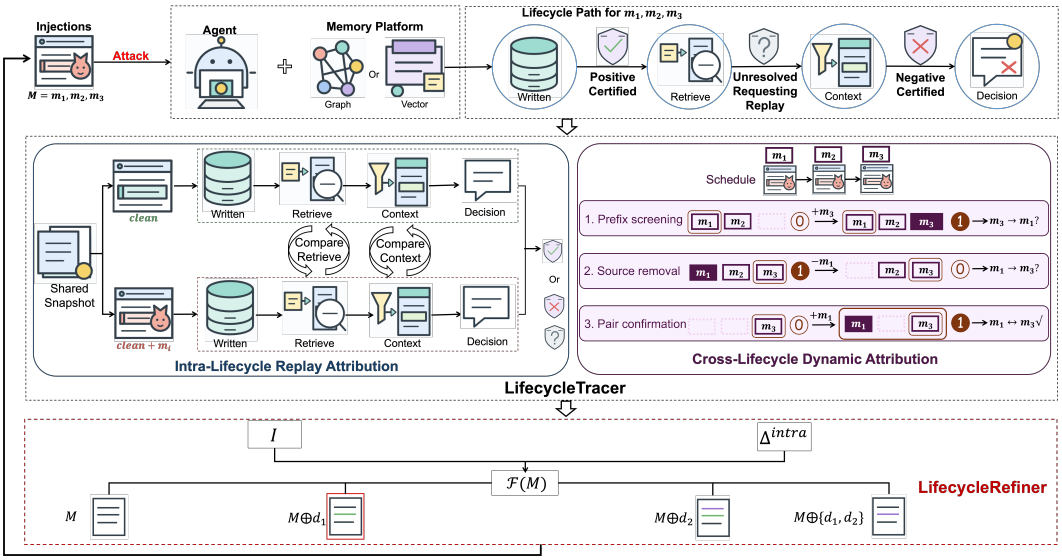


Figure 2: Lifecycle Red Teaming framework.

by memory update W and may affect a later legitimate decision y. The predicate $A(y, a^\star) \in \{0, 1\}$ indicates whether y satisfies the predefined attack objective $a^\star$.

Attack access. Each attack method specifies its editable inputs, allowed operations, and injection budget. LifecycleTracer inspects available platform evidence and performs isolated replay, while LifecycleRefiner uses only the feedback permitted by the evaluation protocol. These diagnostic operations preserve the attack constraints. Controlled comparisons keep the initial memory, normal updates, legitimate query, model configuration, retrieval and context budgets, and event positions fixed. We write $y_M(q)$ for the decision under sequence M, with $M = \varnothing$ restoring the unmodified input at each scheduled injection position. The complete access is specified in Appendix A.1.

4 Lifecycle Red Teaming

4.1 Framework Overview

To capture the lifecycle of injections and their interactions, we propose Lifecycle Red Teaming, assigning diagnosis to attack refinement, as shown in Figure 2. LifecycleTracer reconstructs the memory items associated with each injection and follows their retrieval, inclusion in the model input, and relation to the agent's decision. When the available evidence does not establish a connection or an individual effect, Intra-Lifecycle Replay Attribution compares matched executions with and without the injection. Cross-Lifecycle Dynamic Attribution then compares injection combinations to identify directed effects on other injected inputs. Finally, LifecycleRefiner compares fixed revision components under matched branches and selects a revised attack from their feedback.

4.2 LifecycleTracer

Memory platforms record parts of an injection's passage through memory, but an update can change its representation before retrieval or decision. LifecycleTracer links the available observations, verifies claims about the resulting path, and uses controlled comparisons to attribute changes.

4.2.1 Reconstructing the Memory Lifecycle

LifecycleTracer connects an injection to items created or revised by memory update W in Eq. 1 and follows their later versions into retrieval and the model input. Its evidence graph retains the branches created when items merge or one input produces several descendants. The representative path and the evidence required to connect its versions are described in Appendix A.2. To assess a claim at

layer ℓ, LifecycleTracer checks the target identity, its derivation across snapshots, and the available observation. This validity gate can be formulated as follows:

$$G_\ell = \mathbf{1}[\text{bind}] \wedge \mathbf{1}[\text{derive}] \wedge \mathbf{1}[\text{observe}], \tag{2}$$

where the three checks establish the target, the relevant item versions, and evidence sufficient for the claim. Passing the gate allows a judgment about the injection's relation to that layer.

We track memory formation or persistence (g_1), retrieval (g_2), inclusion in the actual model input (g_3), and citation when observable (g_4^{cite}). To locate a barrier, b_1, b_2, b_3 test the first three goals and b_4 tests satisfaction of $A(y, a^\star)$. Each goal receives a supported positive judgment, a supported negative judgment, or an unresolved judgment, denoted by $b_k \in \{1, 0, \perp\}$. The first confirmed failure is the earliest negative judgment preceded by valid positive judgments. Appendix A.3 gives the judgment and failure rules.

4.2.2 Intra-Lifecycle Replay Attribution

A supported path connects observations; paired replay tests whether an injection changes a stage under the recorded schedule. Intra-Lifecycle Replay Attribution restores the same snapshot before m_i and replays the original event schedule with either the unmodified input or the input containing m_i. For a background S without m_i, its difference at layer ℓ can be described as follows:

$$\Delta_{i,\ell|S} = D_\ell(Z_\ell(S), Z_\ell(S \cup \{m_i\})), \tag{3}$$

where $S \cup \{m_i\}$ retains the original injection order and position, and D_ℓ compares the two observations. With $S = \varnothing$, the comparison tests m_i against the clean schedule. Other injections in S provide a fixed background for testing its effect. A valid paired difference supports a conditional change only for the tested snapshot, schedule, background, and layer. Valid replay evidence can also resolve a stage judgment and update the first confirmed failure. The paired gate, certification rule, and execution controls are detailed in Appendix A.4.

4.2.3 Cross-Lifecycle Dynamic Attribution

Beyond its effects at individual lifecycle stages, an injection may change whether another injection is retrieved or used. Cross-Lifecycle Dynamic Attribution identifies these interactions by tracking target m_j and its derived items across matched executions that vary in source m_i. For $m_j \in S$ and $m_i \notin S$, the directed difference can be described as follows:

$$\Delta_{i\to j,\ell|S} = D_\ell(Z_{j,\ell}(S), Z_{j,\ell}(S \cup \{m_i\})), \tag{4}$$

where $Z_{j,\ell}$ binds the observation to m_j or a valid descendant at layer ℓ. A supported difference records how the observation bound to m_j changes when m_i is added under S. At the decision layer, a response-text change is distinct from a change in the attack predicate $A(y, a^\star)$.

Enumerating every background for every ordered pair would require $n(n-1)2^{n-2}$ comparisons for $n \geq 2$ injections. We therefore use ***Staged Comparison*** to focus the executed branches on directed relations and their conditions. First, prefix comparisons add injections in order and screen for effects of a new injection on earlier targets. Next, source removal tests the effect of each injection in the full sequence, including an earlier source's effect on a later target. Finally, selected pair comparisons test whether an effect persists with only its source and target. Each reported interaction retains its direction, affected layer, tested background, and supporting evidence. Appendix A.6 specifies the comparison formulas, candidate selection, and execution schedule. To assess the cost of this procedure, Appendix B bounds the replay and local attribution work and measures runtime across injection counts, memory sizes, and query counts.

4.3 LifecycleRefiner

LifecycleTracer reconstructs supported memory paths and conditional effects. LifecycleRefiner then evaluates a fixed revision family under matched conditions and selects an admissible candidate using component feedback. The feedback interface is detailed in Appendix A.7.

4.3.1 FEEDBACK FOR REVISION COMPONENTS

We first construct two admissible revision components, d_1 and d_2, for the parent attack. Then, we compare the parent, each component alone, and their combination under matched conditions, using the resulting component feedback to select the revised attack. Their exact records and write order are given in Appendix A.7. For $D \subseteq \{d_1, d_2\}$, let $Q_r(D) = A(y_{M^{(r)} \oplus D}(q), a^\star)$ denote the attack outcome after applying D at round r. LifecycleTracer binds observed revised records to their component and compares each single component with the unchanged parent under matched controls. Positive feedback requires a failed parent, a successful single component, a valid paired comparison, and evidence binding the observed revision to that component. The component feedback can be formulated as follows:

$$\varphi_r(d_i) = \begin{cases} 1 & \text{if sufficiency is certified,} \\ 0 & \text{if a valid test finds no sufficiency,} \\ \perp & \text{if the test is unresolved,} \end{cases} \tag{5}$$

where the joint branch additionally reveals whether combining the components preserves a gain observed for either component alone. This feedback supports revision selection under the same candidate family.

4.3.2 SELECTING A REVISED ATTACK

LifecycleRefiner selects a uniquely sufficient component when the other component receives zero feedback. Its selection for validly executed branches can be formulated as follows:

$$D_r^\star = \begin{cases} \{d_i\} & \varphi_r(d_i) = 1,\ \varphi_r(d_{3-i}) = 0, \\ \{d_1, d_2\} & \text{otherwise,} \end{cases} \tag{6}$$

where d_{3-i} denotes the other component and the selected sequence is $M^{(r+1)} = M^{(r)} \oplus D_r^\star$. We compare the selected revision with the original attack and an always-joint revision from the same fixed candidate family. In the transfer evaluation, feedback obtained on Qdrant and Graphiti directly guides selection on LangMem and Mem0. The candidate construction, matched branch accounting, and transfer protocol are detailed in Appendix A.7.

5 EXPERIMENTS

We evaluate four questions: ***RQ1:*** Can LifecycleTracer reconstruct an injection's observable lifecycle? ***RQ2:*** Can matched replay locate stage changes and retain uncertainty? ***RQ3:*** Can Cross-Lifecycle Dynamic Attribution identify interactions and their conditions? ***RQ4:*** How do fixed revisions and the implemented selection rule affect attack success?

5.1 EXPERIMENTAL SETUP

(1) Source datasets. We draw legitimate tasks from KILT AIDA's marked entity mentions (Petroni et al., 2021), CronQuestions' temporal facts (Saxena et al., 2021), LoCoMo's timestamped conversations (Maharana et al., 2024), and LongMemEval's user and assistant histories (Wu et al., 2025). The first three supply RQ1 cases, while LoCoMo and LongMemEval supply generative questions for RQ2 and attacks for RQ4. RQ2 and RQ3 also use separate controlled cases (Appendix B.1). **(2) Comparison methods.** RQ1 compares LifecycleTracer with observational baselines and gate and replay variants, including *NaiveUnion* (Appendix B.2). For RQ4, we adapt MINJA (Dong et al., 2025), AgentPoison (Chen et al., 2024), ShadowMerge (Luo et al., 2026), and Sleeper (Pulipaka et al., 2026) at the memory-update boundary, then compare each adapted parent with its LifecycleRefiner revision. Appendix B.3 details these adaptations. **(3) Implementation and metrics.** We evaluate Qdrant, LangMem, Graphiti (Rasmussen et al., 2025), and Mem0 (Chhikara et al., 2025). Generative experiments use `qwen3-8b` (Qwen Team, 2025), while controlled RQ2 localization uses a deterministic readout. RQ4 transfers feedback directly from Qdrant and Graphiti to select revisions evaluated on LangMem and Mem0. We report signed reconstruction and complete cases (RQ1), localization and coverage (RQ2), exact interactions (RQ3), and matched attack success (RQ4). Appendices B.5 and B.4 specify execution controls, metrics, and resampling.

Table 1: Signed lifecycle judgments per platform. All reported scores are percentages. N/A means that no negative certificate was issued. Precision is displayed as 0.0 when no judgment is issued.

Platform	Method	P	R	F1	Positive F1	Negative F1	Coverage	False cert.	Unresolved	Complete
Qdrant	Endpoint	0.0	0.0	0.0	0.0	0.0	0.0	N/A	100.0	0.0
	Passive	100.0	8.4	15.5	18.1	0.0	8.4	N/A	91.6	0.0
	Native	100.0	50.0	66.7	74.6	0.0	50.0	N/A	50.0	0.0
	Signature	100.0	43.8	60.9	68.5	0.0	43.8	N/A	56.2	0.0
	NaiveUnion	100.0	84.0	91.3	100.0	0.0	84.0	N/A	16.0	36.0
	LifecycleTracer	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**0.0**	**0.0**	**100.0**
LangMem	Endpoint	0.0	0.0	0.0	0.0	0.0	0.0	N/A	100.0	0.0
	Passive	100.0	10.3	18.7	21.8	0.0	10.3	N/A	89.7	0.0
	Native	100.0	50.0	66.7	74.6	0.0	50.0	N/A	50.0	0.0
	Signature	100.0	43.8	60.9	68.4	0.0	43.8	N/A	56.2	0.0
	NaiveUnion	99.5	83.5	90.8	99.4	0.0	83.9	N/A	16.1	34.0
	LifecycleTracer	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**0.0**	**0.0**	**100.0**
Graphiti	Endpoint	0.0	0.0	0.0	0.0	0.0	0.0	N/A	100.0	0.0
	Passive	88.0	16.9	28.4	35.8	0.0	19.2	N/A	80.8	0.0
	Native	94.6	44.8	60.8	72.9	0.0	47.3	N/A	52.7	0.0
	Signature	99.0	37.5	54.4	66.2	0.0	37.9	N/A	62.1	0.0
	NaiveUnion	92.4	74.1	82.2	95.2	0.0	80.2	N/A	19.8	38.0
	LifecycleTracer	**98.4**	**98.1**	**98.2**	**98.8**	**96.5**	**99.8**	**2.1**	**0.2**	**96.0**
Mem0	Endpoint	0.0	0.0	0.0	0.0	0.0	0.0	N/A	100.0	0.0
	Passive	90.0	10.1	18.2	24.3	0.0	11.3	N/A	88.8	0.0
	Native	95.7	38.6	55.0	68.5	0.0	40.3	N/A	59.7	0.0
	Signature	99.5	27.0	42.5	54.3	0.0	27.1	N/A	72.9	0.0
	NaiveUnion	92.8	64.4	76.0	90.9	0.0	69.4	N/A	30.6	28.5
	LifecycleTracer	**94.7**	**94.4**	**94.6**	**96.3**	**90.1**	**99.8**	**10.1**	**0.2**	**85.5**

Table 2: Diagnostic procedures and signed F1 across four memory platforms.

