
The Price of Safety: Benign-Case Utility and Token Overhead of Memory-Poisoning Defenses in LLM Agents

Pritom Bhowmik
pritimhowmik@ieee.org

Abstract

Memory-poisoning defenses for LLM agents are typically evaluated by their ability to prevent attacks. However, the traffic they process is rarely adversarial. The cost of implementing a defense is paid with each interaction, while its benefits are only seen in a small percentage of cases. We developed a measurement setup that keeps the memory backend, retrieval process, and judge consistent across different conditions, changing only the defense itself. We test each condition three times across five conversations to distinguish the defense’s real effects from noise inherent in the pipeline’s runs, which remains significant even at temperature zero. Across three write-time defenses (input sanitization, provenance checking, and LLM-based anomaly detection) and one read-time defense (reranking), tested on entirely benign traffic, the write-time defenses show no utility cost we can resolve, with 95% confidence intervals spanning roughly ± 4.5 points and including zero. The reranker is different: it lowers core accuracy by 4.4 points (95% CI [-9.0, -0.05], bootstrap; McNemar $p=0.064$), a result that survives replication but sits at the edge of our resolution. Its clearer cost is mechanical rather than statistical. On conversations containing no attack, the reranker quarantines legitimate memories on 33.6% of adjudicated items, reaching as many as 106 false quarantines in a single conversation, at 2.7% token overhead. Stacking all four defenses does not compound this cost: the combined condition’s accuracy loss is smaller, and its confidence interval includes zero, suggesting the write-time defenses may partly offset what the reranker discards. Where a defense intercepts the pipeline, not whether it uses an LLM, appears to determine its benign-case price.

1 Introduction

Almost none of an LLM agent’s traffic is an attack, yet a memory defense runs on all of it; writing on nearly every turn, reading on nearly every turn after. The cost is paid continuously, but the benefit is realized almost never. Evaluation practice does not reflect this: papers report attack success rate on adversarial conversations and stop there. What a defense costs in ordinary traffic is rarely measured, and without that number, a system designer cannot tell whether a defense that blocks most attacks but quietly degrades ordinary interactions is worth deploying.

We measure that cost directly, across four defenses: sanitization, provenance checking, LLM-based anomaly detection, and retrieval-time reranking. All four run against the same identical undefended baseline on the same memory backend, retrieval configuration, and frozen judge. So, any difference is attributable to the defense alone. All conditions run on entirely benign LoCoMo conversations, in which the defense’s cost is paid but its benefit cannot be realized, even in principle. One design choice proved essential: identical conversations through the same pipeline at temperature zero did not yield identical results, so we replicate each condition three times rather than trusting single runs. The

Code: <https://github.com/pritom02bh/memdefense>

decision mattered: one apparent defense effect turned out, upon inspection, to be a scoring artifact, not a real cost.

Our results are categorized by the pipeline stage at which a defense is applied, rather than by whether it employs an LLM. Write-time defenses do not show any benign-case cost distinguishable from noise, whereas the read-time reranker incurs both a borderline but replicated accuracy cost (4.4 points) and a mechanical cost (33.6% false quarantine rate on attack-free traffic). Interestingly, stacking all four defenses does not increase this cost. Our contributions include (1) a replicated measurement framework that isolates benign-case defense costs in LLM agents, available with the paper; (2) the first known comparison of write- and read-time defenses under the same conditions focusing on benign cases; (3) an example where a single evaluation run suggested a false effect, emphasizing the need for replication at this scale; and (4) a false quarantine rate measurement demonstrating that a defense can impact a large portion of legitimate memory, even when its accuracy change appears minimal.

2 Related Work

2.1 Memory Poisoning in LLM Agents

Persistent memory allows agents to retain information across multiple sessions, providing attackers with a persistent foothold that can outlast individual interactions. As a result, content added to memory during one session may reappear later without the agent suspecting it [1, 2]. This is partly due to the structural nature of retrieval, which compares similarity rather than verifying trustworthiness. Hence, a convincingly fabricated memory can be ranked equally with a legitimate one, as there are no criteria to differentiate them [3, 4]. Recent research further elaborates on the attack surface, highlighting activities at writing, retrieval, and action stages [5, 6]. Additional studies examine the trust boundary, pinpointing when untrusted text becomes integrated into the agent’s record [7, 8].

2.2 Proposed Defenses

Most proposed defenses are typically categorized by the stage at which they intervene. Write-time methods act before a memory is stored, with some sanitizing incoming content [9, 10] and others attaching provenance and weighting memories based on the trustworthiness of their sources [11]. A third type screens for anomalies prior to commitment, occasionally utilizing an LLM as the screening tool [12]. Read-time strategies do not alter storage; instead, they rerank retrieved candidates based on trust or consistency signals before incorporating them into the model’s context [13]. A smaller body of research intervenes even later, at act time, filtering a memory just before it affects a decision [14]. Some systems employ a combination of these stages rather than relying solely on one approach [15].

2.3 The Attack-Success-Rate Gap

The evaluation pattern in this work is consistent: construct an attack, test if the defense prevents it, and report the decline in attack success rate [14]. This approach is the natural first step, and the field has addressed it quite well. However, it overlooks what the defense does when there is no attack, most of the time in deployment. The systems literature complements this by focusing on benchmarking memory architectures on accuracy, latency, and token cost [16, 17] often without considering any defense layered on top [18, 19]. Neither approach is wrong; they simply do not intersect. The critical value for deployment, what a defense costs when not actively catching attacks, lies between these perspectives. This paper aims to measure that specific number.

3 Threat Model and Defenses

3.1 Pipeline

The agent we analyze has the typical shape of most memory-augmented systems. A conversation unfolds one turn at a time. Each turn undergoes an extraction process that determines what, if anything, should be remembered and records this information in a vector store. When a question arises, relevant memories are retrieved from the stored data, and the top matches are added to the

context, allowing the model to generate a response based on them. This process creates three points where interventions can occur: Write, when a candidate memory is about to be stored; Read, when memories are retrieved and incorporated into the context; and Act, when the agent uses a retrieved memory to perform an action. Each proposed defense targets one of these points, and its position is more important than its specific mechanism. The main threat addressed is straightforward: an attacker inserts text into the conversation, which becomes a memory that is later retrieved and treated as fact. Our focus is not on testing the attack itself, but on understanding the cost of defenses in benign scenarios where no injection occurs. Therefore, the attack is outside the scope of this analysis; see [5,7] for more detail.

