

Persistent Memory Poisoning in Long-Lived AI Agents: A Threat Analysis and Trust-Aware Memory Defense Framework

Gaurav Chaulagain

Department of Computer Science, St. Xavier's College, Kathmandu, Nepal

Email: gauravchaulagain017@sxc.edu.np

Medhavee Upadhyaya

Department of Computer Science, St. Xavier's College, Kathmandu, Nepal

Email: medhaveeupadhyaya017@sxc.edu.np

Abstract: Many artificial intelligence (AI) agents use persistent memory to maintain context, remember user preferences, and streamline tasks across multiple interactions with a user. This capability enables long-term personalization, but also creates a security risk because the agent's behavior could be affected by information from an earlier interaction, including information from an adversarial source. Persistent memory poisoning is a form of attack in which malicious information continues to affect the agent even after its original source is no longer present. In this paper, persistent memory poisoning in long-lived AI agents is explored, and a trust-aware defense framework called M-Guard is introduced, which incorporates provenance tracking, memory-write validation, trust-aware retrieval, temporal revalidation, and execution-time verification. The proposed research methodology is based on a series of controlled scenarios that include direct memory injection, multi-record contamination, delayed trigger-based poisoning, and provenance laundering. A simulation-based evaluation compares five configurations: unrestricted memory, prompt-level filtering, write-time filtering, trust-aware retrieval, and the complete M-Guard framework. The evaluation measures attack success, legitimate task completion, the false-positive rate, and utility retention. In a single seeded simulation run over 1,000 multi-session trajectories (500 adversarial and 500 benign), M-Guard reduced the attack success rate from 72.4% (362/500 adversarial trajectories) under unrestricted memory to 12.8% (64/500; 95% CI 10.2–16.0%) while keeping benign task completion at 89.2% (446/500), compared with 94.2% (471/500) for unrestricted memory. This study argues that persistent memory should be treated as a security boundary rather than assumed to be a trusted store of knowledge. The proposed framework provides a basis for reproducible experimentation and subsequent validation with real AI agents.

Keywords: Persistent Memory Poisoning, AI Agents, AI Security, Long-Term Memory, Prompt Injection, Trust-Aware Retrieval, Provenance Tracking, Memory Integrity

1. Introduction

1.1 Background

Artificial Intelligence (AI) systems are becoming more and more capable of handling information, finding relevant knowledge and retaining helpful context between tasks. Since the beginnings of machine learning with neural networks, memories have been explored as a way of augmenting the capabilities of a network with information not in its immediate input. Memory networks, which store and retrieve information [1], and differentiable neural computers, which combine neural computation with an external memory [2], are examples of such systems. Lewis et al. combined a parametric generator with non-parametric retrieved knowledge [3]. Guu et al. introduced REALM, a dense retrieval model jointly trained with language-model pretraining [4] and Karpukhin et al. showed the effectiveness of dense retrieval for open-domain question answering [5]. Subsequent studies explored retrieval-based generation and enhanced ways of leveraging retrieved passages to

answer questions [6][7]. The methods presented here highlight that external information can enhance the capabilities of AI systems, but also that it introduces a requirement for reliable and accurate information in storage. Thus, the security of such information is important to consider. Adversarial Machine Learning has shown how a model can be influenced by malicious inputs or corrupted training data [8][9][10]. Another example is backdoor attacks: An attacker may add an association that makes a model misbehave when a certain trigger is shown [11][12]. Although these studies focus on model training or input manipulation, they share a general idea: an AI system can behave in unwanted ways when the inputs or processes it depends on have been manipulated.

1.2 Problem Statement

Persistent Memory poisoning is an attack where the adversary feeds misleading, false, or malicious information into an AI agent's long-term memory which can impact its later responses or actions. The poisoning can happen in a single interaction, but its effect can persist after the original content has left the active conversation. This results in a potential security problem for systems that store information from previous tasks and make decisions based on that information in the future. Several lines of earlier research help to explain this threat. The research on data poisoning focuses on the impact of tampered training instances on learning and prediction [9]. In backdoor research, it's shown that malicious associations can be hidden for a specific trigger [11][12]. Studies of adversarial examples in NLP also demonstrate that changes in the input can affect the output of an NLP model [13][14]. These research areas, however, alone do not provide a complete understanding of how untrusted information can be stored, retrieved, transformed, and reused through multiple interactions with the memory-enabled AI agent. The issue isn't just about inserting a bad record. It also involves the possibility of the record being retrieved later, whether the agent uses the record as evidence, and the impact on a later decision. A defense that considers only the initial input can miss risks that arise after information is stored. Likewise, a memory system that does not remember the source of the information could make it harder to distinguish reliable information from unverifiable or conflicting information. The objective of this study is thus to examine Persistent Memory Poisoning as a lifecycle security issue. It suggests a framework, which we call M-Guard (Memory Guard with Provenance, Validation and Temporal Reassessment), to be used to assess memory records at multiple stages and minimize the chances that the malicious information will impact the subsequent agent action.

1.3 Research Objectives

The objectives of this study are:

- To describe the attack phases and threat model of persistent memory poisoning.
- To design a trust-aware memory defense mechanism for long-lived AI agents.
- To develop an experimental methodology to assess memory poisoning for several sessions and attack scenarios.
- To specify how the proposed framework should be compared with simpler protection mechanisms using security and utility metrics.
- To help determine any remaining weaknesses, obstacles, and avenues for further study.

1.4 Research Questions

The evaluation is designed to answer the following research questions. This paper specifies how each question is measured and answers each one using results from a controlled simulation; confirming the answers with real AI agents remains future work.

RQ1: How does the cumulative attack success rate of persistent memory poisoning change across subsequent agent sessions following the initial memory injection?

RQ2: How effective are provenance tracking, write-time validation and trust-aware retrieval in decreasing the success of memory poisoning attacks?

RQ3: Is there a trade-off between reducing malicious behavior and maintaining legitimate task completion?

RQ4: What kind of attacks are still hard to detect when malicious information is stored in multiple memory records and is triggered later by different contextual stimuli?

1.5 Research Contributions

This paper makes four contributions.

Firstly, this paper provides a threat model based on the life cycle of a memory record, i.e., memory creation, storage, retrieval, use, and removal.