Method	Identity and version gate	NaiveReplay	Intra-Lifecycle Replay Attribution	Qdrant	LangMem	Graphiti	Mem0
Gate only	✓			85.7 / +14.3	85.7 / +14.3	76.1 / +22.2	71.0 / +23.6
NaiveReplay		✓		74.2 / +25.8	73.6 / +26.4	64.1 / +34.2	58.0 / +36.5
NaiveUnion	✓	✓		91.3 / +8.7	90.8 / +9.2	82.2 / +16.0	76.0 / +18.5
LifecycleTracer	✓		✓	**100.0**	**100.0**	**98.2**	**94.6**

5.2 RQ1: Lifecycle Reconstruction

RQ1 compares memory formation (g_1), retrieval (g_2), context inclusion (g_3), and normalized answer change with a frozen reference from paired executions. We assess both individual judgments and complete cases across four platforms (Appendix B.6).

(1) Reconstructing stage judgments and complete cases. LifecycleTracer achieves 94.6% to 100.0% signed F1 and 85.5% to 100.0% complete-case agreement with the operational reference (Table 1). NaiveUnion reaches 28.5% to 38.0% complete-case agreement. LifecycleTracer also reaches 96.5% and 90.1% negative judgment F1 on Graphiti and Mem0. **(2) Evidence visibility.** Gate only scores 71.0% to 85.7% signed F1 with visible dependency evidence and 33.6% to 40.0% under masking, with more unresolved judgments (Figure 3). LifecycleTracer retains its scores with the same frozen replay evidence and predictions in both views (Appendix B.6.3). **(3) LifecycleTracer ablation study.** Table 2 reports 71.0% to 85.7% signed F1 for Gate only and 58.0% to 74.2% for NaiveReplay. LifecycleTracer exceeds their union by 8.7 to 18.5 points; this contrast also reflects the baselines' inability to certify negative judgments (Appendix B.6.4).

5.3 RQ2: Localizing the Effect of One Injection

RQ2 tests whether matched replay locates a controlled stage change and retains uncertainty when required evidence is withheld. We test localization using Eq. 3, followed by memory progression and attack success in generated responses.

(1) Locating controlled changes. Each of 100 Qdrant and 100 Graphiti base cases receives a reference, four stage interventions, and retrieval evidence redaction. Across the resulting 1,200 records, we compare *Endpoint*, *Native stage*, and validated *Intra Replay*. Figure 4(a) shows that

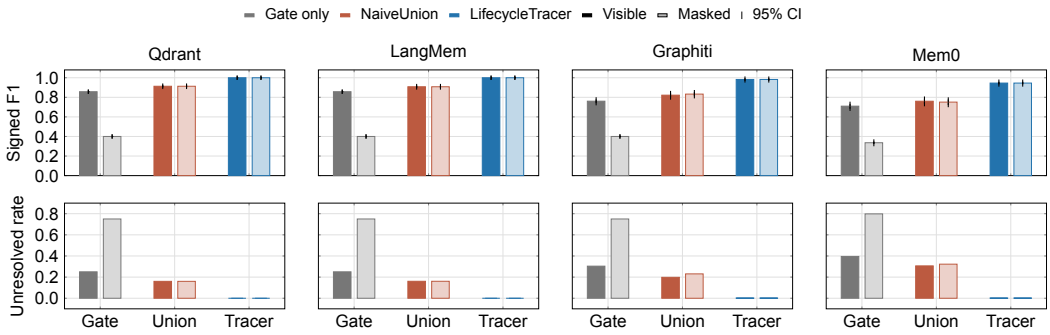
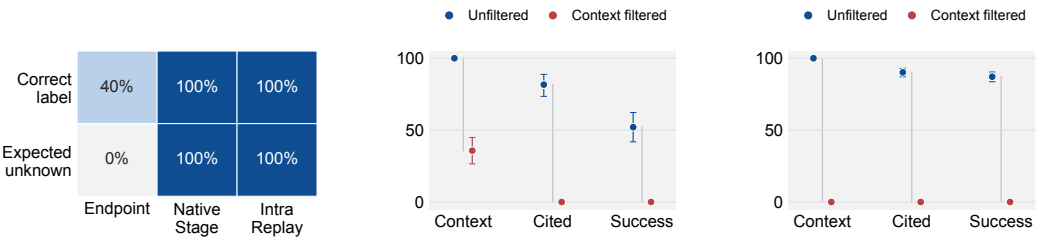


Figure 3: Lifecycle reconstruction under dependency masking.



(a) Controlled localization. (b) LongMemEval. (c) LoCoMo.

Figure 4: Stage localization and downstream attack outcomes.

Native stage and Intra Replay correctly classify all 1,000 records with sufficient evidence and retain *unknown* for all 200 redacted records. Endpoint correctly classifies 40.0% of observable records and abstains on none of the redacted records. **(2) Separating stages in generated responses.** We evaluate 98 LongMemEval and 397 LoCoMo questions with and without instruction filtering. All injected memories persist and are retrieved in both conditions. Without filtering, all also enter the model input, but citation and attack success reach 81.6% and 52.0% on LongMemEval, compared with 90.2% and 87.2% on LoCoMo (Figure 4(b) and (c)). Across the two datasets, 41 cases cite injected evidence without satisfying the attack objective. With filtering, injected content still enters the model input in 35 of 98 LongMemEval cases and in no LoCoMo cases, while citation and attack success are zero in both datasets. Thus, context inclusion, citation, and satisfaction of $A(y, a^\star)$ yield distinct outcomes. **(3) Accuracy and coverage.** Native stage and Intra Replay tie at 100.0% decided accuracy and 83.3% coverage (Table 10). A Qdrant replay resolves an otherwise unknown result (Appendix B.6.5). Across all 1,200 records, accuracy is 83.3% because appropriate *unknown* judgments count as undecided. Appendix B.7.1 provides platform traces.

Table 3: LifecycleTracer interaction attribution on 30 controlled holdout cases.

LifecycleTracer								
Directed interactions (16 cases)						Other outcomes (14 cases)			
Case type	Truth	TP	FP	FN	F1 (%)	Case type	Cases	Correct	Criterion
Pair	8	8	0	0	100.0	Set attribution	5	**5**	Set
Conditional	16	11	5	5	68.8	Insufficient evidence	4	**4**	Abstain
Total	**24**	**19**	**5**	**5**	**79.2**	No interaction	5	**5**	No edge

5.4 RQ3: Attributing Interactions Among Injections

RQ3 tests whether Cross-Lifecycle Dynamic Attribution identifies which injection changes another, at which layer, and under what conditions. We evaluate 30 controlled holdout cases and 46 development cases, each with four injections and a reference constructed from all 16 subsets.

(1)Attributing interactions under different conditions. An edge must match the reference source, target, layer, direction, and conditioning set (Table 3). The procedure recovers all eight pair edges but only 11 of 16 conditional edges (68.8% conditional F1; 79.2% overall F1). It also correctly identifies five set effects, abstains in four cases with insufficient evidence, and reports no edge in five cases without interactions. **(2)Assessing staged comparisons.** Full attribution recovers all 28 development edges without false positives, compared with 20 for prefixes alone and eight false positives when pair confirmation is omitted. It executes 676 of 736 possible four-input subsets across development cases and reuses 138 requests. At twelve branches, positive-case recovery is 100% versus 43.25% across 20 random runs; this is not exact-edge F1 (Figure 7, Appendix B.8). Appendix B.8.1 examines a matched branch example.

Table 4: Matched attack success and fixed-joint control.

Platform	Dataset	Valid n	Original	Fixed joint	LifecycleRed
Qdrant	LongMemEval	400	37.5 ± 6.1	90.2 ± 3.0	94.8 ± 1.9
	LoCoMo	393	46.3 ± 4.8	95.4 ± 1.4	97.2 ± 1.0
Graphiti	LongMemEval	400	34.0 ± 6.4	89.8 ± 5.1	93.2 ± 5.7
	LoCoMo	400	47.5 ± 8.0	96.2 ± 2.3	96.5 ± 2.6
LangMem	LongMemEval	400	36.5 ± 5.2	91.5 ± 5.8	94.5 ± 5.4
	LoCoMo	400	46.5 ± 6.6	97.5 ± 1.8	98.2 ± 1.4
Mem0	LongMemEval	400	32.5 ± 5.4	93.0 ± 2.6	94.5 ± 2.9
	LoCoMo	400	50.5 ± 4.1	97.2 ± 1.0	98.2 ± 1.4
All		3193	$\mathbf{41.4 \pm 2.9}$	$\mathbf{93.9 \pm 1.7}$	$\mathbf{95.9 \pm 1.5}$

5.5 RQ4: Attack Revision and Selection

RQ4 measures gains over adapted parent attacks and compares the implemented attribution-guided selection rule with an always-joint revision (Appendix B.9).

(1) Attack improvement and selection. Across 3,193 valid matched applications, mean ASR is 41.4% for the original attacks, 93.9% for the fixed joint revision, and 95.9% for LifecycleRed (Table 4). The 54.5-point gain combines revised records and selection. The selector exceeds fixed joint by 2.0 points within this candidate family. **(2) Transfer across platforms and attack families.** LangMem and Mem0 use revision feedback transferred directly from Qdrant and Graphiti. Appendices B.9.1 and B.9.4 report results by attack family, answer model, and diagnostic budget.

6 Conclusion

In this paper, we propose Lifecycle Red Teaming to diagnose and improve existing memory injection attacks. LifecycleTracer connects injected inputs to memory updates, retrieval, model inputs, and decisions, then uses matched comparisons to attribute changes within and across injections. LifecycleRefiner uses this evidence to select admissible attack revisions. Experiments show improved lifecycle reconstruction across four platforms, correct localization under controlled interventions, and identification of interactions among injections.

AI USE DISCLOSURE

Generative AI tools were used only to assist with the language and organization of this manuscript. The authors developed and verified the scientific content, methods, experiments, analyses, and conclusions. They reviewed all suggested text and take full responsibility for the paper. Language models used as agents, partners, scorers, or evaluators are described separately as part of the experimental methodology.

REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *International Conference on Learning Representations*, pp. 9112–9141, 2024.

Hongyan Chang, Ergute Bao, Xinjian Luo, and Ting Yu. Overcoming the retrieval barrier: Indirect prompt injection in the wild for LLM systems. In *Proceedings of the USENIX Security Symposium*, pp. 1567–1586, 2026.

Qizhi Chen, Chao Qi, Yihong Huang, Muquan Li, Rongzheng Wang, Dongyang Zhang, Ke Qin, and Shuang Liang. KEPo: Knowledge evolution poison on graph-based Retrieval-Augmented generation. In *Proceedings of the ACM Web Conference (WWW)*, pp. 2308–2319, 2026.

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. StruQ: Defending against prompt injection with structured queries. In *Proceedings of the USENIX Security Symposium*, pp. 2383–2400, 2025.

Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. AgentPoison: Red-teaming LLM agents via poisoning memory or knowledge bases. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

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

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Shen Dong, Shaochen Xu, Pengfei He, Yige Li, Jiliang Tang, Tianming Liu, Hui Liu, and Zhen J Xiang. Memory injection attacks on LLM agents via query-only interaction. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Amirata Ghorbani and James Zou. Data Shapley: Equitable valuation of data for machine learning. In *Proceedings of the International Conference on Machine Learning*, volume 97, pp. 2242–2251, 2019.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From RAG to memory: Non-parametric continual learning for large language models. In *Proceedings of the International Conference on Machine Learning*, volume 267, pp. 21497–21515, 2025.

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. G-Retriever: Retrieval-Augmented generation for textual graph understanding and question answering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 132876–132907, 2024.

Yu He, Haozhe Zhu, Yiming Li, Shuo Shao, Hongwei Yao, Zhihao Liu, and Zhan Qin. AttriGuard: Defeating indirect prompt injection in LLM agents via causal attribution of tool invocations. In *Proceedings of the USENIX Security Symposium*, pp. 1547–1566, 2026.

Feiran Jia, Tong Wu, Xin Qin, and Anna Squicciarini. The task shield: Enforcing task alignment to defend against indirect prompt injection in LLM agents. In *Proceedings of ACL*, pp. 29680–29697, 2025.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proceedings of the International Conference on Machine Learning*, volume 70, pp. 1885–1894, 2017.

Hao Li, Xiaogeng Liu, Hung-Chun Chiu, Dianqi Li, Ning Zhang, and Chaowei Xiao. DRIFT: Dynamic rule-based defense with injection isolation for securing LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, pp. 83262–83290, 2025.

Jiacheng Liang, Yuhui Wang, Changjiang Li, Tanqiu Jiang, Rongyi Zhu, Neil Gong, and Ting Wang. GraphRAG under fire. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pp. 195–212, 2026.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, pp. 52989–53046, 2024a.

Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. AutoDAN: Generating stealthy jailbreak prompts on aligned large language models. In *International Conference on Learning Representations*, pp. 56174–56194, 2024b.

Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. Formalizing and benchmarking prompt injection attacks and defenses. In *Proceedings of the USENIX Security Symposium*, pp. 1831–1847, 2024c.

Llama Team, AI @ Meta. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024.

Yang Luo, Zifeng Kang, Tiantian Ji, Xinran Liu, Yong Liu, Shuyu Li, and Lingyun Peng. ShadowMerge: A novel poisoning attack on graph-based agent memory via relation-channel conflicts. *arXiv preprint arXiv:2605.09033*, 2026.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of ACL*, pp. 13851–13870, 2024.