3.2 Write-Time Defenses

Three of our four defenses activate before memory storage. Sanitization scans each candidate for patterns common in injection attacks, such as commands that direct the model to override instructions or text that addresses the system rather than the user. Any matching pattern is discarded or rewritten prior to storage. This check is purely lexical, requiring no model call, and incurs minimal per-turn cost. Provenance tags indicate each memory’s source and trust level; memories from untrusted or ambiguous origins receive a lower trust weight, checked during storage and retrieval. The cost here is minimal since trust weights are stored as metadata. Anomaly detection, the costliest defense, involves passing each candidate memory to an LLM to verify its consistency with the ongoing conversation. Inconsistent candidates are quarantined instead of stored. Although these defenses differ in mechanism and expense; two are lexical or metadata-based, and one uses a model; they all operate at the same point in the pipeline. Their shared position, rather than their differing methods, is what determines their benign-case *utility* cost, as our results show; token cost, by contrast, tracks mechanism.

3.3 Read-Time Interception

The reranker does not modify storage but operates during memory retrieval, enabling normal retrieval of top candidates based on similarity. It then evaluates each candidate against the query and other retrieved memories. Memories that poorly match or contradict others are flagged and isolated in quarantine, never incorporated into the model’s context. This approach differs from write-time defenses mainly in its timing, not mechanism. Like anomaly detection, the reranker checks for consistency but at a different stage, resulting in two main differences: First, the frequency of memory evaluation. A write-time filter assesses each memory once when stored, making that decision final. The reranker, however, re-evaluates memories during a conversation, so a memory accepted initially can later be rejected. Second, the evidence used for evaluation. A memory at write time includes its originating turn, but at retrieval, it appears as a short, standalone string compared against others. Therefore, the reranker makes decisions with less context. We highlight these points because they influence our results.

3.4 Stacked Composition

The stacked configuration sequentially applies all four defenses: sanitization, provenance, anomaly detection, and reranking. This approach aligns with current deployment recommendations, as no single defense covers the entire attack surface alone. A key concern with stacking is the potential increase in costs, as four filters offer four chances to mistakenly discard legitimate input. Whether these chances add up is an empirical question, which this experiment aims to answer. Constructing the defenses proved more difficult than expected, an important point to note. In the stacked setup, each defense encloses the next, rather than directly protecting the memory store. Consequently, the reranker lost direct access to the underlying store and had to navigate through the wrapper chain. This led to failures in initial runs on every question. The solution was straightforward, but it has significant implications: the reranker had never been tested in this configuration before, so the system never functioned as intended. A crash revealed the issue; a silent failure could have produced misleadingly correct results. Further details are discussed in Appendix D.

4 Measurement Harness

4.1 Isolation

The target measure is the marginal cost of a defense, with all other factors held constant. Our setup keeps the memory backend, retrieval configuration, answering model, question set, and grading process identical across all conditions. Only the defense itself changes. Each condition writes to the same vector store using the same embedding model, and retrieval returns the same number of candidates at a consistent similarity threshold. The answering model always receives its context in the same format, regardless of which defense produced it. The judge remains unchanged throughout the experiment, so no condition is graded by a different model or prompt than any other. Each cell also runs in its own memory namespace, so nothing written during one condition is visible to another. This matters more than it might seem. A defense that quarantines memories during one run would otherwise leave a smaller store for the next run to read from, and the resulting effect would be attributed to the wrong condition entirely.

4.2 Per-Purpose Cost Attribution

Token cost is only meaningful if we can distinguish between what the defense spent and what the agent would have spent on its own. All model traffic in our system routes through a local proxy, with each request including a header indicating its purpose: memory extraction, question answering, judging, or a specific named defense. The proxy logs tokens and costs by purpose. Summing these logs provides a clear separation between defense-related spend and the baseline framework spend for each cell, making the token overhead ratio in Section 5 a factual measurement rather than an estimate. Additionally, this method uncovered an issue that might have gone unnoticed. An early configuration had an outdated price entry that silently overrode the correct one, causing reported costs to be wrong by a factor of five, yet still appear plausible. The per-purpose logging revealed this discrepancy.

4.3 Dataset and Evidence Alignment

We evaluate on LoCoMo [20], a dataset consisting of long, multi-session conversations paired with question-answer sets. To manage ingestion costs across 90 cells, we truncate each of five conversations to 150 turns. This truncation can cause issues, as it may cut off supporting evidence for some questions. If evidence appears in a turn that is cut, no agent can answer that question, whether defending or otherwise. Including such questions would reflect the amount of conversation discarded rather than a true measure of defense. Our initial results showed about 25% accuracy mainly for this reason, which doesn't reflect the quality of defenses. To address this, we implement an evidence alignment filter: we check questions against the retained turns and exclude those with evidence outside the truncated window before running any conditions. We apply this filter equally across all conditions to prevent bias in defenses. Appendix A details the number of questions retained at this truncation length.

4.4 Judge and Calibration

An LLM judge grades answers and remains fixed across the experiment. To avoid a model grading its own output, the judge comes from a different provider than the answering model. We calibrated the judge against hand-graded items before running the experiment and measured agreement at $\kappa = 0.909$. We excluded adversarial questions from this calibration and scored them separately, since they ask whether the agent correctly declined to answer rather than whether it produced a correct fact. The separate scoring path is where our major measurement error happened. Abstention was initially identified by a regex matching a limited set of refusal phrases. Checking the items marked as failures revealed that most were not true failures — the agent had either refused correctly with phrasing the pattern didn't recognize or had properly rejected a false premise in the question. Against 58 blind hand labels, the original detector agreed only 86.2% of the time, $\kappa = 0.663$. We replaced it with a detector built from generalizable negation and refusal patterns rather than fitted to the specific misses. It agrees with all 58 hand labels, $\kappa = 1.000$. Table 1 reports both. The original scores are preserved in the run outputs and were not overwritten, so the correction is auditable.