Secondly, it proposes a layered defense framework, M-Guard, that links each memory record to its origin and evaluates it before it is stored and before it is used.

Third, it establishes a measurable, repeatable design for evaluating the success of the attack, how long the poisoned memory remains active, the detection performance, and the success of legitimate tasks.

Lastly, it reports a simulation-based evaluation that compares M-Guard with four baseline configurations and makes residual risks such as trusted source confusion, multi-record interactions and delayed activation explicit.

2. Related Work and Research Gap

2.1 Memory Architectures and Retrieval-Augmented Systems

The study of memory mechanisms can help to enhance the capacity of neural systems to remember and recall information. Weston et al. introduced this idea with memory networks, which use an explicit memory to support reasoning over stored facts [1]. Graves et al. introduced the concept of differentiable neural computers, a hybrid of neural networks and an external memory that can be read and written during the computation [2]. These methods introduced key ideas for systems that rely on information stored in the system, and not just on the immediate information they receive. A related basis is retrieval-augmented language processing. Lewis et al. used a pretrained sequence-to-sequence model and a dense vector index of external documents [3]. REALM integrates knowledge retrieval into language-model pretraining [4], and Dense Passage Retrieval improves retrieval for open-domain question answering [5]. Following that, Izacard and Grave investigated the possibility of using passages from a retrieved document as a basis for generating answers to a question [6]. Borgeaud et al. showed that accessing data from an external large data store can boost the performance of a language model [7]. These studies focus mainly on capability, retrieval quality and answer generation. Their findings are relevant to persistent memory security because they demonstrate how much downstream output relies on the retrieved information. But just because content is retrieved doesn't necessarily mean the content is accurate and reliable. If a malicious record is relevant to a query, a retrieval system that works correctly will retrieve it. This motivates studying memory integrity and trust assessment alongside retrieval performance.

2.2 Data Poisoning and Backdoor Attacks

Data poisoning is a well-studied problem in the field of adversarial machine learning. Biggio et al. presented a study on poisoning attacks against support vector machines (SVMs), demonstrating that an attacker can alter training data to alter the decision boundaries learned by the machine [8]. Steinhardt et al. explored the shortcomings of defenses against data poisoning and proposed a way to reason about data poisoning robustness [9]. In an attempt to find methods of identifying influential training examples, Koh and Liang studied techniques that can be used to analyse how individual training points affect model predictions [10]. Another type of persistent malicious influence is backdoor attacks. Gu et al. [11] proposed a technique of training a model to generate an attacker-specified output after receiving a trigger, called BadNets. Chen et al. showed that

targeted backdoors can be implanted in deep neural networks through data poisoning to induce attacker-controlled behavior [12]. These works show that harmful effects may appear only under specific conditions and remain hidden otherwise. Adversarial manipulation is also possible for natural-language systems. Ebrahimi et al. proposed white-box attacks that flip characters in the input text to change a model's prediction [13]. Wallace et al. studied universal adversarial triggers, demonstrating that certain sequences of tokens can trigger undesirable model behavior when added to any input [14]. Kurita et al. studied the poisoning of pre-trained language models to add specific behaviours [15]. While these studies offer concepts that can be useful to consider when dealing with malicious data and hidden triggers, they differ from persistent memory poisoning in an important way. Training-time poisoning alters the examples used to train a model, whereas memory poisoning corrupts information that an agent stores during normal operation. In memory poisoning, the model itself is unchanged, yet its later actions can be affected by the tainted memory. This difference motivates evaluating memory as a distinct and potentially vulnerable component.

2.3 Security and Reliability of Language Models

Studies on the trustworthiness of language models reveal other issues with how AI systems behave. In natural language processing, Wallace et al. explored adversarial examples, showing that models can be sensitive to text inputs that are carefully crafted [14]. Kurita et al. demonstrated that pretrained language models are vulnerable to weight-poisoning attacks [15]. The studies suggest that system reliability is related to not only the architecture of the system, but also the information that is fed into and stored by the system. Bender et al. discussed risks of large language models, including the composition of training data, bias, and the tendency of readers to over-interpret generated text [16]. Weidinger et al. proposed a taxonomy of risks from language models (LMs), including risks linked to malicious use, discrimination, misinformation and human-computer interaction (HCI) [17]. These analyses broaden the discussion of AI risk beyond model accuracy. Other security frameworks are also valuable starting points. NIST's Cybersecurity Framework organizes security activities into five functions: identify, protect, detect, respond, and recover [18]. While not designed specifically for AI-agent memory, the risk-management approach could be used to inform the organization of memory protection controls. Work on adversarial machine learning (AML) also stresses that the attacker's capabilities, access, objectives and assumptions must be stated explicitly when a defense is evaluated [19], [20]. For persistent memory systems, these principles imply that a defense should consider where stored information comes from, where an attacker can manipulate it, and what effects the manipulation can have later. A defense should also consider unintended effects, such as rejecting valid information or lowering normal task performance.

2.4 Memory Integrity, Provenance, and Trust-Aware Defenses

Memory integrity refers to the accuracy, authorization and attribution of stored information. Retrieval-augmented systems rely on external documents; memory-based agents may rely on these as well as on information from past interactions. If a record's source is lost when it is copied, summarized, or transformed, later components may be unable to determine whether it is reliable. The aim of data sanitization is to detect and remove suspicious examples before they affect learning [21]. Influence analysis can be used to explore the influence of individual training examples [10]. Backdoor research shows the importance of evaluating behavior under both ordinary and trigger conditions [11], [12], and backdoor-detection methods such as Neural Cleanse search for hidden triggers [22]. These ideas can be translated conceptually to the memory lifecycle, but more experimental testing is needed to validate their utility in persistent AI-agent memory.

Work closer to this study concerns language-model systems that read untrusted text or keep long-term memory. Perez and Ribeiro showed that instructions inserted into a prompt can hijack a

language model's goal or make it leak its prompt [23]. Du et al. showed that adding fabricated documents to, or modifying documents in, the evidence repository of an automated fact-verification system significantly degrades its performance [24]. Long-term conversational agents already store information from earlier sessions and reuse it later, for example by retrieving or summarizing past conversations [25]. On the defense side, Baracaldo et al. used the provenance of training data to detect and filter poisoned data points [26], which is the closest precedent for M-Guard's provenance-aware records. Together, these studies show that untrusted text can steer language-model systems and that cross-session memory is becoming common, but they do not address poisoned information that persists in an agent's memory across sessions.