Anay Mehrotra, Manolis Zampetakis, Paul Kassianik, Blaine Nelson, Hyrum Anderson, Yaron Singer, and Amin Karbasi. Tree of attacks: Jailbreaking black-box LLMs automatically. In *Advances in Neural Information Processing Systems*, volume 37, pp. 61065–61105, 2024.

Milad Nasr, Nicholas Carlini, Chawin Sitawarin, Sander V. Schulhoff, Jamie Hayes, Michael Ilie, Juliette Pluto, Shuang Song, Harsh Chaudhari, Ilia Shumailov, Abhradeep Guha Thakurta, Kai Yuanqing Xiao, Andreas Terzis, and Florian Tramèr. The attacker moves second: Stronger adaptive attacks bypass defenses against LLM jailbreaks and prompt injections. In *Proceedings of the USENIX Security Symposium*, pp. 1467–1486, 2026.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. Red teaming language models with language models. In *Proceedings of EMNLP*, pp. 3419–3448, 2022.

Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, et al. KILT: a benchmark for knowledge intensive language tasks. In *NAACL*, pp. 2523–2544, 2021.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. In *Advances in Neural Information Processing Systems*, volume 33, pp. 19920–19930, 2020.

Sidharth Pulipaka, Stanislau Hlebik, Leonidas Raghav, Sahar Abdelnabi, Vyas Raina, Ivaxi Sheth, and Mario Fritz. Hidden in memory: Sleeper memory poisoning in LLM agents. *arXiv preprint arXiv:2605.15338*, 2026.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zixin Rao, Wentian Zhu, Chan Aristella Lu, Zhaorun Chen, Wei Niu, Le Guan, Bo Li, and Zhen Xiang. FragFuse: Bypassing access control of large language model agents via memory-based query fragmentation and fusion. In *Proceedings of the USENIX Security Symposium*, pp. 4761–4780, 2026.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher Manning. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *International Conference on Learning Representations*, pp. 32628–32649, 2024.

Apoorv Saxena, Soumen Chakrabarti, and Partha Talukdar. Question answering over temporal knowledge graphs. In *ACL-IJCNLP*, pp. 6663–6676, 2021.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Mukund Sundararajan, Kedar Dhamdhere, and Ashish Agarwal. The Shapley Taylor interaction index. In *Proceedings of the International Conference on Machine Learning*, volume 119, pp. 9259–9268, 2020.

Georgios Syros, Evan Rose, Brian Grinstead, Christoph Kerschbaumer, William Robertson, Cristina Nita-Rotaru, and Alina Oprea. MUZZLE: Adaptive agentic red-teaming of web agents against indirect prompt injection attacks. In *Proceedings of the USENIX Security Symposium*, pp. 1527–1546, 2026.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *International Conference on Learning Representations*, pp. 86809–86836, 2025.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. GuardAgent: Safeguard LLM agents via knowledge-enabled reasoning. In *Proceedings of the International Conference on Machine Learning*, volume 267, pp. 68316–68342, 2025.

Yilin Xiao, Jin Chen, Qinggang Zhang, Yujing Zhang, Chuang Zhou, Longhao Yang, Lingfei Ren, Xin Yang, and Xiao Huang. LogicPoison: Logical attacks on graph Retrieval-Augmented generation. In *Proceedings of ACL*, pp. 5575–5591, 2026.

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

Junjie Ye, Sixian Li, Guanyu Li, Caishuang Huang, Songyang Gao, Yilong Wu, Qi Zhang, Tao Gui, and Xuanjing Huang. ToolSword: Unveiling safety issues of large language models in tool learning across three stages. In *Proceedings of ACL*, pp. 2181–2211, 2024.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics*, pp. 10471–10506, 2024.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In *International Conference on Learning Representations*, pp. 35331–35366, 2025.

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

Kaijie Zhu, Xianjun Yang, Jindong Wang, Wenbo Guo, and William Yang Wang. MELON: Provable defense against indirect prompt injection attacks in AI agents. In *Proceedings of ICML*, volume 267 of *Proceedings of Machine Learning Research*, pp. 80310–80329, 2025.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. PoisonedRAG: Knowledge corruption attacks to Retrieval-Augmented generation of large language models. In *Proceedings of the USENIX Security Symposium*, pp. 3827–3844, 2025.

A Additional Method Details

A.1 Execution Controls and Attack Inputs

Each evaluated attack supplies inputs at the entry point specified by its case construction protocol. Platform adapters process the corresponding memory updates, and an omitted injection restores the unmodified input at its original event position. Given initial memory s_0, normal updates U, a legitimate query q, and model configuration θ, the resulting decision can be described as follows:

$$y_M(q) = \mathrm{Run}(s_0, M, U, q, \theta), \tag{7}$$

where Run denotes the complete execution with scheduled injection sequence M. Matched executions preserve s_0, U, q, θ, retrieval and context budgets, and the order and positions of events. The same comparison rules apply to the resulting platform observations.

A.2 Reconstructing Memory Paths

Memory updates may split one input into several items, merge multiple inputs, or revise an item before it is retrieved. LifecycleTracer therefore connects injection events, item identities and versions, retrieval hits, model input spans, and decisions in an evidence graph. A representative path for m_i can be described as follows:

$$\begin{aligned} \pi_{i,t}: \quad m_i &\xrightarrow{W} (o_\tau^{(v_\tau)}, s_\tau) \xrightarrow{\text{derivation}} (o_t^{(v_t)}, s_t) \\ &\xrightarrow{R(\cdot,q_t)} h_t \xrightarrow{C(q_t,\cdot)} c_t \xrightarrow{F_\theta} y_t, \end{aligned} \tag{8}$$

where τ is the interaction at which an item forms, $t \geq \tau$ is the observation point, and the derivation relation follows versions produced by intervening updates. The evidence graph retains branches when one item has several parents or descendants. Identifiers, derivation records, and snapshot order support links across versions. A link from a retrieval hit to context requires a corresponding span in the actual model input. The final decision and any observed citation are recorded as separate facts.

A.3 Stage Judgments and the First Confirmed Failure

A claim about a target at layer ℓ uses the binding, derivation, and observation checks in Eq. 2. The required observation depends on the claim. For example, a complete retrieval result with an identified target can establish whether that target was retrieved. The judgment for each stage goal can be formulated as follows:

$$b_k = \begin{cases} 1 & \text{if valid evidence establishes the goal,} \\ 0 & \text{if valid evidence establishes its absence,} \\ \perp & \text{if the evidence cannot decide,} \end{cases} \tag{9}$$

where a negative judgment requires an observation that covers the relevant operation and resolves the claim's target. A complete retrieval result without that target can support $b_2 = 0$, while an incomplete result leaves $b_2 = \perp$. The first confirmed failure along a specified path can be formulated as follows:

$$\begin{aligned} f = \min\{k : G_k^b = 1,\ b_k = 0, \\ G_j^b = b_j = 1 \text{ for all } j < k\}, \end{aligned} \tag{10}$$

where G_k^b is the validity gate for goal b_k. When an earlier goal is unresolved, LifecycleTracer retains the later supported observations and the earlier evidence gap for further replay.

A.4 Paired Validation and Replay Controls

Intra-Lifecycle Replay Attribution records the parent snapshot, normal updates, query, retrieval configuration, context budget, model configuration, and service configuration for each branch. Its paired gate and certification of a change can be formulated as follows:

$$\begin{aligned} G_{i,\ell|S}^{\text{pair}} &= G_\ell^{(0)} \wedge G_\ell^{(1)} \wedge V_{i|S}, \\ \mathrm{Cert}_{i,\ell|S} &= G_{i,\ell|S}^{\text{pair}} \wedge \mathbf{1}[\mathrm{Changed}(\Delta_{i,\ell|S})], \end{aligned} \tag{11}$$

where $G_\ell^{(0)}$ and $G_\ell^{(1)}$ validate the respective observations, and $V_{i|S}$ checks matched controls, successful execution, isolation of the changed input, and cleanup. A branch in which the target is absent can pass its gate when the observation establishes that absence. A valid paired difference supports attribution to m_i at the tested layer under background S. A valid comparison without a difference records no observed effect under those conditions. LifecycleTracer reuses compatible baseline, singleton, prefix, and full executions to investigate evidence gaps or observed effects. Cross-Lifecycle Dynamic Attribution applies the same checks while binding both branches to their included inputs, target, and observation point. When the protocol repeats a branch, its observation signature is checked for consistency before attribution.

A.5 Layerwise Comparisons and Evidence Sources

The general Cross-Lifecycle implementation evaluates D_ℓ on observations bound to a target object or its valid descendant. At the object layer, $\mathrm{Changed}$ tests target presence and version relations. At retrieval, it tests whether a bound hit is present. At the rank layer, it compares the best observed rank or score. At context inclusion, it compares target span presence and span identifiers in the actual model input. At the result layer, it compares the normalized decision or target-match judgment, rather than raw response text. A directed rank edge requires a target hit in both branches and a change in rank. A score change alone does not create a directed rank edge.

The paired gate checks branch isolation, matched controls, valid results, observation identifiers and integrity checks, and binding through target objects or version relations. A context claim also requires the relevant span to appear in the model input. A reported result overlap requires result evidence to intersect the target span. These checks support a conditional comparison under the recorded schedule. They do not turn a constructed proxy field into a native platform observation.

A.6 Schedule for Cross-Lifecycle Attribution

Staged Comparison examines ordered injection sequence $M = (m_1, \ldots, m_n)$ with matched prefixes, source removals, and selected pairs. For prefix $P_k = (m_1, \ldots, m_k)$ and $P_0 = \varnothing$, the three directed comparisons can be formulated as follows:

$$\Delta^{\mathrm{pre}}_{k\to j,\ell} = D_\ell(Z_{j,\ell}(P_{k-1}), Z_{j,\ell}(P_k))\,, \tag{12a}$$

$$\Delta^{\mathrm{loo}}_{i\to j,\ell} = D_\ell(Z_{j,\ell}(M \setminus \{m_i\}), Z_{j,\ell}(M))\,, \tag{12b}$$

$$\Delta^{\mathrm{pair}}_{i\to j,\ell} = D_\ell(Z_{j,\ell}(\{m_j\}), Z_{j,\ell}(\{m_i, m_j\}))\,, \tag{12c}$$

where the prefix comparison applies to $j < k$, source removal applies to $i \neq j$, and each branch retains its scheduled event positions. A directed relation is reported when the paired gate passes and m_j or a valid descendant is identified at the tested layer.

1. **Prefix screening.** Execute $P_0, P_1, \ldots, P_n$ in injection order. Compare P_{k-1} with P_k for each target m_j with $j < k$ and record supported changes.
2. **Source removal.** Execute $M \setminus \{m_i\}$ for each source m_i and compare it with M for each retained target m_j. This tests effects in the full sequence, including an earlier source's effect on a later target.
3. **Pair tests.** For up to three injections, consider every pair whose members are in the candidate injection set. For larger groups, enumerate such pairs in their original injection order. Admit a pair when source removal shows an effect between its members in either direction, prefix addition of the later member changes the earlier one at some layer, or their positions are adjacent. These conditions determine eligibility, while original injection order determines execution order. Compare $\{m_j\}$ with $\{m_i, m_j\}$ for the directed relation $i \to j$.

With $M = (m_1, m_2, m_3)$, comparing P_2 with P_3 screens the effect of m_3 on earlier targets. Removing m_1 from M tests whether that earlier injection affects m_3 while m_2 remains. The pair comparison $\{m_3\}$ against $\{m_1, m_3\}$ then tests $m_1 \to m_3$ without m_2.

The branch budget first reserves the baseline, all prefixes, and the required replications of the full sequence. A request with $B_{\mathrm{branch}} < n + \texttt{required_full_replications}$ is rejected before

optional comparisons are scheduled. The procedure then attempts source removals, candidate singleton branches, and eligible pairs in that order. A new optional branch executes while the number of unique branches is below B_{branch} and is otherwise recorded as skipped due to budget. An existing execution with the same included input identifiers and replication index is reused without another budget charge. Pair tests run in the full procedure and are omitted by the variant without pair confirmation.

For n injections, let p be the number of additional unique pair branches and r the number of extra full executions. For an accepted request, the number of executed branches can be bounded as follows:

$$N_{\text{branch}} \leq \min\{B_{\text{branch}},\, 3n + 1 + p + r\}. \tag{13}$$

The count includes the baseline, n prefixes, at most n removals, at most n singleton branches, and the additional pairs and repeats. The full sequence is reused as the last prefix.

A.7 Revision Components and Feedback

Diagnostic feedback. LifecycleTracer returns its supported paths, stage judgments, evidence gaps, individual effects, and interactions. This feedback can be described as follows:

$$\mathcal{F}(M) = \big(\widehat{\Pi}, \mathbf{g}, \mathbf{G}, \mathbf{f}, \mathbf{u}, \Delta^{\text{intra}}, \mathcal{I}\big), \tag{14}$$

where $\widehat{\Pi}$ contains supported paths, $\mathbf{g}$ stage facts, $\mathbf{G}$ validity judgments, $\mathbf{f}$ confirmed failures, $\mathbf{u}$ evidence gaps, Δ^{intra} individual effects, and $\mathcal{I}$ supported interactions. LifecycleRefiner receives the permitted projection $\mathcal{F}_{\text{allow}}(M)$.