Scorer	Agreement	Cohen’s κ
Original regex	0.862	0.663
Calibrated detector	1.000	1.000

Table 1: Abstention scorer agreement against 58 blind hand labels drawn from the decision-relevant subset (every item the original scorer marked as a failure).

5 Metrics and Estimation

5.1 Notation

Let $\mathcal{C}$ denote the set of evaluation conversations, with $|\mathcal{C}| = 5$, and let $\mathcal{R}$ denote the set of independent replicate runs of each configuration, with $|\mathcal{R}| = 3$. Let $\mathcal{D} = \{\emptyset, \text{san}, \text{prov}, \text{anom}, \text{rer}, \text{stk}\}$ index the defense conditions, where $\emptyset$ denotes the undefended baseline. A *cell* is a single triple $(d, c, r) \in \mathcal{D} \times \mathcal{C} \times \mathcal{R}$. The full experiment comprises $|\mathcal{D}| \cdot |\mathcal{C}| \cdot |\mathcal{R}| = 90$ cells, executed sequentially against a shared memory backend.

For each conversation c , let Q_c be the set of questions surviving the evidence alignment filter of Section 4, partitioned into a core set Q_c^{core} and an adversarial set Q_c^{adv} . Write $y_{d,c,r,q} \in \{0, 1\}$ for the graded correctness of the agent’s answer to question q under condition d in replicate r , and $y_{\emptyset,c,r,q}$ for the corresponding baseline outcome.

5.2 Collapsing Replicates

The pipeline is not deterministic, even at temperature zero. Identical inputs produce different memory counts across nominally repeated runs, and downstream accuracy varies with them. Replicates exist to absorb that variation rather than to enlarge the sample, so we collapse each cell’s replicates to a single verdict per question by majority vote:

$$\hat{y}_{d,c,q} = \mathbf{1} \left[\frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} y_{d,c,r,q} > \frac{1}{2} \right] \quad (1)$$

An item counts as correct only if it was answered correctly in strictly more than half of the runs. With a single replicate, Equation (1) reduces to the identity, and every estimator below reduces to its unreplicated form.

5.3 Benign Utility Delta

Between-conversation variation in this corpus is roughly an order of magnitude larger than the effects we are trying to measure, so comparing group means across conditions would be uninformative. Every condition answers the same questions about the same conversations, however, so each item has a matched baseline counterpart, and differencing within that pair removes conversation difficulty exactly.

Let $\tilde{Q}_{d,c} \subseteq Q_c$ be the questions graded under both condition d and the baseline. The per-conversation paired delta is

$$\Delta_{d,c} = \frac{1}{|\tilde{Q}_{d,c}|} \sum_{q \in \tilde{Q}_{d,c}} (\hat{y}_{d,c,q} - \hat{y}_{\emptyset,c,q}) \quad (2)$$

and the benign utility delta (BUD) weights conversations equally rather than by question count, so that a long conversation does not dominate a short one:

$$\Delta_d = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \Delta_{d,c} \quad (3)$$

A value $\Delta_d < 0$ means the defense destroyed utility on traffic containing no attack. We compute Equation (3) separately over Q_c^{core} and Q_c^{adv} .

5.4 Token Overhead

Every model call is tagged at the proxy with the purpose that issued it, so spend attributable to a defense is separable from spend the framework would have incurred regardless. Let T_d^{def} denote total tokens consumed by defense components under condition d , and T_d^{base} tokens consumed by the underlying agent. The token overhead ratio is

$$\tau_d = \frac{T_d^{\text{def}}}{T_d^{\text{base}}} \quad (4)$$

We report τ_d rather than wall-clock latency or dollar cost. Latency under our execution conditions is contaminated by provider rate-limit backoff, and dollar cost is a transient function of published prices, whereas τ_d is a property of the defense itself.

5.5 False Quarantine Rate

Our evaluation corpus is benign by construction. It contains no injected adversarial memory. Every item a defense quarantines is therefore a false positive by definition, with no compensating true positive available. Writing $n_{d,c,r}^{\text{quar}}$ for quarantined items and $n_{d,c,r}^{\text{seen}}$ for items the defense adjudicated,

$$\text{FQR}_{d,c,r} = \frac{n_{d,c,r}^{\text{quar}}}{n_{d,c,r}^{\text{seen}}}, \quad \text{FQR}_d = \frac{1}{|\mathcal{C}||\mathcal{R}|} \sum_{c,r} \text{FQR}_{d,c,r} \quad (5)$$

This quantity behaves differently from Δ_d in a way that matters for our results. The benign utility delta is an end-to-end outcome and is therefore subject to the noise floor of the entire pipeline. The false quarantine rate is read directly off the defense’s own decisions and is not attenuated by retrieval or generation downstream of it.

5.6 Uncertainty

We report three quantities. Each answers a different question, and we treat the first as the headline.

Stratified bootstrap interval Resampling questions within conversations preserves the pairing and respects the equal-weight convention of Equation (3) [21]. For $b = 1, \dots, B$ with $B = 10^4$, we draw $\tilde{Q}_{d,c}^{(b)}$ with replacement from $\tilde{Q}_{d,c}$ independently for each c , recompute $\Delta_d^{(b)}$ via Equations (2) and (3), and report the empirical 2.5th and 97.5th percentiles of $\{\Delta_d^{(b)}\}_{b=1}^B$.

Exact McNemar test Pooling paired items across conversations [22], let n_{10} count items the baseline answered correctly and the defended agent did not, and n_{01} the reverse:

$$n_{10} = \#\{(c, q) : \hat{y}_{\emptyset, c, q} = 1, \hat{y}_{d, c, q} = 0\} \quad (6)$$

$$n_{01} = \#\{(c, q) : \hat{y}_{\emptyset, c, q} = 0, \hat{y}_{d, c, q} = 1\} \quad (7)$$

Concordant items carry no information about the effect. Under the null hypothesis each discordant pair is a fair coin, which gives the exact two-sided p -value

$$p = \min \left(1, 2 \sum_{i=0}^{\min(n_{10}, n_{01})} \binom{n_{10} + n_{01}}{i} 2^{-(n_{10} + n_{01})} \right) \quad (8)$$

We use the exact binomial rather than the χ^2 approximation because $n_{10} + n_{01}$ is small for several conditions, reaching single digits in one case.