Further research published in 2022 provides additional insight into how conversational systems maintain and update long-term memory. Xu et al. [27] introduced PLATO-LTM, a framework that extracts and continuously updates long-term persona information to improve conversational consistency. Bae et al. [28] investigated memory management in long-term conversations and proposed selectively removing outdated or redundant information from stored memory. These studies demonstrate that conversational memory can be maintained and modified across multiple interactions rather than remaining a static collection of records. However, their primary focus is on conversational quality, consistency, and memory management rather than the security implications of malicious information being introduced into persistent memory. In particular, they do not experimentally investigate whether adversarially modified memories can influence unauthorized actions in later sessions or how provenance-based security checks might prevent such behavior.

The combination of prompt manipulation, evidence poisoning, and long-term conversational memory also suggests an important security concern. Information obtained from an external document or other untrusted source may be incorporated into an agent's memory and later retrieved when the original context is no longer available. While Perez and Ribeiro [23] examined adversarial instructions in prompts and Du et al. [24] demonstrated the consequences of manipulated evidence repositories, these studies did not specifically evaluate the persistence of malicious instructions across multiple agent-memory sessions. Similarly, existing conversational memory systems [25], [27], [28] demonstrate mechanisms for retaining and updating information, but do not establish whether stored content remains trustworthy when it originates from an adversarial source. This distinction motivates a defense that evaluates memory throughout its lifecycle, including its origin, storage, retrieval, transformation, and influence on subsequent actions.

This study proposes M-Guard as a layered approach comprising five elements:

Provenance-aware records: Retain information about the source and origin of each memory record.

Write-time validation: Examine candidate records before accepting them into persistent memory.

Trust-aware retrieval: Evaluate a record's trust status in addition to its relevance to the current task.

Temporal revalidation: Reconsider the validity and trustworthiness of stored information as circumstances change.

Execution-time verification: Check whether actions influenced by retrieved memories are authorized and consistent with the task's constraints.

These components are proposed design choices, not established outcomes from the cited studies. Their effectiveness must be assessed experimentally, including whether they reduce malicious behavior without excessively blocking legitimate memory use.

2.5 Research Gap

The survey above provides a number of bases for the study of persistent memory poisoning. Memory architectures illustrate the way information can be stored and subsequently used for computation [1][2]. The retrieval-augmented systems illustrate the effects of external knowledge

on generation [3]. Data-poisoning and backdoor studies show that data can influence model behaviour and can be silent in the context of specific conditions [8], [11], [12], [15]. Language-model risks and cybersecurity studies offer more general principles for the definition and management of system-level threats [16], [17], [18]. Prompt-injection and evidence-poisoning studies show that untrusted text can steer language-model systems [23], [24], long-term conversational agents already reuse information across sessions [25], and data provenance has been used to defend training pipelines against poisoning [26],[27], [28]. However, although earlier studies examined long-term conversational memory, prompt manipulation, evidence poisoning, and data provenance, they did not jointly evaluate an integrated security framework for protecting persistent agent memory against adversarial information across its lifecycle. This study addresses that gap by combining provenance tracking, write-time validation, trust-aware retrieval, temporal revalidation, and execution-time verification. Based on the above, the following research gaps have been identified for this study.

Lifecycle coverage: Existing foundational research commonly examines memory architecture, retrieval, training-data integrity, or model behavior. A persistent-memory defense should connect these concerns across the complete path from memory creation to downstream action.

Temporal persistence: A malicious record may remain stored and influence an agent only when a relevant query or contextual condition appears. Evaluations should distinguish immediate attack success from later retrieval and cross-session behavioral effects.

Provenance and trust: Retrieval relevance does not establish that a record is accurate or safe. A defense should preserve source information and assess trust independently of semantic relevance.

Security–utility balance: More aggressive filtering may reduce malicious influence while also rejecting legitimate information. Evaluations should therefore measure both security outcomes and benign task completion.

Reproducible evaluation: A robust evaluation should report multiple outcomes, including injection success, retrieval of poisoned records, persistence over time, harmful behavioral influence, false positives, and legitimate task completion.

To address these gaps, this paper proposes M-Guard and a multi-session evaluation methodology. Its contribution is a lifecycle-oriented defense design grounded in earlier research on memory, retrieval, poisoning, and AI security. The framework’s performance should be established through experiments rather than assumed from the results of related work.

3. Methodology

3.1 Research Approach

The study follows a design-science approach and evaluates the framework in a controlled simulation. The results reported in this paper come from that simulation, not from a deployment with real AI agents. The research process includes threat identification, development of a framework, controlled test scenarios, comparing to baseline defenses, and threat and utility analysis. The evaluation starts with a reproducible simulator; evaluation with real AI agents is left for future work. The simulator can be used to evaluate the effect of individual aspects of the policy, and a subsequent real-agent evaluation is required to assess the effectiveness of the proposed mechanisms under actual model behavior. The methodology consists of the following four stages:

Threat modeling: Identify attacker capabilities, protected assets and security goals.

Framework construction: Define memory provenance, validation rules, retrieval checks and run-time safeguards.

Controlled evaluation: Run each attack family against the baseline and proposed defenses.

Analysis: Compute attack success rate, detection recall, task completion rate, persistence, and

false-positive rate.

3.2 Threat Model

The intended system is an AI agent that can store information from previous interactions and recall it in later ones.

The key assets to be protected are:

Integrity of stored memory records.

Accuracy of remembered user preferences and project information.

Safety of future agent actions.

Confidentiality of user data.

The integrity of tool calls and other resulting actions.

Ability to provide legitimate memory-dependent functionality.

Attacker capabilities

The attacker is assumed to be able to place misleading or malicious content where the agent can access it, for instance in a webpage, document, email, repository, or user-visible interaction. The attacker can try to get the agent to summarise, store, or recall that information. In the base threat model, the attacker has no control over the trusted system policy, or access control infrastructure of the memory database. For more sophisticated experiments, these assumptions can be dropped to explore compromised connectors, manipulated metadata, or a coordinated attack.