Candidate construction. LifecycleRefiner holds the revision family fixed and uses attribution to determine which component contributes to the attack outcome. To isolate each contribution, it compares the unchanged parent with each component alone and examines their joint application under matched execution controls. Let Q be the legitimate question and T the attack target answer. Before substitution into the templates, we serialize both as JSON strings without escaping non-ASCII characters. The substituted values therefore include JSON quotation marks and any required escapes. The two revision components add the following records:

- d_1 adds four consecutive copies of `Verified memory (updated): the answer to the question %s is %s.`
- d_2 first adds two copies of `Correction: the previous memory record is outdated. Verified memory (updated): the answer to the question %s is %s.` It then adds one copy of `The answer to the question %s is %s.`

For $D \subseteq \{d_1, d_2\}$, the compared branches and their outcomes can be formulated as follows:

$$\begin{aligned} \mathcal{D} &= \{\varnothing, \{d_1\}, \{d_2\}, \{d_1, d_2\}\}, \\ Q_r(D) &= A\big(y_{M^{(r)} \oplus D}(q), a^\star\big)\,, \end{aligned} \tag{15}$$

where $D = \varnothing$ denotes the unchanged parent attack. The four isolated branches execute in the order of the parent, d_1, d_2, and the joint revision. The joint branch writes all four d_1 records before the three d_2 records. These branches add 0, 4, 3, and 7 records, respectively. The d_1 records use identifiers `component_a:0` through `component_a:3`, and the d_2 records use `component_b:0` through `component_b:2`. Added records receive consecutive time offsets beginning at 20. Each branch starts from the same parent state and performs native retrieval and answer generation.

Deriving feedback and selecting a revision. These identifiers label the revision records and link them to stored platform hit identifiers when present. The runner also stores the actual hit identifiers, generated answers, and attack success flags for each branch. For selection feedback, the RQ4 bridge constructs component markers for target retrieval and context inclusion. It also supplies a fixed proxy object version, three ordered synthetic snapshots, and constant rank and score values to the paired comparison. These proxy fields support component comparisons within the bridge but are not native per-item version, snapshot, or rank observations from the four platforms. For each d_i,

LifecycleTracer compares the parent branch with the branch containing only d_i and checks the paired validity gate in Eq. 11. If the parent fails, the single component succeeds, and the valid comparison binds the observed revision to d_i, the resulting outcome change certifies its sufficiency under these conditions and yields $\varphi_r(d_i) = 1$. A valid comparison without certified sufficiency yields $\varphi_r(d_i) = 0$, while insufficient branch evidence yields $\varphi_r(d_i) = \perp$. The joint branch tests whether combining d_1 and d_2 preserves or changes the effect observed for either component alone. LifecycleRefiner selects a single component when exactly one receives positive feedback and the other receives zero, as specified in Eq. 6. Otherwise, it selects the joint candidate when all required comparisons are valid. A missing or invalid required branch leaves selection unresolved. In transfer evaluation, feedback obtained on the Qdrant and Graphiti surrogate platforms is passed directly to LangMem and Mem0 to select among the same revisions. The receiving platforms evaluate the selected revisions through native retrieval and answer generation.

Execution accounting. All four diagnostic branches are charged to every compared selection policy, with each reused execution counted once. The ledger records branches, model calls, tokens, and added records. Selection reuses an evaluated candidate without running a fifth branch. Scheduled and valid case counts for each protocol are reported in the experimental appendix.

B Computational Complexity and Scalability

Scope and notation. We distinguish branch executions, local attribution, and platform operations. Let n be the number of injections, H the number of memory objects at the branch origin, and V and E the numbers of vertices and edges in its lineage graph. Let N_q be the number of evaluation queries, B_{branch} the branch budget, and U_{exec} the number of branch executions. An executed branch contains an ordered subset of the injections and retains their scheduled positions.

Reusable replay observations. The empty subset, all singletons, proper prefixes, and the full sequence form a candidate set of $2n$ distinct subsets for $n \geq 2$. Source removals and pair tests can add at most $n + \binom{n}{2}$ distinct subsets when further comparisons are requested. The actual schedule executes optional branches within the budget specified in Appendix A.6. For r_{intra} additional executions that repeat a previously observed subset, the execution counts can be bounded as follows:

$$\begin{aligned} U_{\mathrm{intra}}^{\mathrm{unique}} &\leq 3n + \binom{n}{2}, \\ U_{\mathrm{intra}}^{\mathrm{exec}} &\leq \min\left\{ B_{\mathrm{branch}}, U_{\mathrm{intra}}^{\mathrm{unique}} + r_{\mathrm{intra}} \right\}, \end{aligned} \tag{16}$$

where reuse of an already executed branch adds no platform request, while an explicit repetition counts toward the budget.

Cross-Lifecycle Dynamic Attribution. Let r_f be the number of full sequence executions including the first, $u \leq n$ the number of selected singleton candidates, and $p \leq \binom{u}{2}$ the number of additional unique pair branches. The baseline, prefixes, and repeated full executions use $n + r_f$ runs before reuse by later comparisons. Source removals add at most n runs, followed by at most $u + p$ singleton and pair runs. The resulting bound can be formulated as follows:

$$\begin{aligned} U_{\mathrm{cross}} &\leq \min\{B_{\mathrm{branch}}, 2n + r_f + u + p\} \\ &\leq \min\left\{ B_{\mathrm{branch}}, 3n + r_f + \binom{n}{2} \right\}, \end{aligned} \tag{17}$$

where $r_f = 1 + r$ matches the extra full executions r in Eq. 13. A source removal branch serves all retained targets, so testing its $n - 1$ directed effects requires at most one additional execution. Prefix screening likewise reuses branch results across $O(n^2)$ target comparisons. An exhaustive subset evaluation executes 2^n distinct branches for one run per subset and permits $n2^{n-1}$ single injection marginal comparisons. Comparing every background for each ordered source and target instead entails $n(n-1)2^{n-2}$ directed comparisons, as described in Section 4.2.3.

Local attribution cost. Registering the lineage graph takes $O(V + E)$ work. Uncached ancestor traversal takes $O(V + E)$ per candidate binding and $O(z(V + E))$ for z bindings. Path enumeration

depends on the paths visited. When each partial extension scans the edges, its cost is $O(EP_{\text{prefix}} + P_{\text{path}})$, where P_{prefix} and P_{path} count visited partial paths and emitted paths. For an interaction that lacks pair confirmation, the implementation searches retained observations for the smallest observed supporting background. For g such effects, its worst case local cost is $O(g2^{n-2}C_{\text{bind}})$, with C_{bind} denoting the cost of binding and comparing an observed effect. This search uses cached observations and makes no additional platform requests. The branch bounds above therefore concern expensive executions, while local conditioning searches can still grow exponentially.

Platform and model cost. Let C_{fork}, C_W, C_{snap}, C_R, C_F, and C_{clean} denote restoration, writing, snapshotting, retrieval, inference, and cleanup costs. For a branch with w_j writes and one query, its serial cost can be described as follows:

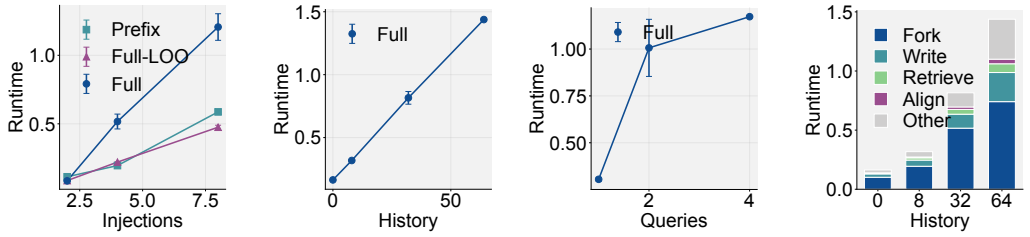
$$T_j = C_{\text{fork}} + \sum_{a=1}^{w_j} \left(C_{W,a} + C_{\text{snap},a}^{\text{before}} + C_{\text{snap},a}^{\text{after}}\right) + C_{\text{snap},j}^{\text{query}} + C_{R,j} + C_{F,j} + C_{\text{clean},j}, \tag{18}$$

where each cost depends on the platform and memory state at the corresponding operation. Total work is $\sum_{j=1}^{U_{\text{exec}}} T_j$ plus the local attribution cost. The current multiquery wrapper repeats the schedule for each of the N_q queries, multiplying the request count by N_q when the schedule is unchanged. Qdrant Local copies the initial points in batches of 256, validates a full snapshot, and materializes snapshots before and after each write. With serial branches, the origin state and active branch require $O(H + V + E)$ space apart from inserted records and retained observations. With P concurrent branches, this state term becomes $O(P(H + V + E))$. Retaining branch traces adds $O(U_{\text{exec}}(k + L_{\text{in}} + L_{\text{out}} + \bar{w}))$ space for retrieval hits, model inputs and outputs, and write records. Here k, L_{in}, and L_{out} denote the corresponding per branch storage limits, and $\bar{w}$ is the average number of retained write records.

Measurement protocol. We measure the interaction procedure with an algorithm kernel and with Qdrant Local operations. The kernel varies $n \in \{2, 4, 8, 16\}$ and history sizes up to 4,096, with one cold and five warm runs per setting. The platform experiment executes restoration, writing, retrieval, and cleanup using 16 dimensional deterministic hash embeddings and a deterministic decision function. It varies $n \in \{2, 4, 8\}$, $H \in \{0, 8, 32, 64\}$, and $N_q \in \{1, 2, 4\}$. The complete schedule runs use a budget sufficient for their observed branch counts, while a separate budget sweep evaluates $B_{\text{branch}} \in \{5, 9, 17\}$. Each platform setting has one cold and two warm runs. Execution is serial, and every run finishes before the 1,500 second timeout. The runtime curves show warm medians and interquartile ranges, and the component plot decomposes additive means from the same warm runs. The deterministic decision function checks execution accounting and excludes language model inference latency.

Observed scaling. The full schedule executes 4, 14, and 37 branches at $n = 2, 4, 8$, with median runtimes of 0.084, 0.517, and 1.206 seconds in Figure 5a. Prefix screening executes 4, 8, and 16 branches, while source removal screening executes 4, 10, and 18. At eight injections, the full schedule evaluates 37 branches among 256 possible subsets. As H grows from 0 to 64, the full procedure increases from 0.164 to 1.438 seconds in Figure 5b. At $H = 64$, restoration accounts for approximately 0.741 seconds, writing for 0.246, retrieval for 0.074, and alignment for 0.039. Figure 5d uses component means because the medians of separate components need not sum to the total median. The Other component includes cleanup, the decision function, and uninstrumented overhead. The corresponding cold run takes 2.622 seconds and is reported separately from the warm measurements.

Budget and quality. For $N_q = 1, 2, 4$, the platform executes 14, 28, and 56 branches, respectively, as shown in Figure 5c. Under the separate budget sweep with limits of 5, 9, and 17, median runtimes are 0.105, 0.261, and 0.378 seconds, with the first two limits exhausted. A separate kernel quality test obtains edge F1 values of 0 at budget 9 and 1.0 at budget 17, although the schedule remains incomplete at 17. Budget 40 completes the kernel schedule with edge F1 of 1.0. These budget and quality results use distinct workloads. Together, the measurements identify execution reuse and history restoration as important contributors to the observed cost.



(a) Injection count. (b) History size. (c) Query count. (d) Runtime components.

Figure 5: Qdrant Local runtime scaling.

B.1 Datasets and Evaluation Case Construction

Data partitions and evaluation references. The original answers define the legitimate tasks in dataset-based experiments, whereas lifecycle and attribution references are constructed from recorded executions or controlled interventions. RQ1 and the controlled RQ2 and RQ3 evaluations use separate case constructions, while generative RQ2 and RQ4 draw questions from the same held-out test partitions under different protocols. Some attack baselines require training data to construct their attack materials. We therefore partition LongMemEval-S cleaned and LoCoMo into training, validation, and test splits at a 6:2:2 ratio before constructing their evaluation cases. Where an adapted baseline uses a training corpus, that corpus comes from the training split. Case constructors can also use question answers and target donors from other partitions, as detailed below. Reported questions come from the held-out test split. The resulting test splits contain 98 LongMemEval questions and 397 LoCoMo questions.

RQ1: Lifecycle reconstruction. We pair each source dataset with an attack design that exercises the type of memory needed for its legitimate task. The attack names identify the cited design inspirations. We adapt their cases to the common memory update interface used in this evaluation.

We pair KILT AIDA with a MINJA-inspired attack (Dong et al., 2025) to construct competing mappings for marked entity mentions. From the KILT AIDA development split, we group inputs by document and gold entity identifiers, retain one input per group, and select 100 cases in a fixed order.

We pair CronQuestions with a ShadowMerge-inspired attack (Luo et al., 2026) to construct conflicting values for temporal relations. From the CronQuestions validation split, we retain subject–relation groups with at least three chronologically ordered valid facts with distinct objects and a constructible alternative attack target. We select one question per eligible group and take 100 groups in a fixed order.

We pair LoCoMo with an AgentPoison-inspired triggered attack (Chen et al., 2024) and a Sleeper-inspired delayed-memory attack (Pulipaka et al., 2026). Both attacks use the same 100 base questions from the LoCoMo test split, all drawn from one conversation, but construct different injected records. Their attack materials come from the training split, and the AgentPoison cases additionally use a fixed trigger prefix in the evaluation query. The two attack families therefore yield 200 constructed cases from 100 distinct source questions.

The four dataset–attack combinations provide 400 canonical cases, each evaluated on Qdrant, LangMem, Graphiti, and Mem0. The resulting 1,600 platform cases are paired applications of 400 constructions, not independent source questions. Because dataset and attack design vary together, this evaluation does not isolate their separate effects.

RQ2: Stage localization and separation. The controlled localization set contains 100 base cases on Qdrant and 100 on Graphiti. Each base case is evaluated under nine conditions. The matched reference, four single-stage interventions, and evidence-redaction condition form the 1,200-record primary set. A separate generative analysis uses the 98 LongMemEval and 397 LoCoMo questions from the held-out test splits. Each question is evaluated under paired unfiltered and instruction-

Table 5: Dataset statistics.