Exact sign-flip permutation test. The most conservative of the three treats the conversation, not the question, as the unit of analysis. Over all $2^{|\mathcal{C}|}$ sign assignments $\sigma \in \{-1, +1\}^{|\mathcal{C}|}$,

$$p_{\text{perm}} = 2^{-|\mathcal{C}|} \left| \left\{ \sigma : \left| \frac{1}{|\mathcal{C}|} \sum_c \sigma_c \Delta_{d,c} \right| \geq |\Delta_d| \right\} \right| \quad (9)$$

This test has a hard floor that must be stated explicitly. With $|\mathcal{C}| = 5$ there are only $2^5 = 32$ possible assignments, so the smallest attainable two-sided p -value is $2/32 = 0.0625$. At this sample size Equation (9) cannot reach $p < 0.05$ for any effect size whatsoever. That is a property of the design, not of the data, and it is why we treat the bootstrap interval as primary evidence and report p_{perm} only as a conservative bound.

5.7 Measurement Resolution

The width of the bootstrap interval determines what this experiment can and cannot see. Across conditions the intervals span roughly ± 4.5 percentage points around their point estimates. An effect smaller than that is indistinguishable from noise here, regardless of which estimator is applied.

Reported nulls should therefore be read as *no effect detectable at this resolution*, not as evidence of an effect of exactly zero. This bound follows from $|\mathcal{C}| = 5$ together with the pipeline’s intrinsic run-to-run variance, and we return to it in Section 9.

6 Experimental Setup

6.1 Models and Configuration

Table 2 lists every model used in the experiment. A single backbone, `gemini-3.1-flash-lite`, handles memory extraction, question answering, and the anomaly detection defense.

Using a single model for all three matters for interpretation: when comparing an LLM-based defense to a pattern-matching one, the difference observed is due to the defense method itself, not a change in the underlying model. We arrived at this backbone through elimination rather than choice. An earlier setup employed a more capable Gemini variant, but its expense made 90 cells unfeasible. This change also revealed a bug worth noting. The extraction step lacked an explicit token limit, causing the larger model to focus on reasoning within its budget before outputting structured data. This silently led to truncating memories instead of causing a failure. As a result, extraction produced only about a quarter of the expected memories, with no indication of a problem in the output. Introducing an explicit limit fixed this issue. We return to this class of failure in Appendix D.

Role	Model	Provider
Memory extraction	<code>gemini-3.1-flash-lite</code>	Google
Question answering	<code>gemini-3.1-flash-lite</code>	Google
Anomaly detection	<code>gemini-3.1-flash-lite</code>	Google
Retrieval embedding	<code>text-embedding-3-small</code>	OpenAI
Grading judge	<code>claude-haiku-4-5-20251001</code>	Anthropic

Table 2: Models used in every cell of the experiment. The same backbone serves extraction, answering, and the LLM-based defense, so differences between conditions cannot be attributed to a model change. The judge comes from a different provider than the backbone so no model grades its own output.

Generation parameters are fixed across all cells, with temperature set to zero throughout. This matters because it did not make the pipeline deterministic, a point we revisit in Section 6.3. Retrieval uses `text-embedding-3-small` with a fixed candidate count and similarity threshold, held constant across all conditions. Vector storage is local, with each cell writing to its own isolated namespace.

6.2 Conditions and Protocol

Six conditions are assessed: the baseline without defenses, each of the three write-time protections alone, the read-time reranker alone, and a combined stack of all four. All conditions are tested with the same five conversations, each limited to 150 turns and a maximum of 75 questions post-evidence alignment. Each cell operates in two phases. First, the truncated conversation is processed turn by turn, with the relevant defense activated at its respective pipeline stage. Then, the questions remaining for that conversation are answered using the resulting memory store, and a judge evaluates each response.

6.3 Paired Design

We assess each non-baseline condition by comparing it to the undefended baseline within the same conversation and question set. This approach ensures the validity of the paired estimators in Section 5: a defense is never contrasted with a different conversation, question set, or judge run than its baseline. We run each condition three times. This was not originally planned. Early tests of the same condition on the same conversation showed varying memory counts between runs, even at temperature zero, indicating the pipeline had an inherent noise floor that a single run could not distinguish from actual defense effects. Section 5 explains how we aggregate replicates before calculating statistics, while Appendix D illustrates a case where this repetition changed our conclusion.

6.4 Reproducibility

Each cell maintains a comprehensive summary, including memory count, token costs by purpose, question-specific grading results, and the quarantine log. All reported statistics trace back to their originating run. We provide all related code with this paper, covering the evidence alignment filter, judge prompts, abstention detector, and complete per-cell outputs. Two limitations should be explicitly acknowledged. First, because the pipeline has inherent run-to-run variability, no single cell replicate is definitive; only the aggregated, replicated results are reliable. Second, our backbone and judge models are commercial, accessed via an API, and their behavior and pricing may change. This variability explains why Section 5 presents overhead as a ratio rather than focusing on dollar costs as a primary metric.

7 Results

7.1 Benign Utility

Defense	Stage	Utility Δ (pp)		Token ovh.	FQR
No defense	—	—		—	0.000
Sanitize	write	+1.8	[−1.7, +5.6]	0.0%	0.000
Provenance	write	+1.9	[−2.2, +6.0]	0.0%	0.000
Anomaly	write	+3.1	[−0.9, +7.1]	1.2%	0.000
Rerank	read	−4.4*	[−9.0, −0.05]	2.7%	0.336
Stacked	write + read	−3.1	[−7.4, +1.0]	4.1%	0.303

Table 3: Benign-case costs for each defense, averaged across three replicates of five conversations (totaling 15 runs per condition). Utility Δ indicates the paired change in core accuracy relative to the undefended baseline, with 95% stratified bootstrap confidence intervals. Token overhead represents defense-attributed tokens as a fraction of framework baseline tokens. FQR denotes the average false quarantine rate on purely benign traffic.*McNemar $p = 0.064$; interval excludes zero, but narrowly.

Table 3 shows the paired benign utility delta for each condition relative to the undefended baseline, combined from three replicates of five conversations. Figure 1 visualizes these findings as a forest plot, facilitating easier comparison across conditions at a glance than the table alone.