Security assumptions

The framework assumes that the memory guard, provenance registry, and tool-execution controls are all correct. All of these components are included in the trusted computing base. The framework does not require an AI model to perform a complete semantic analysis of every malicious instruction. It also does not guarantee that the information received from a trusted source is true or harmless.

3.3 Attack Taxonomy

The proposed evaluation includes four attack categories.

A1: Direct memory injection. The attacker tries to get a destructive command or incorrect fact to be recorded as a memory item.

A2: Multi-record contamination. Several records are added or modified in such a way that when these changes are added together, they affect later action by an agent, although each might seem innocuous on its own.

A3: Delayed or trigger-based poisoning. A record is stored in one interaction but affects behavior in a later interaction only when a trigger appears, such as a particular task, keyword, project state, or context.

A4: Provenance laundering. Untrusted content is summarised, copied or transformed in a manner that conceals its origin or makes it appear more authoritative than it is.

These categories are intended for controlled security testing. The experiment should be sandboxed and based on mock tools, to ensure the attacks do not impact real accounts, systems, or personal data.

3.4 Proposed M-Guard Framework

M-Guard is Memory Guard with Provenance, Validation, and Temporal Reassessment. It is a proposed defense architecture of five complementary components.

Component 1: Provenance-aware memory records

For each memory record, there is accompanying metadata that records where it came from, its creation time, its confidence score, how the record was transformed, and whether it had been validated. For instance, if an agent summarizes a record that was extracted from an external web page, that record will still be classified as externally sourced. A memory record may be represented

as:

$$m_i = (c_i, o_i, t_i, \gamma_i, h_i, s_i)$$

where:

- c_i is the stored content;
- o_i is the origin or provenance label;
- t_i is the creation time;
- γ_i is the confidence or validation score;
- h_i is the transformation history;
- s_i is the current security status.

Provenance metadata must be maintained by the memory infrastructure rather than accepted solely from the AI's self-reported explanation.

Component 2: Memory-write validation

The write gate filters the record before it is written to memory. Validation checks for instructions unrelated to the user's task, claims the system cannot support, attempts to override system behavior, and requests for actions the user is not authorized to perform. The gate returns one of three decisions:

- Accept: Save the record, along with the provenance and validation metadata.
- Quarantine: Keep record apart for further examination and do not use as trusted memory.
- Reject: Do not store the record if it violates a defined security policy.

The validation process should also check whether the record conflicts with high-confidence memories. If there is a conflict, this does not necessarily mean the new record is malicious, but rather that there may be a need to do further verification.

Component 3: Trust-aware retrieval

When a later task retrieves relevant memories, the retrieval layer considers both relevance and trust. A semantically relevant record is not automatically authorized to influence every type of decision. A simple ranking function is:

$$R(m_i, q) = \alpha S_{rel}(m_i, q) + \beta S_{trust}(m_i) + \delta S_{fresh}(m_i)$$

where $S_{rel}(m_i, q)$ measures relevance to query (q), $S_{trust}(m_i)$ represents trust evidence, and $S_{fresh}(m_i)$ represents temporal freshness. The coefficients α, β, δ are design parameters that must be calibrated experimentally. Security checks must remain independent of ranking: a highly relevant but prohibited memory should not become admissible simply because its relevance score is high.

Component 4: Temporal revalidation

Stored memories are re-evaluated when their age, sensitivity, provenance, or intended use warrants it. High-impact records, such as payment instructions or permission changes, should be validated more rigorously than low impact preferences like preferred document formatting. Temporal decay can reduce the influence of stale data, but age alone is not a good indicator of quality. A long-standing preference should not be ignored simply because it is old, and a new record should not be trusted simply because it is new.

Component 5: Execution-time verification

An agent that wants to take a "consequential action" must check with the runtime guard whether the action can be performed based on the user's request and authorization. A memory record can inform a proposal, but it cannot grant new permissions. For instance, a stored preference might recommend a specific email format, but a remembered instruction should not automatically allow the sending of confidential data to a new recipient. The execution layer may allow low risk actions to be executed, ask for confirmation of ambiguous or high impact actions, and deny actions that are prohibited by an explicit policy.

3.5 Framework Workflow

The following is the process for the entire workflow:

1. Receive input from the user or external content.
2. Record the source of the input.
3. Identify proposed memory writes.
4. Accept, quarantine or reject each candidate record.
5. Retrieve relevant records for a later task.
6. Re-evaluate trust, freshness and potential conflicts.
7. Generate the agent's proposed response or action.
8. Check the proposed action against the current task and authorization policy.
9. Log the decision for audit and future evaluation.