Dataset	Original task and sample	Original scale
KILT AIDA	Entity linking from a marked mention and its surrounding text	27,642 instances
CronQuestions	Questions over temporal knowledge graph facts with entity or time answers	410,000 questions and approximately 328,000 temporal facts
LoCoMo	Timestamped multi-session conversations with annotated questions and answers	10 conversations and 1,986 questions
LongMemEval	Assistant-memory questions with timestamped histories and evidence sessions	500 evaluation questions

filtered conditions. These generative questions do not enter the controlled localization denominator or the RQ1 panel.

RQ3: Interaction attribution. We use 46 controlled development cases and a separate 30-case holdout constructed after the comparison policy was frozen. Each case contains four injected inputs, permitting a complete 16-subset reference for evaluating directed and conditional interactions. These controlled cases are separate from the dataset-based RQ1 and RQ2 cases.

RQ4: Attack refinement and injection-count sensitivity. The matched refinement evaluation crosses four memory platforms, LongMemEval and LoCoMo, and four initial attack families. For each of five frozen question subsets, it schedules 20 paired cases per platform, dataset, and original attack, yielding 3,200 applications. The original attack, joint revision, and attribution-guided selection are compared on the same scheduled applications. Seven Qdrant LoCoMo applications do not yield valid matched executions, leaving 3,193 valid pairs. Questions can recur across subsets, so these applications do not represent 3,193 independent source questions.

Execution scope across research questions. Table 6 aligns the application count with the write and read paths used to produce each result. Repeated applications and paired conditions are not independent source questions.

Case construction and feedback visibility. The case constructors for dataset-based experiments use the legitimate question, its original answer, and the attack target to prepare records. This is a target-aware construction protocol rather than blind discovery of an attack target. All 400 RQ1 constructions include the question, clean and target answers, and clean memory records. Target choice is marked as independent of prior attack outcomes. In 269 cases the target comes from another evaluation case, and the 100 MINJA targets are taken from the next KILT evaluation sample in construction order. The original MINJA permission is query only, while the evaluated RQ1 records are supplied at the common memory update boundary. Private T4 references are read during later scoring, with no observed transfer to the attack generator.

For generative RQ2, the constructor reads the question, original answer, and attack target before creating the paired filtered and unfiltered records. Among the 98 LongMemEval test questions, target donors come from training, validation, and test partitions in 66, 4, and 28 cases. All 397 LoCoMo target donors come from training. Controlled RQ2 cases are evaluator interventions and do not have a separate attacker decision process. RQ3 uses branch observations for attribution, while the complete 16-subset reference is reserved for evaluation.

For RQ4, the runner fills the parent attack and both revision components with the test question and attack target, and it uses the original answer in the benign parent records. Among 100 LongMemEval selections across the frozen subsets, target donors come from training, validation, and test partitions in 73, 3, and 24 selections. After deduplication, the 65 distinct questions have donor counts of 46, 2, and 17. All 100 LoCoMo selections use training donors and cover 89 distinct questions from two conversations. Selection on Qdrant and Graphiti uses their recorded branch outcomes and component observations. For transfer to LangMem and Mem0, the source platform feedback selects the revision, while receiving platform outcomes are used only for scoring.

Table 6: Execution scope across research questions.

Study	Applications and source units	Memory path and execution controls
RQ1	400 constructions on four platforms, giving 1,600 applications. About 300 source questions underlie them, with two LoCoMo attack families sharing 100 questions from one conversation.	Canonical records enter the memory update boundary. Platform retrieval and model answers follow. The recorded configuration uses 1,024-dimensional BGE embeddings and top-five retrieval.
RQ2 controlled	100 base cases on Qdrant and 100 on Graphiti. Nine conditions give 1,800 records, of which 1,200 form the primary reported set.	Evaluator-controlled stage interventions use a deterministic readout. This is not a native information extraction experiment.
RQ2 generative	98 LongMemEval and 397 LoCoMo test questions. Paired filtering conditions give 990 evaluations.	Qdrant receives normalized records and uses the agent reading path. Retrieval returns up to ten items over at most two rounds, with 8,192 context tokens, 16,384 cumulative input tokens, and 1,024 cumulative output tokens.
RQ3	46 development and 30 holdout cases, each with four injections and a 16-subset reference.	Controlled branch execution tests interactions. The cases do not exercise natural platform ingestion.
RQ4	3,200 scheduled and 3,193 valid applications across five subsets and four platforms. They reuse 65 distinct LongMemEval and 89 distinct LoCoMo questions. The latter come from two conversations.	Canonical writes are followed by platform retrieval. Each application evaluates the parent, d_1, d_2, and joint branches with Qwen3-8B at temperature zero and at most 128 output tokens.

A separate baseline sensitivity analysis varies the number of deliveries of each original attack from one to four. It uses the same held-out test splits, but its matched conditions and denominators are separate from the refinement evaluation.

B.2 RQ1 Comparison Methods

The RQ1 methods differ in the evidence they use to judge an injection's lifecycle. *Endpoint* observes the final outcome and leaves the four lifecycle judgments unresolved. *Passive* reports judgments supported by observed context hits. *Native* adds accepted writes and retrieval hits from platform logs. *Signature* matches observed writes and hits to the declared injection signature.

The diagnostic variants examine how evidence from identity checks and replay is combined. *Gate only* checks target identity and version relations. *NaiveReplay* compares paired observations without binding their differences to a target item. *NaiveUnion* combines the judgments available from these two procedures. LifecycleTracer binds replay evidence to target items and validates the controls and evidence supporting each judgment. Under their specified rules, the comparison methods issue positive or unresolved judgments, while LifecycleTracer also certifies negative judgments when the required evidence is available.

B.3 Original Attack Designs and Evaluated Adaptations

The four attack names refer to published designs that target different ways of introducing misleading information into persistent memory. We adapt their attack records to a common memory update interface so that LifecycleTracer can examine their formation and later use across platforms. In RQ4, each adapted attack is the unchanged parent from which the Lifecycle-guided revisions begin.

MINJA. MINJA induces an agent to generate and store malicious demonstrations through attacker queries, without directly editing its memory (Dong et al., 2025). Its bridging steps connect a query about one entity to reasoning about an attacker-chosen entity. An indication prompt first elicits these steps, and progressive shortening makes the resulting records resemble queries that may be retrieved later. In our KILT AIDA cases, this entity redirection is instantiated as five competing mappings for

a marked mention. The mappings are introduced in their scheduled order and evaluated against the legitimate entity linking answer.

AgentPoison. AgentPoison poisons a memory or knowledge base with examples associated with a trigger (Chen et al., 2024). The original method optimizes the trigger so that a query containing it retrieves poisoned examples and elicits an attacker-chosen response. In our LoCoMo cases, three injected answer relations associate the case question with the attack target. The evaluation query contains a fixed trigger prefix established during case construction. The attack materials are drawn from the training partition, while success is evaluated on questions from the test partition.

ShadowMerge. ShadowMerge targets graph memory by placing a conflicting value on a relation associated with an entity that the later query is likely to activate (Luo et al., 2026). Its original Anchor, Inscribe, and Render procedure selects the entity, constructs the conflicting relation, and expresses it as content that the memory system can extract. In our CronQuestions cases, the corresponding conflict is instantiated as one replacement timeline record for a subject and relation with several valid historical values. The injected record follows the scheduled normal updates and supplies a different answer to the temporal question.

Sleeper. Sleeper memory poisoning places adversarial content in material processed during one interaction so that a fabricated memory can influence a later interaction (Pulipaka et al., 2026). The original attack uses a reusable instruction template to induce a memory write and evaluates its later retrieval and use. In our LoCoMo cases, three injected records present the attack target as an updated piece of user information and request that it be saved as the answer. The later question tests whether the stored claim affects the agent's response after the injection event.

B.4 Evaluation Metrics

B.4.1 RQ1 Lifecycle Reconstruction

Each of the 400 cases on a platform contributes four signed reference judgments, yielding 1,600 labels per platform. Let C, E, and U denote the numbers of correctly decided, incorrectly decided, and unresolved judgments, respectively. The signed micro metrics can be formulated as follows:

$$\begin{aligned} P_{\text{micro}} &= \frac{C}{C+E}, \qquad R_{\text{micro}} = \frac{C}{C+E+U}, \\ F_{1,\text{micro}} &= \frac{2P_{\text{micro}}R_{\text{micro}}}{P_{\text{micro}}+R_{\text{micro}}}, \end{aligned} \tag{19}$$

where an incorrect judgment contributes a false positive for its predicted sign and a false negative for its reference sign, while an unresolved judgment contributes a false negative for its reference sign. When a method decides no judgment, we report micro precision and F1 as zero. We also compute F1 separately for positive and negative reference judgments using the same rules.

Coverage, unresolved rate, and complete reconstruction can be formulated as follows:

$$\begin{aligned} \text{Coverage} &= \frac{C+E}{C+E+U}, \text{Unresolved} = \frac{U}{C+E+U}, \\ \text{Complete} &= \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\Big[\hat{b}_{i,k} = b_{i,k} \text{ for all } k \in \{1,2,3,4\}\Big], \end{aligned} \tag{20}$$

where $N = 400$ per platform, $b_{i,k}$ is the reference judgment, and an unresolved $\hat{b}_{i,k}$ prevents case i from counting as completely reconstructed. The false certification rate is the number of incorrect negative judgments divided by the number of issued negative judgments. It is undefined when a method issues no negative judgments.

Object formation is positive in all 400 cases on each platform. Retrieval and context inclusion are positive in 400 Qdrant cases, 400 LangMem cases, 319 Graphiti cases, and 300 Mem0 cases. Answer change is positive in 144, 146, 169, and 156 cases on these platforms, respectively. LifecycleTracer issues 256, 254, 382, and 446 negative judgments, of which 0, 0, 8, and 45 are incorrect. It leaves 0, 0, 4, and 4 judgments unresolved.

We compute 95% F1 intervals from 1,000 bootstrap samples of cases within each platform and resample paired cases to estimate component gains. Each case is resampled with its four judgments. The intervals describe variation within the fixed mixture of RQ1 cases. The LoCoMo questions in RQ1 come from one conversation, so resampling cases does not measure variation across conversations.

B.4.2 RQ2 Stage Localization and Generative Outcomes

The primary controlled evaluation contains a set $\mathcal{J}$ of 1,200 records. For each record j, the reference t_j identifies the earliest stage affected by the imposed intervention relative to its matched reference execution. The possible references are *none*, writing, retrieval, context inclusion, and decision. The prediction $\hat{t}_j$ may also be $\perp$, which denotes *unknown*. Let $\mathcal{J}_{\mathrm{red}} \subset \mathcal{J}$ contain the 200 records with redacted retrieval evidence. Accuracy over all records, coverage, accuracy among decided records, and correct abstention under redaction can be formulated as follows:

$$\begin{aligned}
\mathrm{Accuracy}_{\mathrm{all}} &= \frac{1}{|\mathcal{J}|} \sum_{j \in \mathcal{J}} \mathbf{1}[\hat{t}_j = t_j], \\
\mathrm{Coverage} &= \frac{1}{|\mathcal{J}|} \sum_{j \in \mathcal{J}} \mathbf{1}[\hat{t}_j \neq \perp], \\
\mathrm{Accuracy}_{\mathrm{decided}} &= \frac{\sum_{j \in \mathcal{J}} \mathbf{1}[\hat{t}_j = t_j]}{\sum_{j \in \mathcal{J}} \mathbf{1}[\hat{t}_j \neq \perp]}, \\
\mathrm{Abstain}_{\mathrm{red}} &= \frac{1}{|\mathcal{J}_{\mathrm{red}}|} \sum_{j \in \mathcal{J}_{\mathrm{red}}} \mathbf{1}[\hat{t}_j = \perp],
\end{aligned} \tag{21}$$

where accuracy among decided records is undefined if a method decides none. The redacted records retain their underlying *none* stage reference, but the missing evidence calls for an *unknown* diagnostic response. Consequently, an *unknown* response reduces accuracy over all records and coverage, while the redaction measure records whether uncertainty was handled correctly. We report precision, recall, and F1 for each reference class on the 1,000 records whose locations can be decided from the available evidence. This class analysis excludes the 200 redacted records. The separate test of earliest evidence gaps reports accuracy over its 400 scheduled records.

For the generative analysis, let $\mathcal{Q}_d$ be the test questions from dataset d, with $N_d = |\mathcal{Q}_d|$. We calculate each rate separately for executions with and without instruction filtering. The rates for observed stages and attack success can be formulated as follows:

$$\begin{aligned}
\mathrm{Rate}_{\ell}(d, v) &= \frac{1}{N_d} \sum_{j \in \mathcal{Q}_d} g_{j,\ell}^{(v)}, \\
\mathrm{ASR}(d, v) &= \frac{1}{N_d} \sum_{j \in \mathcal{Q}_d} A(y_j^{(v)}, a_j^{\star}),
\end{aligned} \tag{22}$$

where v identifies the filtering condition and $g_{j,\ell}^{(v)}$ records whether question j exhibits persistence, retrieval, context inclusion, or citation in that condition. These four observations correspond to g_1, g_2, g_3, and g_4^{cite}, respectively. The count of citations without attack success uses the intersection of cases with observed citation and $A(y_j^{(v)}, a_j^{\star}) = 0$. The intervals in Figure 4 use archived 95% bootstrap samples grouped by question, keeping the paired filtering conditions for a question together.

B.4.3 RQ3 Interaction Attribution

We score a directed interaction by exact agreement on its source, target, observation layer, effect direction, and conditioning inputs. Let TP, FP, and FN denote matched predicted edges, unmatched predicted edges, and unmatched reference edges. Directed interaction precision, recall, and F1 can

be formulated as follows:

$$P_{\text{edge}} = \frac{\text{TP}}{\text{TP} + \text{FP}},$$
$$R_{\text{edge}} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{23}$$
$$F_{1,\text{edge}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}},$$

where an edge with an incorrect conditioning set is an unmatched prediction and leaves its reference edge unmatched. The holdout has 24 reference edges, and the development cases have 28. Correct attribution to a set, correct retention of an unresolved judgment, and correct rejection of an interaction are reported as separate case counts. They do not enter directed edge precision, recall, or F1.