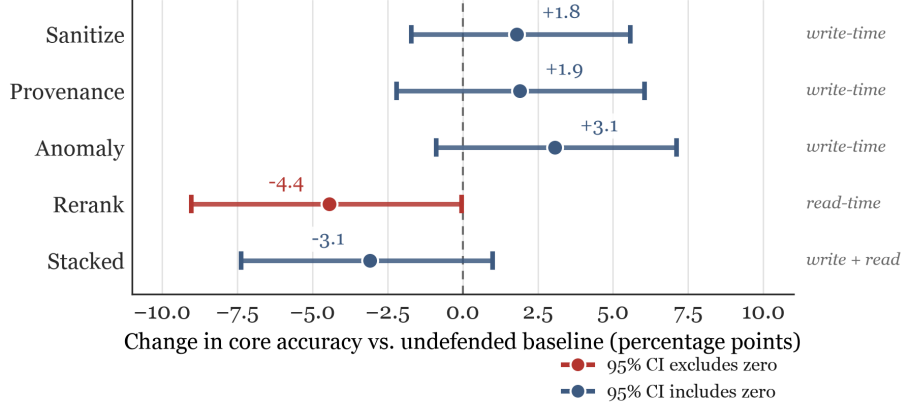


Figure 1: Paired differences in core accuracy compared to the undefended baseline, averaged over three replicates across five conversations. Bars indicate stratified bootstrap 95% confidence intervals.

The three write-time defenses show no measurable effect distinct from random noise. Sanitization, provenance checking, and anomaly detection all yield confidence intervals that comfortably include zero. All three estimates are slightly positive, which we interpret as sampling variation rather than true benefit. This pattern persists whether or not the defense involves a model call. Anomaly detection, which is based on LLM, also results in the same null outcome as the two defenses that rely on simple pattern matching.

The reranker does not follow this pattern. It reduces core accuracy by 4.4 points, with a 95% bootstrap interval of $[-9.0, -0.05]$. Since this interval is very close to zero, we do not consider this result definitive on its own — the upper bound is just below zero, and the McNemar test on the pooled data yields $p = 0.064$. This is slightly above the usual 0.05 significance threshold. We report this as a genuine finding because the effect maintained its sign and approximate size across three independent replicates. However, a single run showing this result would not have provided the same level of confidence.

The stacked condition sits between the two. Its accuracy loss (3.1 points, 95% CI $[-7.4, +1.0]$) is smaller than the reranker’s alone, and its own interval includes zero. We return to what this does and does not tell us about composition in Section 7.4.

7.2 Token Overhead

Table 3 also reports token overhead by condition, and the pattern here reflects mechanism rather than pipeline stage. Sanitization and provenance checking cost almost nothing, since neither makes a model call. Anomaly detection, despite acting at write time alongside these two, carries a measurable overhead because every candidate memory is sent to an LLM before it is stored. The reranker’s overhead is larger still, since it runs once per question rather than once per memory. The stacked condition’s overhead is close to the sum of its parts, the one place in our results where cost behaves additively even though accuracy does not. We report this ratio rather than latency, for the reason given in Section 5: our later cells were affected by provider rate-limit backoff that inflates wall-clock time without reflecting anything about the defense itself.

7.3 False Quarantine on Benign Traffic

Figure 2 shows the false quarantine rate by condition, and this is the result we consider the clearest in the paper.

The baseline and all three write-time defenses report a false quarantine rate of zero, indicating that no non-attack traffic is quarantined; expected and correct behavior. In contrast, the reranker does not exhibit this property; on conversations without any attack, it quarantines 33.6% of legitimate memories it evaluates.

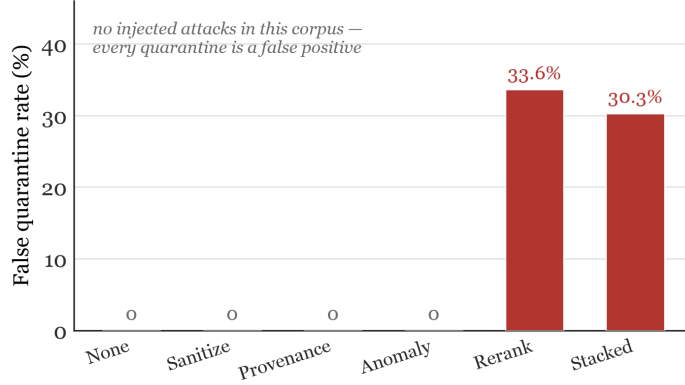


Figure 2: False quarantine rate per condition: since the corpus contains no injected adversarial memory, all quarantines are automatically false positives by design.

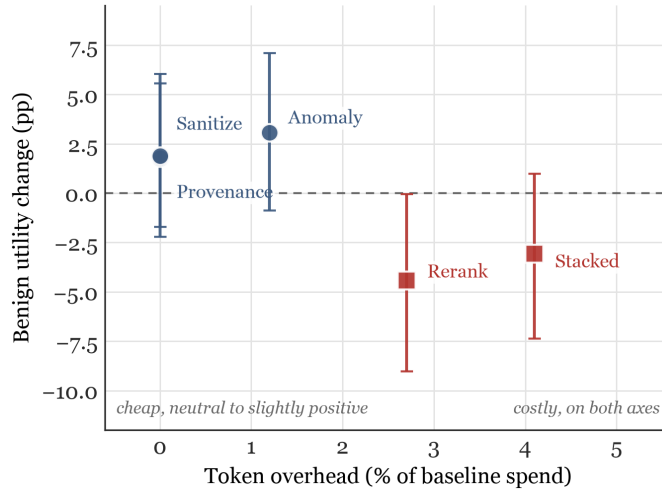


Figure 3: Token overhead for benign utility changes. Write-time defenses are concentrated near the origin, while the read-time reranker is isolated in the expensive quadrant.

We place greater emphasis on this false quarantine rate than on the accuracy result. The accuracy measure is borderline statistically, and a cautious reader might require more evidence before trusting it. However, the false quarantine rate is a straightforward count, not accompanied by a confidence interval like the benign utility delta. It directly reflects the defense’s decisions on a dataset purposely known to contain no items to quarantine. A defense that discards a third of legitimate memories during normal traffic incurs its own costs, regardless of its impact on subsequent accuracy.