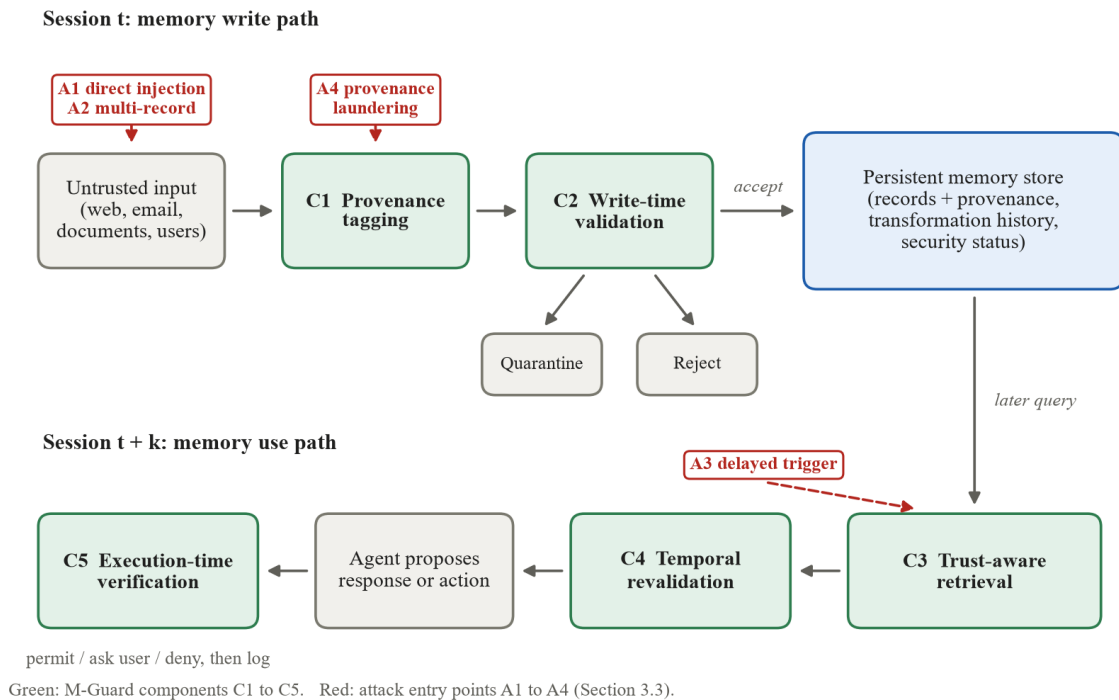


Figure 1. Persistent Memory Poisoning and Defense Lifecycle

3.6 Algorithmic Decision Procedure

The following pseudocode summarizes the proposed defense.

Algorithm 1: M-Guard Memory and Action Validation

Input: Candidate memory m , current task q , proposed action a

1. Register provenance and transformation history of m
2. Evaluate content, integrity, and policy compliance
3. If m violates an explicit policy:
 - Reject m
4. Else if m requires further verification:
 - Quarantine m
5. Else:
 - Store m with provenance metadata
6. When task q arrives:
 - Retrieve relevant memories
 - Reassess trust, freshness, and conflicts
 - Exclude or quarantine inadmissible memories

7. Generate proposed action a
8. Verify a against the current user request and policy
9. If a is unauthorized:
 - Deny a
10. Else if a requires confirmation:
 - Ask the user
11. Else:
 - Permit a and log the decision

Its effectiveness depends on the reliability of the provenance registry, validation logic, memory isolation, and execution guard.

3.7 Experimental Environment and Dataset Construction

The evaluation takes place in a controlled multi-session setting with benign and adversarial tasks. Synthetic user profiles, mock documents, isolated memory stores, and simulated tool calls are used in the environment. The simulation uses 1,000 task trajectories, split into 500 benign trajectories and 500 adversarial trajectories. Each trajectory includes more than one session so that memory written in one session can be tested in later sessions; adversarial trajectories are followed for ten sessions after the initial injection and evaluated at checkpoints 1, 2, 3, 5 and 10. Across all trajectories, the simulator generated 3,250 candidate memory records (1,250 malicious and 2,000 benign), which are the denominators of the record-level detection metrics in Section 5.4. For the real-agent evaluation, the sample size should be confirmed through a pilot study and a power analysis rather than chosen to produce desired results.

- Each trajectory should record:
 - The initial user goal.
 - Each candidate memory and its source label.
 - The validation decision.
 - The results of each subsequent session.
 - Whether the poisoned content influenced the generated behavior.
 - Whether or not a consequential action was tried or performed.
 - Whether the legitimate task was completed.
 - Any false positives or false negatives.

The dataset should include distinct training or calibration, validation, and held-out test partitions. Near-duplicate scenarios should not be allowed to leak across these partitions.

3.8 Baseline Methods

Five configurations are compared under the same scenario set.

Configuration	Description
B0: Unrestricted memory	Stores and retrieves records without dedicated poisoning defenses.
B1: Prompt-level filtering	Attempts to detect malicious instructions in the current context but does not independently protect persistent memory.
B2: Write-time filtering	Evaluates candidate records before storage but does not apply a dedicated retrieval-time trust check.
B3: Trust-aware retrieval	Evaluates the origin and trust status of retrieved records but has limited write-time validation.
M-Guard	Combines provenance tracking, write-time validation, trust-aware retrieval, temporal revalidation, and execution-time verification.

These are experimental configurations, not claims that all named methods correspond to specific published systems. Baseline implementations must be documented precisely so that the comparison can be reproduced.

3.9 Evaluation Metrics

The evaluation should report multiple complementary metrics.

Attack Success Rate (ASR): the percentage of adversarial trajectories in which the attack achieves its harmful objective. A trajectory is counted as successful if the unauthorized action is executed at least once in any session up to the final checkpoint (session 10). Each trajectory contributes exactly one outcome, so the denominator is the number of adversarial trajectories (500 per configuration), not the number of session-level attempts.

$$ASR = \frac{N_{\text{successful adversarial trajectories}}}{N_{\text{total adversarial trajectories}}} \times 100\%$$

Memory Injection Success Rate (MISR): This is the percentage of memory injection attempts that inject a poisoned memory into persistent memory.

Cross-Session Attack Success Rate (C-ASR@k): Cumulative percentage of adversarial trajectories in which the targeted harmful behavior, caused by poisoning in an earlier session, has occurred at least once by checkpoint k. Because it counts first successes, C-ASR@k cannot decrease as k grows, and its value at the final checkpoint equals the overall ASR.

Poisoned Memory Persistence (PMP): Percentage of the initially accepted poisoned records that are either retained or influential after a certain number of following sessions. Behavioral influence and physical record retention should be assessed separately.

Detection Recall: Percentage of malicious records that the defense correctly detects.

False-Positive Rate (FPR): Percentage of benign records misclassified as malicious.

Benign Task Completion Rate (BTCR): Percentage of legitimate tasks completed successfully.

Utility Retention: legitimate task completion as a proportion of task completion in the baseline with no defense.

$$UR = \frac{BTCR_{\text{defense}}}{BTCR_{\text{baseline}}} \times 100\%$$

These measures prevent a defense from appearing effective merely because it blocks nearly all memory use. A useful defense should reduce harmful behavior while preserving legitimate functionality.

3.10 Statistical Analysis

Every rate is reported as a count over its stated denominator, together with a 95% Wilson score confidence interval. Each configuration was evaluated once on the same seeded set of trajectories, so no averaging across repetitions is involved. Because all configurations share the same trajectories, comparisons between configurations are paired: differences in attack success and benign completion are reported with 95% paired Wald confidence intervals and tested with exact McNemar tests, and the four comparisons between M-Guard and the baselines are corrected for multiplicity with the Holm method. Task completion should be measured separately from attack success in order to evaluate benign task completion. A granular score should also be reported per attack category and session horizon besides relying only on the overall score. When multiple AI backbones are tested, model identity and task category should be included as factors in the analysis. Where several comparisons are made, a correction for multiple comparisons should be applied.