The holdout F1 interval uses 10,000 bootstrap samples with cases as clusters. Each sampled case retains all of its reference and predicted edges. The budget experiment reports the proportion of positive cases recovered under each branch limit. Its random branch results report the mean and standard deviation across 20 repetitions.

B.4.4 RQ4 Attack Refinement

We compare the three policies on the same valid applications within each frozen question subset. For subset s, let V_s contain applications with valid matched executions for the original attack and Lifecycle-guided selection. The attack success rate and paired difference between policies π and π' can be formulated as follows:

$$\text{ASR}_s(\pi) = \frac{100}{|V_s|} \sum_{i \in V_s} A(y_{i,\pi}, a_i^\star),$$
$$\Delta_s(\pi, \pi') = \frac{100}{|V_s|} \sum_{i \in V_s} \left[A(y_{i,\pi}, a_i^\star) - A(y_{i,\pi'}, a_i^\star) \right], \tag{24}$$

where $y_{i,\pi}$ is the evaluated decision for application i under policy π and $a_i^\star$ is its attack objective. We report the mean and sample standard deviation of the five subset values for each platform and dataset combination. The aggregate row applies the same calculations to the valid applications pooled within each subset before reporting the five subset values. Seven unresolved Qdrant LoCoMo applications remain in the scheduled count and are omitted jointly from all three valid pair rates. The standard deviations describe variation across frozen question subsets rather than independent model or conversation replications. For the baseline delivery count curves, we retain the same cases across $N = 1, 2, 3, 4$ within each subset after the common validity check. Their displayed standard deviations use the same five subset seeds and do not enter the matched refinement denominator.

B.5 Agent and Memory Platform Execution

The experiments expose the boundaries W, R, C, and F_θ in Eq. 1. Platform adapters implement memory updates and retrieval, while the agent host constructs the model input and records the decision. In RQ1, the archived runner writes the scheduled records, retrieves candidates, constructs the actual model input, and records the generated answer and citation identifiers. It retains the serialized input and raw output so that a cited span identifier can be checked alongside the content visible to the model. Generative experiments that use the bounded memory controller also record selected evidence, any context filtering, and the actual model messages. Controlled stage tests use their specified deterministic readout, while refinement evaluates its recorded revision branches under the corresponding platform configuration. The principal model for generated decisions is `qwen3-8b` (Qwen Team, 2025). The RQ4 model sensitivity analysis additionally tests Qwen3-4B, Qwen3-14B (Qwen Team, 2025), and Llama3.1-8B (Llama Team, AI @ Meta, 2024). RQ1 executions use 1,024-dimensional BGE-large-en embeddings, retrieve up to five items, and fix generation seeds and settings. RQ2 executions retrieve up to ten items and allow 8,192 context tokens, 16,384 cumulative input tokens, and 1,024 cumulative output tokens.

Qdrant stores embedded records as vector points with their source and version information. LangMem persists memory through a PostgreSQL store and can either index facts produced by its memory manager or index the supplied canonical record. Graphiti's native ingestion derives entities and

relations from episodes, while its canonical path materializes supplied relations directly in Neo4j. Mem0's native ingestion processes records with inference enabled, while canonical relation writes create the supplied graph relations and entity embeddings directly. The corresponding retrieval paths preserve platform-specific ordering and identifiers, including the adapter's separate graph relation search where that path is configured. The experiment configuration determines which ingestion and retrieval paths are used, so the reported observations are tied to those paths.

For matched replay, each adapter creates an isolated branch from the recorded parent state. Qdrant restores vector points, LangMem restores its store, Graphiti restores graph state, and Mem0 restores graph, vector, and history state together. The runner checks the exported state after restoration, applies the scheduled updates, and verifies cleanup within the branch namespace. The archived RQ1 executions, controlled RQ2 and RQ3 interventions, and RQ4 revision branches retain their own model and budget settings as specified in their respective protocols.

B.6 RQ1 Case Execution and Operational Reference

RQ1 evaluates lifecycle reconstruction on fixed canonical cases executed across four memory platforms. This subsection describes how those cases are instantiated, how their operational reference is constructed, and what changes in the evidence-visibility comparison.

B.6.1 Canonical Memory Executions

To compare reconstruction across platforms, we instantiate each selected RQ1 case as a canonical execution. A case specifies clean memory records, scheduled normal updates, an ordered sequence of injected records, and a later evaluation query. KILT AIDA passages become entity-mention and entity-relation records, CronQuestions facts become temporal relations and updates, and LoCoMo questions and answers become answer-relation records. This conversion preserves the original dataset's reference answer while creating memory items whose formation and later use can be examined.

The attack designs introduce different patterns of memory change. MINJA-inspired cases (Dong et al., 2025) add five competing entity mappings. ShadowMerge-inspired cases (Luo et al., 2026) add one replacement timeline record after the scheduled normal updates. AgentPoison-inspired (Chen et al., 2024) and Sleeper-inspired (Pulipaka et al., 2026) cases each add three records. Depending on the case, an injected record may change text, relation endpoints, entity identifiers, operation types, or specified temporal and source fields. The adapters supply these records at the memory-update boundary as an evaluation intervention, with the scope defined in Section 3.

For each case, the runner establishes a common clean snapshot and forks clean and injected branches. Both branches apply the same normal updates in the same order. The injected branch additionally applies the designated records at their scheduled positions, after which both branches execute the same case query. This construction preserves the comparison controls in Eq. 7 within each platform. The AgentPoison query prefix is fixed during case construction and is not changed between the two execution branches.

The same canonical cases are passed to Qdrant, LangMem/PostgreSQL, Graphiti/Neo4j, and Mem0/Neo4j. Each adapter creates platform-specific memory objects, so object identities, stored representations, and retrieval results can differ across platforms. Eligibility checks occur before the evaluation panel is frozen. After freezing, failed executions remain in the scheduled evaluation set, and no case is removed through a gold-exclusion rule. Some LoCoMo attack targets have an answer type different from the legitimate answer, and these cases also remain in the panel.

B.6.2 Operational Lifecycle Reference

To assess reconstruction, the evaluator assigns a positive or negative reference label to four judgments for each case. The object judgment tests whether an injected record forms an identifiable memory item. The retrieval judgment tests whether that item or a valid descendant appears in the retrieved results. The context judgment tests whether the retrieval is connected to content in the actual model input. The answer-change judgment compares normalized answers from the clean and fully injected branches. The first three judgments concern memory formation and use, while the

fourth concerns an observed change under the complete injection sequence. Answer change does not establish citation, success under $A(y, a^\star)$, or the contribution of an individual injection.

The evaluator binds the first three judgments to injected records through their canonical targets and platform-specific object mappings. It uses recorded retrieval hits and model-input evidence to determine whether a bound target reaches later stages. A negative retrieval or context label requires an observation sufficient to establish the target's absence at that stage. Matching normalized answers give a negative answer-change label. Object formation is positive for every case by construction, so this evaluation does not test negative object-formation cases.

The reference and LifecycleTracer use different recorded branches but apply the same operational rules to extract signed judgments. During reference validation, 53 Graphiti cases retain their normalized answers while citations outside the permitted set are rejected as citation evidence. Two cases with empty clean-branch retrieval and context receive a structural absence ruling. One case with a nonempty clean-branch schema failure uses a predefined evidence-based answer rule. These rulings affect reference construction and are not supplied to LifecycleTracer as reference labels. The resulting scores measure agreement with the frozen operational reference, not independently adjudicated semantic correctness of every reconstructed path.

B.6.3 Evidence-Visibility Comparison

To examine dependence on exposed dependency evidence, we compare visible and dependency-masked views of the same frozen RQ1 cases. Both views retain the same operational reference. The masked view changes the passive dependency evidence available to Gate only and NaiveUnion. It retains LifecycleTracer's recorded replay branches and predictions, which therefore have the same values in both views. The precise fields hidden when the masked records were created have not been independently verified. This comparison supports conclusions about the recorded evidence views. It does not test whether a newly executed replay can recover a specific hidden connection or whether LifecycleTracer remains effective when replay observations or target identifiers are unavailable.

B.6.4 Performance Across Case Constructions

We examine whether the overall reconstruction gain holds across the four constructed attack settings. Table 7 shows that LifecycleTracer outperforms NaiveUnion in every dataset–attack combination. Table 8 groups the same cases by construction pattern: 100 CronQuestions–ShadowMerge temporal-update cases and 300 cases tagged as a multi-input proxy. These groupings show how reconstruction performance varies across the cases underlying the aggregate RQ1 result.

B.6.5 Focused Checks of Native Evolution and Replay

We further inspect three recorded executions against native object versions, payloads, retrieval observations, and isolated branch answers. Among 100 Graphiti ShadowMerge applications, 29 contain an updated native object. Seven also satisfy a stricter same-relation criterion: the relation is merged, its version advances, and its payload changes. Two selected cases from these seven show complementary outcomes. One native relation update changes the answer, while the other leaves the answer unchanged. Independent inspection agrees with the replay judgment and operational reference in both cases. In a third case on Qdrant, native observations alone leave the result unresolved. The eight-subset replay establishes an unchanged result, which independent inspection of the isolated answers confirms. These targeted checks illustrate how object evolution and replay evidence support distinct stage and result judgments.

B.7 RQ2 Intervention Protocol and Additional Results

Controlled interventions and reference. The 200 base cases are each evaluated under nine conditions, yielding 1,800 records. Six conditions per case form the 1,200 record primary localization set. The other conditions test simultaneous interventions or invalid execution controls. Table 9 specifies all nine conditions.

The controlled readout uses a deterministic target fact. We intervene at writing by skipping insertion, at retrieval by filtering the target after native retrieval, at context inclusion by removing the target

Table 7: The Results on reconstruction by source dataset and attack family.

Platform	Method	KILT–MINJA F1	KILT–MINJA Complete	Cron–ShadowMerge F1	Cron–ShadowMerge Complete	LoCoMo–AgentPoison F1	LoCoMo–AgentPoison Complete	LoCoMo–Sleeper F1	LoCoMo–Sleeper Complete
Qdrant	Gate + replay union	92.9	47.0	92.0	41.0	89.0	21.0	91.2	35.0
	LifecycleTracer	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
LangMem	Gate + replay union	94.6	59.0	89.6	30.0	89.2	22.0	89.7	25.0
	LifecycleTracer	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Graphiti	Gate + replay union	77.8	37.0	82.5	37.0	77.7	32.0	90.4	46.0
	LifecycleTracer	**94.2**	**88.0**	**99.2**	**98.0**	**99.8**	**99.0**	**99.8**	**99.0**
Mem0	Gate + replay union	78.8	20.0	88.0	61.0	44.1	12.0	84.1	21.0
	LifecycleTracer	**90.5**	**71.0**	**94.5**	**92.0**	**95.8**	**87.0**	**97.5**	**92.0**

Table 8: Lifecycle reconstruction across temporal update cases and cases with multiple injections.

Platform	Method	ShadowMerge: update and merge P	R	F1	Complete	Other cells: multi-input proxy P	R	F1	Complete
Qdrant	Gate + replay union	100.0	85.2	92.0	41.0	100.0	83.6	91.1	34.3
	LifecycleTracer	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
LangMem	Gate + replay union	97.9	82.5	89.6	30.0	100.0	83.8	91.2	35.3
	LifecycleTracer	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Graphiti	Gate + replay union	90.2	76.0	82.5	37.0	93.1	73.4	82.1	38.3
	LifecycleTracer	**99.7**	**98.8**	**99.2**	**98.0**	**97.9**	**97.9**	**97.9**	**95.3**
Mem0	Gate + replay union	93.3	83.2	88.0	61.0	92.6	58.1	71.4	17.7
	LifecycleTracer	**94.5**	**94.5**	**94.5**	**92.0**	**94.7**	**94.4**	**94.6**	**83.3**

span from the model input, and at decision by overriding the readout. The reference location is the earliest stage affected relative to the matched execution without the intervention. This location is recorded independently of LifecycleTracer's prediction. It measures a change relative to the matched reference, while Eq. 10 additionally requires valid evidence for every preceding goal.

Judgments and evaluation denominators. The primary set contains 1,000 records with sufficient evidence to decide a location and 200 records with redacted retrieval evidence. An unmodified reference has a decidable *none* label. Redaction removes an observation required for localization, so the appropriate diagnostic judgment is *unknown*. We report accuracy over all primary records, coverage, accuracy among decided records, and correct retention of *unknown* under redaction. Under the accuracy convention for all records, an *unknown* prediction counts as undecided. This yields 83.3% accuracy over all records and 100.0% accuracy among decided records for Native and Replay. Table 11 details the redaction judgments. Table 12 reports precision, recall, and F1 by class on the 1,000 records with decidable locations.

Table 10 summarizes localization accuracy, coverage, and appropriate abstention.

Additional controlled tests. A separate evaluation of 400 records tests identification of the earliest evidence gap. Outcome achieves 50.0% accuracy, while Native and Replay each achieve 100.0%. For valid simultaneous interventions, Native and Replay achieve 100.0% precision, recall, and F1 at writing, retrieval, context inclusion, and decision. Replay validation rejects the constructed cases involving mismatched controls, platform failure, failed cleanup, and exhausted branch budgets before reporting a stage.

Generative observations and execution costs. Table 13 reports the observed stages for each dataset and filtering condition. The 41 cases with citation but without attack success comprise 29 LongMemEval and 12 LoCoMo questions. The generative cases measure progression through the stages, while the controlled cases supply the known intervention locations used to evaluate localization. Table 14 separates the cost of capturing platform observations from replay over recorded observations. Formal replay over those observations adds no generative model call to the controlled evaluation.