7.4 What This Experiment Can and Cannot Resolve

Figure 3 plots each condition on a graph with token overhead on one axis and accuracy change on the other. This visualizes the clear division in our results: the write-time defenses cluster close to the origin, while the reranker appears isolated in the costly, high-overhead quadrant.

Our bootstrap intervals cover approximately ± 4.5 percentage points around each condition, establishing a strict limit on our claims. The three write-time nulls should be interpreted as *no effect detectable at this resolution*, not as evidence that these defenses are free; any real cost of one or two points would be undetectable with our current data. The reranker’s result is close to our resolution threshold, which is why we describe it as borderline rather than significant. The comparison between the reranker alone and the stacked condition warrants caution. The smaller point estimate for the stacked condition aligns with the idea that write-time defenses partly counteract what the reranker

dismisses, but its confidence interval overlaps significantly with that of the reranker. We can say the data show no evidence of compounding. We cannot say the data prove the defenses interact protectively. These are two distinct claims, and only the first is supported by our data.

8 Discussion

Our primary conclusion is that a defense’s placement in the pipeline, rather than its internal components, primarily influences cost. Anomaly detection employs an LLM, yet behaves statistically like sanitization and provenance checking, which are string and metadata operations requiring no model call at all. The reranker, our only read-time defense, differs from the other three, even though it makes a similar consistency assessment to anomaly detection. The assumption that LLM-based defenses incur higher utility costs is unfounded here; instead, the cost depends on the defense’s position in the process, not its composition.

This split in cost structure follows from what we outlined in Section 3. Write-time defenses make a final decision based on the memory and context at that moment. In contrast, read-time defenses repeatedly evaluate the same memory, with only limited context available each time. This repetition increases the chance of discarding legitimate memories, especially as less context makes judgments less reliable. We haven’t definitively isolated which factor—repetition or other aspects of read-time checks—is responsible, but this offers a plausible explanation.

The most critical metric for practitioners is the false quarantine rate. A benign-case accuracy difference close to zero is easy to read as a defense that costs nothing, but the reranker’s false quarantine rate reveals a different cost: on non-attack traffic, legitimate memories are discarded roughly one-third of the time when judged, leading to silent memory loss without error messages. Accurate metrics can mask this issue when redundant information supports facts across multiple memories. Understanding how often legitimate memories are discarded is more useful for deployment decisions than a general accuracy score.

Results from the stacked defenses challenge the idea that costs simply add up. If that were true, stacking four defenses (including the reranker) should be at least as costly as using the reranker alone. Instead, we observe less accuracy loss and fewer false quarantines. One possible explanation is that the write-time defenses reduce the amount of material the reranker must evaluate, making its judgments easier and reducing errors. Our data support this hypothesis but do not conclusively demonstrate it, as we varied which defenses were active without controlling for order or specific interactions.

Previous evaluations, discussed in Section 2, primarily focus on attack success rates. Our findings do not contradict this; rather, they highlight that attack success is only part of the story. While attack success indicates effectiveness against targeted threats, benign-case cost reveals the impact on normal traffic. Attack success alone is insufficient for deployment decisions, as these metrics don’t correlate in a way that lets one substitute for the other.

9 Limitations

This experiment has certain limitations, which we clearly state instead of leaving unstated. Five conversations and three replicates give bootstrap intervals of roughly ± 4.5 percentage points, discussed in Section 5. This resolution can detect the reranker’s cost when it is large enough to be near the edge of the interval. However, it cannot identify a true cost of one or two points in any situation. Our null results on the write-time defenses should be interpreted with this in mind: not observing an effect does not mean there is no effect below this resolution.

Every condition uses one model, `gemini-3.1-flash-lite`, for extraction, answering, and the one LLM-based defense. This isolates the defense as the sole variable in our experiment, fulfilling the design’s purpose. However, it remains uncertain whether the observed write-time and read-time split would persist with a different backbone. A model with varied instruction-following behavior or differing sensitivity to brief, context-limited inputs during retrieval could potentially reduce or increase the gap between write and read costs.

Core accuracy is graded by a judge calibrated against hand-labeled items to $\kappa = 0.909$; adversarial abstention is graded by the detector described in Section 4, calibrated separately to $\kappa = 1.000$ against

58 hand labels. We performed both calibrations on items drawn from our own runs rather than an independently constructed benchmark, which is standard practice but still means the calibration and evaluation share a source.

Our threat model in Section 3 identifies three intervention points for defenses, but we only measure two. We do not evaluate act-time interception here and lack evidence to compare its benign-case cost to the write-time and read-time results presented.

Token overhead, shown as a ratio in Section 5, remained consistent during our experiment’s pricing period. However, the dollar cost implied by this ratio varies, as it depends on current API prices, which fluctuate over time. Therefore, we present the ratio rather than a dollar value. The ratio assumes a specific model and would need to be recalculated if a different backbone is used.

10 Conclusion

We evaluated the costs of four memory-poisoning defenses on conversations without attacks, using a harness that holds every variable fixed except the defense, and replicates each condition three times. This approach helps us differentiate genuine effects from noise. Three write-time defenses, including one based on large language models, showed no detectable benign-case cost beyond noise levels. In contrast, a single read-time defense exhibited a marginal but consistent accuracy decline and a clear false quarantine rate, where every quarantine was, by design, a mistake. Combining all four defenses did not increase this cost. Where a defense intercepts the pipeline predicted its benign-case price better than whether it used a language model, and that distinction is invisible to any evaluation reporting attack success rate alone.

A Retention Under Truncation

Truncating each conversation to 150 turns results in the removal of some supporting evidence for questions, as explained in Section 4. Table 4 shows, for each of the five conversations, how many questions remain after applying the evidence alignment filter at the truncation length used here, along with the total number of questions before filtering.

Conversation	Total turns	Questions (raw)	Questions (aligned)
conv-26	150	86	75
conv-30	150	51	51
conv-41	150	32	32
conv-42	150	42	42
conv-43	150	55	55

Table 4: Question retention after evidence alignment at 150-turn truncation.

B Judge Calibration Items

The judge described in Section 4 was calibrated using 24 hand-graded core-accuracy items before the experiment, achieving an agreement of $\kappa = 0.909$ with human labels. Adversarial abstention was calibrated separately on 58 hand-labeled items, as detailed in Table 1 of the main text. To keep the appendix concise, we omit the full item text here; the complete calibration set is available with the code accompanying this paper.