4. Experimental Setup and Evaluation Design

4.1 Experimental Conditions

The experiment compares a defense that protects the whole memory lifecycle with defenses that act at a single point. The simulation employs 1,000 multi-session trajectories (500 benign and 500 adversarial). In each trajectory, there is a series of interactions that allow the agent to write new memories, retrieve memories from the past and/or perform a task. For the real-agent evaluation,

the number of trajectories, attack repetitions, and the length of a session should be determined by pilot testing. The test harness should capture four phases for each adversarial trajectory:

- Injection: Whether the attack successfully influences a candidate memory write.
- Persistence: Whether the poisoned record remains available in later sessions.
- Retrieval: Whether the record is retrieved for use in another task.
- Behavioral influence: Whether a retrieved record causes the agent to deviate from the predetermined task objective or security policy.

This decomposition matters because an attack may write a record without causing harmful action. Conversely, a low injection rate is not necessarily safe if the few records that are stored have a large downstream effect. The evaluation examines persistent memory poisoning across multiple sessions following an initial memory injection. Attack success is recorded cumulatively at predefined session checkpoints to assess when unauthorized behavior first occurs. The experiment does not include a separate condition in which attacks are restricted to a single session. Consequently, the temporal analysis characterizes cross-session attack behavior but does not establish a comparative advantage or disadvantage relative to independent single-session attacks.

4.2 Experimental Scenarios

Table 1. Distribution of simulated trajectories.

Scenario	Benign trajectories	Adversarial trajectories	Attack types tested	Primary objective
Preference and profile memory	100	100	A1, A4	Detect false preferences and fabricated user instructions
Project and document memory	100	100	A1, A4	Evaluate misleading historical information
Tool-use and workflow memory	100	100	A1, A4	Prevent unauthorized future actions
Multi-record contamination	100	100	A2	Evaluate interactions between individually plausible records
Delayed or trigger-based memory	100	100	A3	Measure behavior across later sessions
Total	500	500	A1–A4	

The benign scenarios reflect typical memory function, such as recalling preferences, maintaining project continuity, and remembering past decisions. In adversarial scenarios, controlled misinformation or unauthorized instructions are added to the same kinds of workflows. The test harness should be set up so that the harmful objective is stated explicitly before any experiment is run, and success is judged from observable task outcomes rather than by another AI model's judgment. Table 1 also shows which attack types (Section 3.3) each scenario tests. Direct memory injection (A1) and provenance laundering (A4) do not have separate scenarios: in each of the three memory-type scenarios, 50 adversarial trajectories use direct injection and 50 use laundered

content. Multi-record contamination (A2) and delayed or trigger-based poisoning (A3) have their own scenarios. Each attack type is therefore tested on 150 (A1, A4) or 100 (A2, A3) adversarial trajectories.

4.3 Reproducibility and Implementation

The simulator was implemented in Python using only the standard library. All results in Section 5 come from a single run with random seed 20261024. The run evaluated ten policy configurations (the five in Table 2 and the five ablations in Table 4) on the same 500 adversarial and 500 benign trajectories, giving 10,000 trajectory outcomes and 21,679 logged events. Each configuration was evaluated once, so every percentage in Section 5 is a count of trajectories or records divided by the stated denominator; no averaging across repetitions was performed. The run produced a machine-readable trajectory log with one row per configuration and trajectory, recording the attack family, the session of first success, the memory-write decision, the retrieval outcome, the action verdict, the stage at which the attack was stopped, and the binary attack and benign-completion outcomes. It also produced a summary log with the counts behind Tables 2–6 and automated consistency checks, which confirmed that the attack-family counts sum to the overall counts, that the record-level false-positive rate equals the M-Guard value in Table 2, that the final cumulative checkpoint in Table 6 equals the overall attack success rate, and that every successful unsafe action corresponds to an event trace. Future model-based testing should additionally document the following information:

- The model and version number.
- System and agent prompts.
- Memory storage and retrieval settings.
- Embedding model and retrieval parameters (if applicable).
- Validation thresholds and policy rules.
- Number of sessions and number of repetitions of attacks.
- Tool permissions and execution environment.
- Random seeds, if available.
- Evaluation scripts and outcome labels.

A fixed seed doesn't make stochastic AI inference deterministic; it only helps in reproducing the synthetic generator. Therefore, model-based results should be reported only for repeated trials, together with the execution settings.

4.4 Ethical and Safety Considerations

All experiments were performed in a controlled, simulated environment. Because all tool calls were simulated, no attack resulted in an actual payment, a sent email, a deleted file or a changed permission. All simulated users and documents were synthetic. In future real-agent experiments, attack payloads should only include what is necessary for the controlled test cases and any actions generated by models should go through an execution layer that is sandboxed. The evaluation also examines whether a defense blocks accurate memories, because a security mechanism that removes useful memory functionality would not be adequate for general-purpose use.

5. Simulation Results and Analysis

5.1 Main Comparison of Defense Configurations

Table 2. Simulation results for the main security–utility comparison. Values are % (count)

Configuration	Attack success rate, % (n/500) ↓	Benign task completion, % (n/500) ↑	False-positive rate, % (n/2,000) ↓
B0: Unrestricted	72.4 (362) [68.3–	94.2 (471) [91.8–	0.0 (0) [0.0–0.2]