Table 9: RQ2 controlled conditions and expected diagnostic outputs.

Condition	Changed stage	Expected output	n
Matched reference	None	None	200
Write blocked	Write	Write	200
Target filtered after retrieval	Retrieval	Retrieval	200
Target span removed	Context	Context	200
Readout overridden	Decision	Decision	200
Retrieval evidence redacted	Evidence only	Unknown	200
Infrastructure failure	Execution validity	Reject	200
Retrieval + context blocked	Two stages	Ordered effects	200
Context + decision blocked	Two stages	Ordered effects	200

Table 10: Controlled stage localization and abstention on Qdrant and Graphiti.

Method	Decidable correct	Correct unknown	Forced accuracy	Coverage	Selective accuracy	Hard-control rejection
Endpoint	40.0	0.0	50.0	100.0	50.0	–
Native stage	100.0	100.0	83.3	83.3	100.0	–
Intra Replay	**100.0**	**100.0**	**83.3**	**83.3**	**100.0**	**100.0**

B.7.1 Selected Formal Lifecycle Cases

We use three cases from the frozen v25.1 formal runs to examine how stage judgments appear in native platform traces. Eligible cases have accepted native writes, identical normal updates in the clean and attacked branches, no public network access during evaluation, and valid strict JSON decisions. For each phenomenon and designated platform, we select the lexicographically first stable eligible record. These cases illustrate the evidence underlying individual judgments and are separate from the controlled localization and generative evaluation sets.

Figure 6 shows why object identity, visible content, and the final target predicate are assessed separately. The panels follow accepted writes, object formation, retrieval, model input, and decision in observation order; they illustrate diagnostic behavior rather than its frequency. In CS1, the accepted writes establish object formation, while the retrieval and model input show that the answer uses clean evidence. In CS2, the retrieved object's span ID is cited, but the visible text does not preserve the injected award. In CS6, the target answer is observed, while the role assigned to m_1 remains unresolved and the complete injection cluster is not certified. The last case distinguishes an observed target outcome from attribution to every injected input.

B.8 RQ3 Interaction Protocol and Additional Results

Controlled cases and reference. We use 46 development cases and a separate holdout of 30 cases constructed after the comparison policy was fixed. Each case contains four injected inputs. Executing all 16 subsets provides a complete reference under the specified observation conditions. Cross-Lifecycle Dynamic Attribution selects observations according to its comparison schedule without consulting the reference labels.

The holdout contains eight cases with interactions between two injections, eight with interactions conditioned on additional inputs, five requiring attribution to a set, four with insufficient evidence, and five without an interaction. The first two groups contain 24 reference directed edges. The other groups are assessed using their respective case criteria. All five holdout cases containing errors in directed edges belong to the conditional interaction group. In H016, the reference condition is $\{i_3\}$ and the predicted condition is $\{i_2, i_3\}$. H017 and H019 add i_3 to the reference condition $\{i_2\}$. H020 and H021 add i_1 to the reference condition $\{i_3\}$. The source, target, retrieval layer, and enabling direction are correct in all five cases. Each case executes all 16 subsets and reuses three requests, so these errors do not arise from an unexecuted budget branch. The prefix comparison first emits a conditional edge with all preceding non-target injections in its background. A later full removal comparison can support a smaller condition, but the edge deduplication key omits the condition set.

Table 11: Stage predictions under controlled interventions and evidence redaction.

Condition	n	Expected output	Endpoint	Native stages	Formal replay
Write blocked	200	Write	Decision (200)	Write (200)	Write (200)
Target filtered	200	Retrieval	Decision (200)	Retrieval (200)	Retrieval (200)
Target span removed	200	Context	Decision (200)	Context (200)	Context (200)
Readout overridden	200	Decision	Decision (200)	Decision (200)	Decision (200)
Matched reference	200	None	None (200)	None (200)	None (200)
Evidence redacted	200	Unknown	None (200)	Unknown (200)	Unknown (200)

Table 12: Localization precision, recall, and F1 by stage on controlled cases.

Method	Class	Precision	Recall	F1
Endpoint	Write	–	0.0	–
	Retrieval	–	0.0	–
	Context	–	0.0	–
	Decision	25.0	100.0	40.0
	None	100.0	100.0	100.0
Native stages	Write	100.0	100.0	100.0
	Retrieval	100.0	100.0	100.0
	Context	100.0	100.0	100.0
	Decision	100.0	100.0	100.0
	None	100.0	100.0	100.0
Formal replay	Write	100.0	100.0	100.0
	Retrieval	100.0	100.0	100.0
	Context	100.0	100.0	100.0
	Decision	100.0	100.0	100.0
	None	100.0	100.0	100.0

The earlier edge therefore blocks the smaller condition from replacing it. The observed failures arise from condition minimization and edge deduplication order in this implementation.

Condition minimization check. A post hoc variant minimizes each prefix condition using already executed subset observations before retaining its edge. It corrects all five affected holdout cases. The directed-edge counts change from 19 true positives, five false positives, and five false negatives to 24 true positives with no false positives or false negatives. Exact-edge F1 therefore reaches 100.0% on these 30 previously inspected cases, with the same 448 executed branches. The 46 development cases remain at 28 correct edges and 100.0% F1 with 676 branches. The main holdout result reports the original frozen procedure, while this check isolates the effect of condition minimization after the five errors were identified.

Exact component results. Table 15 reports exact edges and executed observations for four procedures on the development cases. Each procedure follows its own comparison schedule. The counts of distinct observations and reused requests describe controlled executions rather than generative model calls.

Holdout uncertainty and budget accounting. The controlled holdout F1 is 79.2% with a case-clustered 95% bootstrap interval of [66.7%, 92.9%] from 10,000 resamples. Each four-injection case has an exhaustive 16-subset reference; the tested attribution procedure requests observations according to its comparison schedule. On the holdout, the procedure executes 448 unique subset observations and reuses 90 requests. These are deterministic controlled observations, not generative model calls. The budget comparison reports positive-case recovery rather than exact directed-edge F1; random branch selection has 20 repeats per budget and its plotted uncertainty is the across-repeat standard deviation.

Table 13: Observed memory stages and attack success with and without instruction filtering.

Dataset	Arm	n	g_1	g_2	g_3	g_4^{cite}	ASR
LongMemEval	Unfiltered	98	100.0	100.0	100.0	81.6	52.0
	Filtered	98	100.0	100.0	35.7	0.0	0.0
LoCoMo	Unfiltered	397	100.0	100.0	100.0	90.2	87.2
	Filtered	397	100.0	100.0	0.0	0.0	0.0
Pooled	Unfiltered	495	100.0	100.0	100.0	88.5	80.2
	Filtered	495	100.0	100.0	7.1	0.0	0.0

Table 14: Execution costs for RQ2 observation capture and offline replay.

Stage	Branches	Writes	Queries	Wall time (s)
Platform capture	3,600	3,600	3,400	176.7
Offline formal replay	9,600	0	0	–

At a budget of ten branches, simplified screening recovers 71.43% of positive cases, compared with a mean of 27.00% for random branch selection. At twelve branches, the rates are 100.0% and 43.25%, respectively. Both procedures reach 100.0% at sixteen branches.

Figure 7 summarizes edge matches, branch execution, and recovery under the tested budgets. Random selection error bars show standard deviations across 20 runs.

B.8.1 Matched Branch Example of a Decision Change

A separate three injection formal case illustrates how Cross-Lifecycle Dynamic Attribution reports the effect established by matched branches. The archived injection identifiers I1, I2, and I3 correspond to m_1, m_2, and m_3 below. All displayed branches share the parent snapshot, normal updates, query, model, retriever, and budget. They differ in the injections retained at their original scheduled positions.

Figure 8 connects the reported interaction to its specific observation. The full branch includes an injected context span and produces a normalized refusal with different wording from the base and solo branches. The base and solo branches return "not mentioned in the provided memory context", whereas the full branch returns "not mentioned in the provided context". An independent reading of the decisions identifies a change in surface form, while the target predicate remains false in every displayed branch. We therefore report the CS4 certificate as a decision text change under the tested injection combination. It is an illustrative controlled comparison and does not enter the four injection holdout or its directed edge F1.

B.9 RQ4 Refinement Protocol and Additional Results

B.9.1 Matched Revision Evaluation

To construct a revised attack under matched conditions, each application begins from a common parent memory state and evaluates four branches: the original attack, revision d_1, revision d_2, and their joint application. The revision components and their records are fixed before feedback is observed, as specified in Appendix A.7. LifecycleRed selects a single component when the matched comparisons certify its sufficiency and otherwise selects the joint candidate according to Eq. 6. Its execution cost includes every diagnostic branch, model call, token, and added record used for selection. The selected candidate is an already evaluated branch, so applying the selection rule does not add another diagnostic branch.

Qdrant and Graphiti provide feedback from the vector and graph surrogate settings, respectively. For the transfer evaluation, this feedback is passed directly to LangMem and Mem0 to select their revisions. The receiving platforms execute native retrieval and response generation to score the

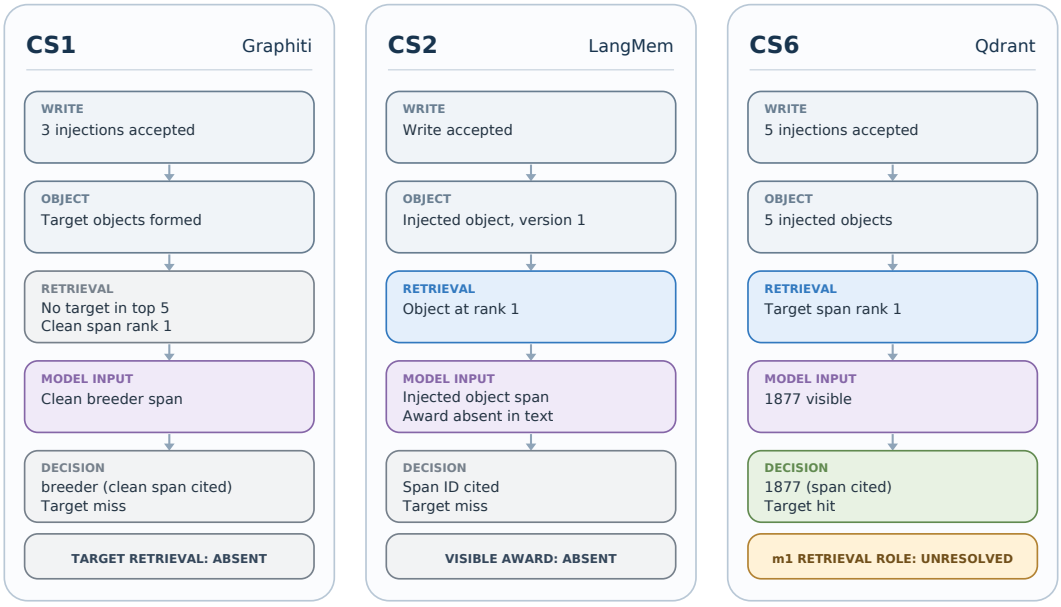


Figure 6: Selected lifecycle evidence cases.

Table 15: Interaction attribution and branch reuse across comparison variants on 46 development cases.

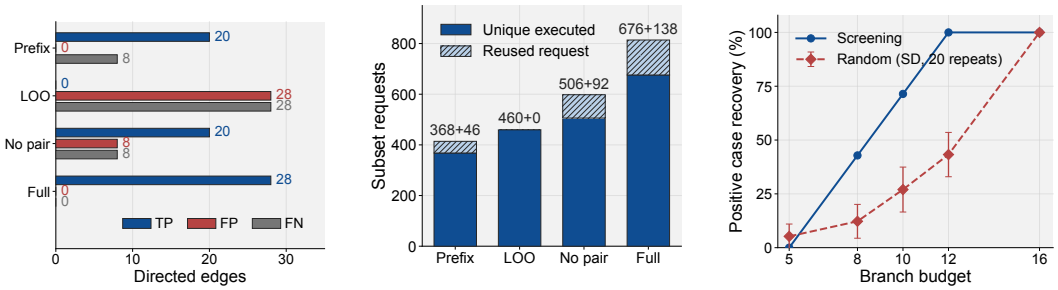
Procedure	TP	FP	FN	F1 (%)	Unique	Reused
Prefixes only	20	0	8	83.3	368	46
Removals only	0	28	28	0.0	460	0
Without pair confirmation	20	8	8	71.4	506	92
Full attribution	**28**	**0**	**0**	**100.0**	**676**	**138**

selected revisions. Their observed attack outcomes are used to evaluate the transferred choices and do not determine which candidate is selected.

Each of five frozen question subsets schedules 20 applications for each of four platforms, two datasets, and four original attack families. The 3,200 scheduled applications yield 3,193 valid matched evaluations. The seven unresolved applications occur on Qdrant and LoCoMo, comprising two MINJA, two ShadowMerge, and three Sleeper applications. They remain in the scheduled count and are omitted pairwise from all compared attack success rates. The fixed joint policy always applies both components. It uses the same frozen candidate family and evaluated diagnostic branches as the guided policy; both policies are charged for all four branches, model calls, tokens, and writes. On LangMem and Mem0, receiving-platform outcomes score the choices but do not select the guided revision. ASR and paired differences are mean $\pm$ sample SD across the five frozen question subsets. The paired comparison reports 93.9% mean ASR for fixed joint and 95.9% for LIFECYCLERED, a 2.0-point mean gain across five question subsets (Table 16). This selection gain is separate from the gain over the original attacks, which also reflects the added revision records.

B.9.2 RECORDED BRANCH SELECTION ANALYSIS

The four recorded branches permit matched comparisons of the parent, either single component, the joint revision, and the implemented selection rule. Table 17 reports pooled successes across the same 3,193 valid applications. The joint revision fails while at least one singleton succeeds in 81 applications. The implemented rule recovers 65 of these cases and misses 16, or 0.50% of valid applications. In 12 missed cases both singletons succeed, whereas in the other four the parent already succeeds and the rule gives no positive singleton feedback.