C Per-Cell Results

Table 5 details all 90 cells in this experiment: six conditions, five conversations, and three replicates.

Table 5: Per-cell results, all 90 runs.

Condition	Conv.	Seed	Mem.	n_q	Core4	Adv.	Quar.
No defense	conv-26	1	309	75	0.651	1.000	0
No defense	conv-26	2	292	75	0.697	1.000	0
No defense	conv-26	3	316	75	0.621	0.889	0
No defense	conv-30	1	282	51	0.684	0.923	0
No defense	conv-30	2	248	51	0.737	0.923	0
No defense	conv-30	3	253	51	0.711	1.000	0
No defense	conv-41	1	250	32	0.800	1.000	0
No defense	conv-41	2	234	32	0.840	1.000	0
No defense	conv-41	3	243	32	0.800	1.000	0
No defense	conv-42	1	247	42	0.485	1.000	0
No defense	conv-42	2	248	42	0.545	1.000	0
No defense	conv-42	3	247	42	0.485	1.000	0
No defense	conv-43	1	232	55	0.632	1.000	0
No defense	conv-43	2	253	55	0.658	0.882	0
No defense	conv-43	3	246	55	0.605	0.941	0
Sanitize	conv-26	1	319	75	0.606	0.667	0
Sanitize	conv-26	2	314	75	0.576	0.778	0
Sanitize	conv-26	3	314	75	0.636	0.889	0
Sanitize	conv-30	1	233	51	0.711	0.846	0
Sanitize	conv-30	2	256	51	0.711	0.846	0
Sanitize	conv-30	3	247	51	0.658	1.000	0
Sanitize	conv-41	1	229	32	0.840	1.000	0
Sanitize	conv-41	2	238	32	0.840	1.000	0
Sanitize	conv-41	3	249	32	0.840	1.000	0
Sanitize	conv-42	1	265	42	0.576	1.000	0
Sanitize	conv-42	2	258	42	0.545	1.000	0
Sanitize	conv-42	3	280	42	0.576	0.889	0
Sanitize	conv-43	1	230	55	0.632	0.882	0
Sanitize	conv-43	2	242	55	0.605	0.882	0
Sanitize	conv-43	3	247	55	0.605	0.882	0
Provenance	conv-26	1	291	75	0.576	1.000	0
Provenance	conv-26	2	309	75	0.667	1.000	0
Provenance	conv-26	3	290	75	0.697	1.000	0
Provenance	conv-30	1	271	51	0.658	0.923	0
Provenance	conv-30	2	253	51	0.684	0.923	0
Provenance	conv-30	3	260	51	0.711	0.923	0
Provenance	conv-41	1	239	32	0.800	1.000	0
Provenance	conv-41	2	237	32	0.840	1.000	0
Provenance	conv-41	3	246	32	0.800	1.000	0
Provenance	conv-42	1	241	42	0.606	0.889	0
Provenance	conv-42	2	230	42	0.545	1.000	0
Provenance	conv-42	3	253	42	0.545	1.000	0
Provenance	conv-43	1	232	55	0.605	0.882	0
Provenance	conv-43	2	245	55	0.579	0.882	0
Provenance	conv-43	3	219	55	0.605	0.882	0
Anomaly	conv-26	1	288	75	0.576	0.889	0
Anomaly	conv-26	2	295	75	0.712	1.000	0
Anomaly	conv-26	3	323	75	0.621	0.889	0
Anomaly	conv-30	1	256	51	0.711	0.923	0
Anomaly	conv-30	2	228	51	0.763	0.923	0
Anomaly	conv-30	3	239	51	0.763	0.846	0
Anomaly	conv-41	1	235	32	0.840	1.000	0
Anomaly	conv-41	2	228	32	0.840	1.000	0
Anomaly	conv-41	3	265	32	0.840	1.000	0
Anomaly	conv-42	1	254	42	0.545	1.000	0
Anomaly	conv-42	2	259	42	0.576	1.000	0

Condition	Conv.	Seed	Mem.	n_q	Core4	Adv.	Quar.
Anomaly	conv-42	3	227	42	0.545	1.000	0
Anomaly	conv-43	1	230	55	0.579	0.882	0
Anomaly	conv-43	2	237	55	0.605	0.882	0
Anomaly	conv-43	3	221	55	0.605	0.882	0
Rerank	conv-26	1	292	75	0.591	0.778	40
Rerank	conv-26	2	329	75	0.606	1.000	11
Rerank	conv-26	3	299	75	0.591	1.000	46
Rerank	conv-30	1	243	51	0.553	0.923	50
Rerank	conv-30	2	248	51	0.474	1.000	38
Rerank	conv-30	3	249	51	0.632	1.000	47
Rerank	conv-41	1	219	32	0.760	1.000	20
Rerank	conv-41	2	247	32	0.760	1.000	21
Rerank	conv-41	3	245	32	0.800	1.000	13
Rerank	conv-42	1	241	42	0.455	1.000	48
Rerank	conv-42	2	246	42	0.576	1.000	61
Rerank	conv-42	3	245	41	0.531	1.000	60
Rerank	conv-43	1	252	55	0.500	1.000	100
Rerank	conv-43	2	238	55	0.526	1.000	95
Rerank	conv-43	3	236	55	0.500	1.000	106
Stacked (all four)	conv-26	1	334	75	0.636	1.000	24
Stacked (all four)	conv-26	2	318	75	0.667	1.000	29
Stacked (all four)	conv-26	3	312	75	0.697	1.000	24
Stacked (all four)	conv-30	1	247	51	0.658	0.923	37
Stacked (all four)	conv-30	2	246	51	0.605	1.000	37
Stacked (all four)	conv-30	3	258	51	0.632	1.000	43
Stacked (all four)	conv-41	1	255	32	0.720	1.000	10
Stacked (all four)	conv-41	2	241	32	0.800	0.857	10
Stacked (all four)	conv-41	3	240	32	0.800	1.000	13
Stacked (all four)	conv-42	1	260	42	0.485	1.000	59
Stacked (all four)	conv-42	2	249	42	0.545	1.000	37
Stacked (all four)	conv-42	3	251	42	0.515	1.000	56
Stacked (all four)	conv-43	1	240	55	0.500	1.000	94
Stacked (all four)	conv-43	2	249	55	0.526	1.000	83
Stacked (all four)	conv-43	3	239	55	0.474	0.941	125

D Implementation Pitfalls That Silently Corrupt Measurement

Three bugs arose during this project, all sharing a common pattern: each produced a seemingly plausible number rather than a clear failure. We document them here because a measurement paper’s credibility partly depends on its ability to identify its own errors.