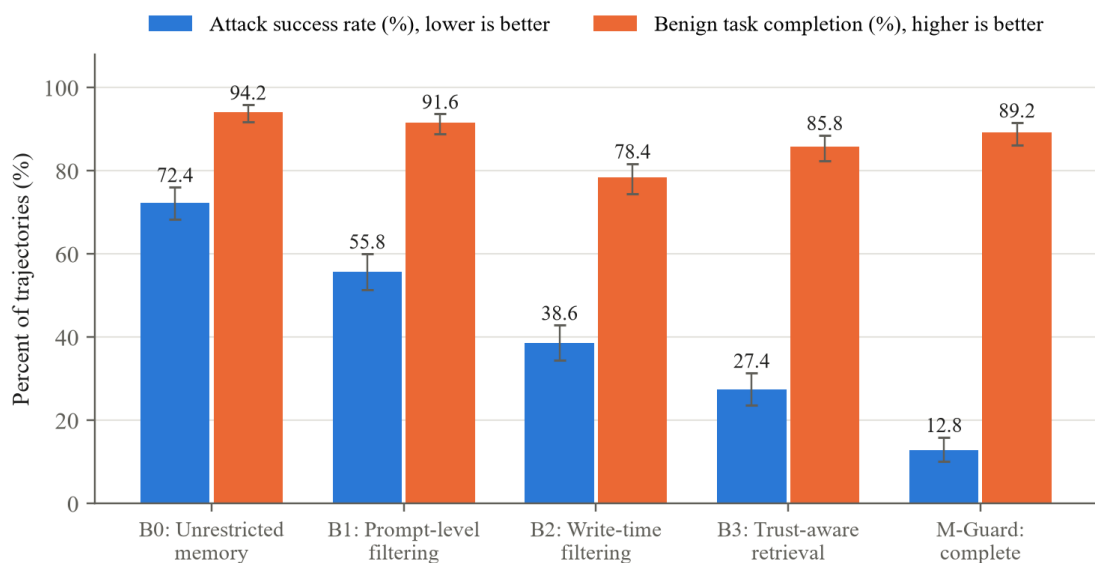
memory	76.1]	95.9]	
B1: Prompt-level filtering	55.8 (279) [51.4–60.1]	91.6 (458) [88.8–93.7]	6.8 (136) [5.8–8.0]
B2: Write-time filtering	38.6 (193) [34.4–42.9]	78.4 (392) [74.6–81.8]	15.2 (304) [13.7–16.8]
B3: Trust-aware retrieval	27.4 (137) [23.7–31.5]	85.8 (429) [82.5–88.6]	9.6 (192) [8.4–11.0]
M-Guard: Complete framework	12.8 (64) [10.2–16.0]	89.2 (446) [86.2–91.6]	7.4 (148) [6.3–8.6]

The simulation results show a trade-off between security and legitimate functionality. The unrestricted baseline has high benign task completion and a high attack success rate. Prompt-level filtering offers a modest reduction in attack success, whereas write-time filtering yields a substantial reduction in attack success—at the cost of more benign task failures. In the simulation, M-Guard has an attack success rate of 12.8% and unrestricted memory 72.4%, a relative reduction of about 82.3%:

$$Relative\ Reduction = \frac{72.4 - 12.8}{72.4} \times 100\%$$

$$Relative\ Reduction \approx 82.3\%$$

Meanwhile, M-Guard keeps a benign task completion rate of 89.2%. This suggests that a defense need not block every memory operation to reduce harmful behavior. Because every configuration was evaluated on the same trajectories, the comparison is paired. M-Guard's attack success rate was 59.6 percentage points lower than that of unrestricted memory (95% CI 55.3–63.9; exact McNemar $p < 0.001$) and 14.6 points lower than that of the strongest single-point baseline, trust-aware retrieval (95% CI 11.5–17.7; $p < 0.001$); all four baseline comparisons remain significant after Holm correction. The utility cost was a benign completion rate 5.0 points lower than unrestricted memory (95% CI 1.6–8.4; Holm-adjusted $p = 0.016$), while M-Guard completed 10.8 points more benign tasks than write-time filtering (95% CI 6.4–15.2). These results come from a simulation, so they do not show that M-Guard outperforms the baselines with real AI agents; matched experiments with real agents are needed to make such a conclusion.



Simulation results ($n = 500$ per bar; error bars: 95% Wilson CI), not real-agent deployment

Figure 2. Simulated attack success rate and benign task completion for the five configurations, with 95% Wilson confidence intervals

5.2 Analysis of Attack Categories

The overall attack success rate can mask large differences between attack families. While multi-record contamination or delayed activation can be hard to detect even if each individual record is initially validated, direct memory injection could be relatively easy to detect.

Table 3. Simulated attack success rate by attack family, % (successful/adversarial trajectories).

Attack family	Unrestricted memory ASR, % (n/N)	M-Guard ASR, % (n/N)	Hypothesis to test
Direct memory injection	77.3 (116/150)	7.3 (11/150)	Write-time validation may block many direct attempts
Multi-record contamination	68.0 (68/100)	18.0 (18/100)	Interactions between records may leave residual risk
Delayed or trigger-based poisoning	70.0 (70/100)	16.0 (16/100)	Later retrieval may require additional safeguards
Provenance laundering	72.0 (108/150)	12.7 (19/150)	Transformation history may be important for preserving trust
Overall (weighted by trajectories)	72.4 (362/500)	12.8 (64/500)	

The overall row of Table 3 weights each attack family by its number of adversarial trajectories (150 each for A1 and A4, 100 each for A2 and A3; see Table 1), so it equals the overall values in Table 2. In the simulation, direct injection was the easiest attack to prevent (7.3%), possibly because suspicious content can be detected during memory creation. Multi-record contamination, on the other hand, was the hardest to prevent (18.0%), likely because its effect does not become apparent until multiple memory records are combined. Each family estimate rests on only 100 or 150 trajectories, so its confidence interval is wide (for example, 11.7–26.7% for multi-record contamination under M-Guard), and the simulation does not statistically distinguish the residual risk of multi-record, delayed, and laundered attacks from one another. There is also a difference in evaluation strategy for delayed attacks. A seemingly innocuous memory from the first session can have an impact when a second task calls it up. Evaluating only immediate responses would therefore miss the delayed damage that persistent memory poisoning can cause. Another challenge is 'provenance laundering'. A record can be summarized or copied in a way that hides its original source. If the system treats a transformed record as agent-generated, the defense loses the information it needs to determine where the record came from.

5.3 Ablation Study

An ablation study removes one defense component at a time while keeping the scenario distribution and other conditions unchanged. This helps identify which components contribute to the overall result.

Table 4. Simulated ablation results. Values are % (count out of 500 trajectories).