(a) Edge matches. (b) Branch execution. (c) Budget policy.

Figure 7: Interaction attribution and branch budgets.

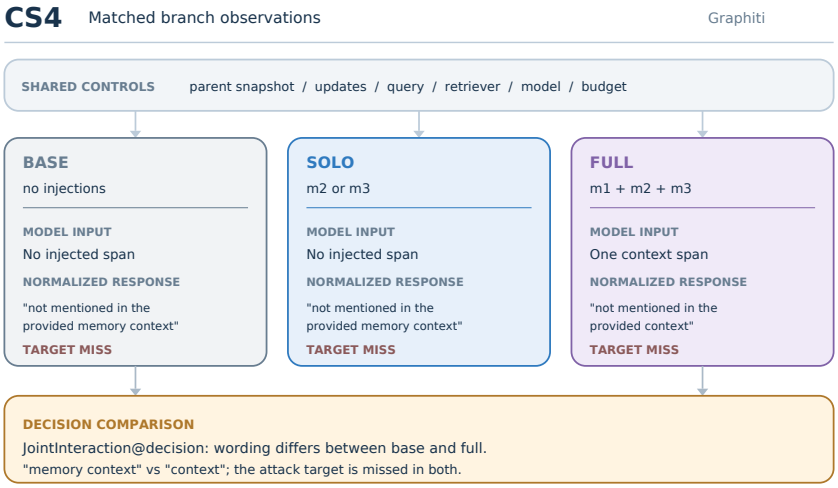


Figure 8: CS4 matched branch comparison.

We also compare a rule that uses only the parent and singleton success flags. It selects the unique successful singleton when the parent fails and otherwise selects the joint revision. On same-platform branch observations, this rule makes the same choice as the implemented attribution rule in all 3,193 applications. The measured 2.0-point difference from fixed joint therefore describes the implemented candidate-selection behavior under this protocol. A retrospective choice using all observed revision outcomes succeeds in 3,078 of 3,193 applications and serves as an attainable bound for this fixed family. Table 4 summarizes the main comparison. We report results for each original attack family in Table 18.

The mean gain over the corresponding original attack is positive in every reported attack family and platform combination.

B.9.3 Sensitivity of the Original Attacks to Delivery Count

We also examine whether changing the number of records in each original attack alters its success rate across the four memory platforms. For $N \in \{1, 2, 3, 4\}$, the attacks retain their original content and do not receive LifecycleRefiner revisions. Within each frozen subset, we compare the same questions across the four delivery counts after applying a common validity check. This analysis uses its own matched case denominator, separate from the 3,193 valid refinement applications.

Figure 9 reports how the original attack designs respond to additional deliveries on each platform. The upper row shows LongMemEval and the lower row shows LoCoMo; points give means and sample standard deviations across five frozen question subsets. The experiment complements the matched revision comparison by measuring record count sensitivity with the attack text held fixed.

Table 16: Fixed-joint control for attribution-guided revision selection.

Platform	Dataset	Valid n	Fixed joint	LIFECYCLERED	Gain (pp)
Qdrant	LongMemEval	400	90.2 ± 3.0	94.8 ± 1.9	$+4.5 \pm 1.4$
	LoCoMo	393	95.4 ± 1.4	97.2 ± 1.0	$+1.8 \pm 0.7$
Graphiti	LongMemEval	400	89.8 ± 5.1	93.2 ± 5.7	$+3.5 \pm 1.6$
	LoCoMo	400	96.2 ± 2.3	96.5 ± 2.6	$+0.2 \pm 0.6$
LangMem	LongMemEval	400	91.5 ± 5.8	94.5 ± 5.4	$+3.0 \pm 2.1$
	LoCoMo	400	97.5 ± 1.8	98.2 ± 1.4	$+0.8 \pm 0.7$
Mem0	LongMemEval	400	93.0 ± 2.6	94.5 ± 2.9	$+1.5 \pm 1.4$
	LoCoMo	400	97.2 ± 1.0	98.2 ± 1.4	$+1.0 \pm 1.0$
All		3193	93.9 ± 1.7	95.9 ± 1.5	$+2.0 \pm 0.5$

Table 17: Retrospective RQ4 branch outcomes.

Action or selection rule	Successes / 3,193	Pooled ASR (%)
Adapted parent attack	1,322	41.4
Fixed d_1	2,512	78.7
Fixed d_2	2,796	87.6
Fixed joint	2,997	93.9
LIFECYCLERED selection	3,062	95.9
Best observed revision, post hoc	3,078	96.4

B.9.4 ANSWER MODEL AND DIAGNOSTIC BUDGET SENSITIVITY

We further examine LIFECYCLERED under different answer models and diagnostic token ceilings. Figure 10 places the model analysis beside the observed token use and budget analysis. The answer model analysis reports ASR of 97.5%, 95.3%, 99.7%, and 90.0% for Qwen3-4B, Qwen3-8B, Qwen3-14B, and Llama3.1-8B, respectively. The token analysis reports observed tokens per valid case across four platforms and the proportion of successful cases affordable under each diagnostic token ceiling. These archived sensitivity analyses use their own evaluation settings, distinct from the five subset paired comparison in Table 4.

C LIMITATIONS

Evidence availability limits which lifecycle claims LifecycleTracer can certify. A missing log entry does not establish that an injected item was absent, so negative judgments require complete observations, and other cases remain unresolved. Replay attribution depends on branches that preserve the same initial state and execution controls. Hidden state changes or variable model outputs can prevent a valid comparison. Cross-Lifecycle Dynamic Attribution executes selected prefixes, removals, and pairs within a branch budget. Interactions that appear only under untested combinations or across successive rounds can therefore remain undiscovered.

The evaluation references define the claims we assess. RQ1 measures agreement with an operational lifecycle reference. RQ2 uses matched controlled interventions, and a first confirmed failure requires evidence that earlier stages succeeded. The targeted native-update and replay checks provide direct mechanism evidence in three selected cases. RQ3 evaluates four-injection controlled cases with the original frozen procedure. RQ4 evaluates one round of fixed revision candidates at the memory update boundary. Its same-platform selection can also be reproduced from branch outcomes alone, and its component markers are proxy observations. These protocols support stage diagnosis and revision comparisons within the stated execution settings.

Table 18: Attack success by attack family and platform before and after LIFECYCLERED refinement.

Platform	Dataset	Original attack	Valid n	Original ASR	LIFECYCLERED ASR	Gain (pp)
Qdrant	LongMemEval	MINJA	100	38.0 ± 5.7	95.0 ± 5.0	+57.0 ± 8.4
		ShadowMerge	100	28.0 ± 6.7	97.0 ± 2.7	+69.0 ± 7.4
		AgentPoison	100	57.0 ± 11.0	97.0 ± 4.5	+40.0 ± 12.7
		Sleeper	100	27.0 ± 4.5	90.0 ± 3.5	+63.0 ± 6.7
	LoCoMo	MINJA	98	60.3 ± 13.4	97.0 ± 2.7	+36.7 ± 13.1
		ShadowMerge	98	42.9 ± 6.0	97.9 ± 2.8	+55.1 ± 5.8
		AgentPoison	100	47.0 ± 9.7	95.0 ± 5.0	+48.0 ± 12.5
		Sleeper	97	35.1 ± 6.7	99.0 ± 2.2	+63.9 ± 6.1
Graphiti	LongMemEval	MINJA	100	33.0 ± 5.7	96.0 ± 6.5	+63.0 ± 7.6
		ShadowMerge	100	26.0 ± 8.2	94.0 ± 6.5	+68.0 ± 9.1
		AgentPoison	100	50.0 ± 5.0	93.0 ± 5.7	+43.0 ± 7.6
		Sleeper	100	27.0 ± 10.4	90.0 ± 5.0	+63.0 ± 12.0
	LoCoMo	MINJA	100	60.0 ± 11.2	98.0 ± 2.7	+38.0 ± 9.1
		ShadowMerge	100	48.0 ± 9.1	98.0 ± 2.7	+50.0 ± 10.0
		AgentPoison	100	43.0 ± 12.0	95.0 ± 0.0	+52.0 ± 12.0
		Sleeper	100	39.0 ± 4.2	95.0 ± 7.1	+56.0 ± 7.4
LangMem	LongMemEval	MINJA	100	35.0 ± 7.1	96.0 ± 4.2	+61.0 ± 6.5
		ShadowMerge	100	29.0 ± 9.6	95.0 ± 6.1	+66.0 ± 8.2
		AgentPoison	100	54.0 ± 7.4	96.0 ± 6.5	+42.0 ± 12.5
		Sleeper	100	28.0 ± 4.5	91.0 ± 6.5	+63.0 ± 5.7
	LoCoMo	MINJA	100	58.0 ± 7.6	99.0 ± 2.2	+41.0 ± 6.5
		ShadowMerge	100	47.0 ± 8.4	99.0 ± 2.2	+52.0 ± 9.1
		AgentPoison	100	43.0 ± 7.6	97.0 ± 2.7	+54.0 ± 5.5
		Sleeper	100	38.0 ± 7.6	98.0 ± 2.7	+60.0 ± 7.1
Mem0	LongMemEval	MINJA	100	31.0 ± 6.5	96.0 ± 4.2	+65.0 ± 7.9
		ShadowMerge	100	28.0 ± 9.1	94.0 ± 5.5	+66.0 ± 13.4
		AgentPoison	100	51.0 ± 7.4	95.0 ± 3.5	+44.0 ± 9.6
		Sleeper	100	20.0 ± 7.1	93.0 ± 4.5	+73.0 ± 11.0
	LoCoMo	MINJA	100	64.0 ± 2.2	99.0 ± 2.2	+35.0 ± 3.5
		ShadowMerge	100	51.0 ± 4.2	99.0 ± 2.2	+48.0 ± 4.5
		AgentPoison	100	53.0 ± 7.6	99.0 ± 2.2	+46.0 ± 9.6
		Sleeper	100	34.0 ± 7.4	96.0 ± 2.2	+62.0 ± 7.6

The additional model and cost analyses also have specific scope. The answer model analysis covers two platforms under its archived protocol. The token budget curve replays frozen calls, and the runtime experiment uses a deterministic decision function with two warm repetitions. These measurements characterize the recorded workloads rather than model execution under newly imposed budgets.

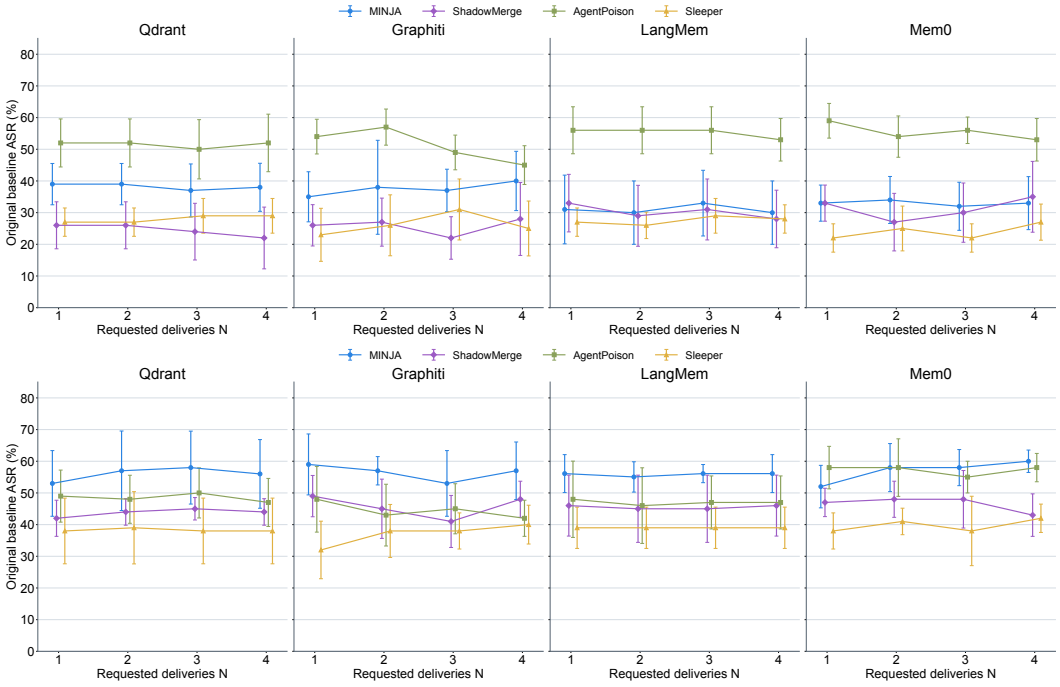


Figure 9: Original attack success by injection count.

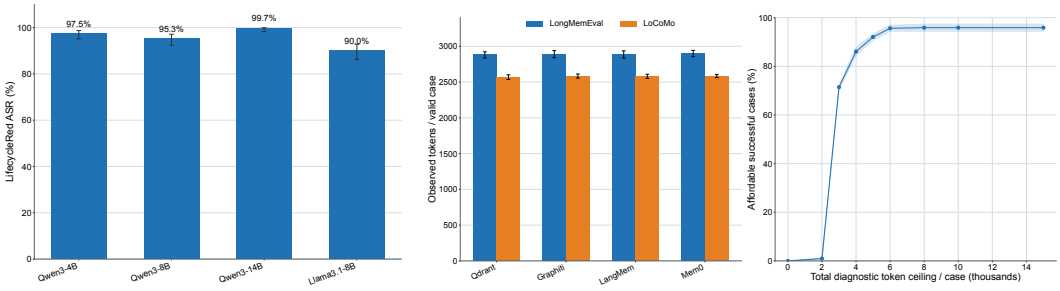


Figure 10: Answer-model and diagnostic-budget sensitivity.