Silent truncation under a token budget Initially, our memory extraction step had no explicit output token limit. When using a larger backbone model, the model exhausted its token budget on internal reasoning before generating the structured output, resulting in truncated responses. The pipeline did not raise an error; it simply stored whatever partial output remained, discarding roughly three-quarters of the memories it should have produced. Imposing an explicit output token limit resolved this issue.

Defense composition breaking on a wrapped store In the stacked configuration, each defense wraps the next rather than wrapping the memory store directly. Our reranker assumed direct access to the store and lacked a way to reach it through wrappers, causing attribute errors on every question. The fix involved having the reranker traverse the wrapper chain to access the store.

A narrow detector mistaken for ground truth An automated abstention detector, based on expected refusal phrases, silently misgraded correct refusals that used different wording. Instead of

flagging an error, it produced a plausible, confidently wrong score. The issue was only uncovered by examining the actual text of items it marked as failures.

References

- [1] Dong, S., Xu, S., He, P., Li, Y., Tang, J., Liu, T., Liu, H. & Xiang, Z. (2025) A practical memory injection attack against LLM agents. *arXiv preprint arXiv:2503.03704*.
- [2] Srivastava, S.S. & He, H. (2025) MemoryGraft: Persistent compromise of LLM agents via poisoned experience retrieval. *arXiv preprint arXiv:2512.16962*.
- [3] Chen, Z., Xiang, Z., Xiao, C., Song, D. & Li, B. (2024) AgentPoison: Red-teaming LLM agents via poisoning memory or knowledge bases. In *Advances in Neural Information Processing Systems 37*, pp. 130185–130213.
- [4] Zou, W., Geng, R., Wang, B. & Jia, J. (2025) PoisonedRAG: Knowledge corruption attacks to retrieval-augmented generation of large language models. In *Proceedings of the 34th USENIX Security Symposium*, pp. 3827–3844. Seattle, WA.
- [5] Chu, K. (2026) A systematic survey of security threats and defenses in LLM-based AI agents: A layered attack surface framework. *arXiv preprint arXiv:2604.23338*.
- [6] Zhang, H., Huang, J., Mei, K., Yao, Y., Wang, Z., Zhan, C., Wang, H. & Zhang, Y. (2025) Agent Security Bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In *International Conference on Learning Representations*.
- [7] Dash, P., Ge, T., Jain, A., Shah, T. & Shang, Z. (2026) From untrusted input to trusted memory: A systematic study of memory poisoning attacks in LLM agents. *arXiv preprint arXiv:2606.04329*.
- [8] Debenedetti, E., Shumailov, I., Fan, T., Hayes, J., Carlini, N., Fabian, D., Kern, C., Shi, C., Terzis, A. & Tramèr, F. (2025) Defeating prompt injections by design. *arXiv preprint arXiv:2503.18813*.
- [9] Sunil, B.D., Sinha, I., Maheshwari, P., Todmal, S., Mallik, S. & Mishra, S. (2026) Memory poisoning attack and defense on memory based LLM-agents. *arXiv preprint arXiv:2601.05504*.
- [10] Zhou, H., Lee, K.-H., Zhan, Z., Chen, Y., Li, Z., Wang, Z., Haddadi, H. & Yilmaz, E. (2025) TrustRAG: Enhancing robustness and trustworthiness in retrieval-augmented generation. *arXiv preprint arXiv:2501.00879*.
- [11] Sharma, T. (2026) SMSR: Certified defence against runtime memory poisoning in persistent LLM agent systems. *arXiv preprint arXiv:2606.12703*.
- [12] Gowda, I. (2026) MEMSAD: Gradient-coupled anomaly detection for memory poisoning in retrieval-augmented agents. *arXiv preprint arXiv:2605.03482*.
- [13] Xiang, C., Wu, T., Zhong, Z., Wagner, D., Chen, D. & Mittal, P. (2024) Certifiably robust RAG against retrieval corruption. *arXiv preprint arXiv:2405.15556*.
- [14] Wei, Q., Yang, T., Wang, Y., Li, X., Li, L., Yin, Z., Zhan, Y., Holz, T., Lin, Z. & Wang, X. (2025) A-MemGuard: A proactive defense framework for LLM-based agent memory. *arXiv preprint arXiv:2510.02373*.
- [15] Beurer-Kellner, L., Buesser, B., Crețu, A.-M., Debenedetti, E., Dobos, D., Fabian, D., Fischer, M., Froelicher, D., Grosse, K., Naeff, D., Ozoani, E., Paverd, A., Tramèr, F. & Volhejn, V. (2025) Design patterns for securing LLM agents against prompt injections. *arXiv preprint arXiv:2506.08837*.
- [16] Chhikara, P., Khant, D., Aryan, S., Singh, T. & Yadav, D. (2025) Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*.
- [17] Wu, D., Wang, H., Yu, W., Zhang, Y., Chang, K.-W. & Yu, D. (2025) LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *International Conference on Learning Representations*, pp. 86809–86836.
- [18] Cherif, A. (2026) AgentMemBench: A systematic benchmark for evaluating long-term memory management strategies in conversational AI agents. *arXiv preprint arXiv:2608.00009*.
- [19] Wolff, B. & Bennati, J. (2026) Cost and accuracy of long-term memory in distributed multi-agent systems based on large language models. *arXiv preprint arXiv:2601.07978*.
- [20] Maharana, A., Lee, D.-H., Tulyakov, S., Bansal, M., Barbieri, F. & Fang, Y. (2024) Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 13851–13870.
- [21] Efron, B. & Tibshirani, R.J. (1993) *An Introduction to the Bootstrap*. New York, NY: Chapman & Hall/CRC.

[22] McNemar, Q. (1947) Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika* **12**(2):153–157.