Configuration	Attack success rate, % (n/500) ↓	Benign completion, % (n/500) ↑
Complete M-Guard	12.8 (64)	89.2 (446)
Without provenance tracking	30.4 (152)	91.0 (455)

Without write-time validation	26.2 (131)	92.4 (462)
Without trust-aware retrieval	27.8 (139)	90.2 (451)
Without temporal revalidation	18.8 (94)	89.6 (448)
Without execution-time verification	25.6 (128)	90.8 (454)

The ablation results show how much each component contributes. Removing provenance tracking had the largest effect, raising attack success from 12.8% to 30.4% (paired difference 17.6 points, 95% CI 14.3–20.9), while removing temporal revalidation had the smallest (18.8%; difference 6.0 points, 95% CI 3.9–8.1). Every single-component removal significantly increased attack success (exact McNemar $p < 0.001$ in each case), whereas no removal changed benign completion significantly (all $p > 0.1$). The intervals for removing write-time validation, trust-aware retrieval, and execution-time verification overlap, so the simulation does not rank those three components against each other. Eliminating provenance tracking likely reduces the system's ability to differentiate between information obtained externally and validated memories. Without write-time validation, more poisoned records may be stored; without retrieval-time checks, more previously stored records may influence later decisions without being rechecked. Temporal revalidation might matter less for recent attacks and more for old, stale or repeated records. Execution-time verification can be used to prevent a harmful action even after previous memory controls have failed. These mechanisms also overlap: a component may help on its own but contribute more in combination with others. Ablation experiments with real agents must therefore measure interactions between components rather than assume that their effects are additive.

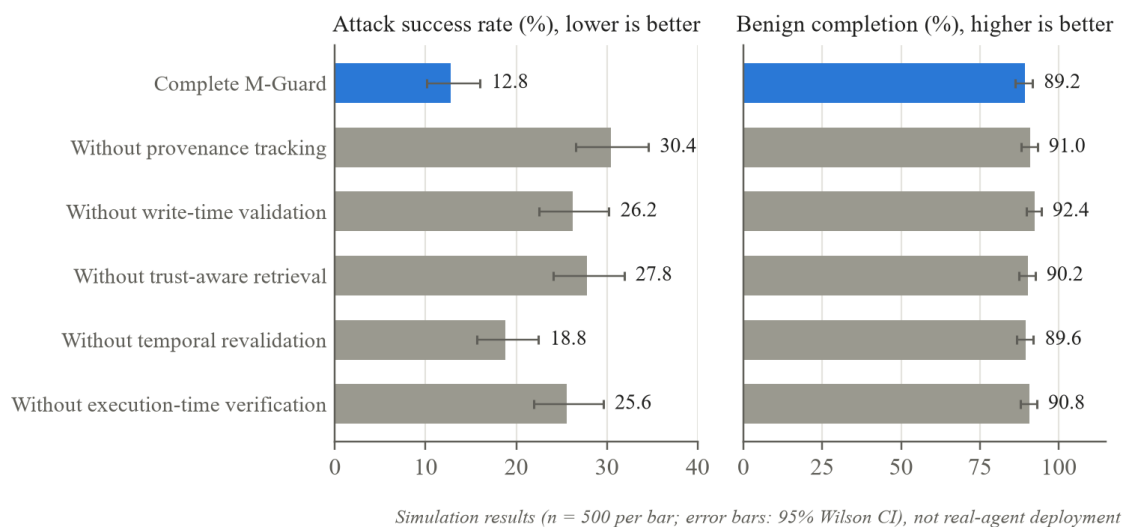


Figure 3. Simulated attack success rate and benign completion for the complete framework

5.4 Detection Performance and False Positives

A defense that detects malicious records accurately but rejects many legitimate memories may impair the primary purpose of persistent memory. The detection results must therefore include both malicious and benign cases.

Table 5. Simulated detection metrics.

Metric	Value (count) [95% Wilson CI]
Detection recall	91.2% (1,140/1,250) [89.5–92.6]
Precision	88.5% (1,140/1,288) [86.7–90.1]
False-positive rate	7.4% (148/2,000) [6.3–8.6]
F1-score	89.8%

Precision measures the proportion of flagged records that are actually malicious, while recall measures the proportion of malicious records that are identified. The F1-score is the harmonic mean of precision and recall.

$$F1\ Score = \frac{2 \times (precision \times recall)}{(precision + recall)}$$

These results illustrate that more than one metric can be used to evaluate a defense. However, the proportion of malicious records in the test set strongly affects precision and accuracy, whereas recall and the false-positive rate do not depend on it. The record-level confusion matrix for M-Guard is TP = 1,140, FN = 110, FP = 148, and TN = 1,852, computed over 1,250 malicious and 2,000 benign records (38.5% malicious). Also, not all records that are suspicious need to be refused immediately. A quarantine mechanism can be used to maintain potentially useful information but block it from impacting on sensitive tasks until its source or accuracy is verified. The detection metrics in Table 5 are computed per memory record, whereas the attack success rate in Table 2 is computed per trajectory. An attack that spreads across several records can succeed if any one of them is missed, so the two rates are not expected to be complementary.

5.5 Temporal Persistence Analysis

Persistent memory poisoning is fundamentally a cross-session threat. A defense should therefore measure whether poisoned information continues to influence behavior over time.

Table 6. Simulated cumulative cross-session attack success

Checkpoint (session after initial injection)	Unrestricted memory cumulative ASR, % (n) [95% CI]	M-Guard cumulative ASR, % (n) [95% CI]
Session 1	36.0 (180) [31.9–40.3]	2.0 (10) [1.1–3.6]
Session 2	50.0 (250) [45.6–54.4]	4.4 (22) [2.9–6.6]
Session 3	61.0 (305) [56.7–65.2]	7.2 (36) [5.2–9.8]
Session 5	68.2 (341) [64.0–72.1]	10.4 (52) [8.0–13.4]
Session 10	72.4 (362) [68.3–76.1]	12.8 (64) [10.2–16.0]

Table 6 reports cumulative attack success: the percentage of adversarial trajectories in which an unauthorized action had been executed at least once by each checkpoint. Session-level rates are not averaged. Under unrestricted memory, cumulative attack success rises from 36.0% at session 1 to 72.4% at session 10, whereas under M-Guard it rises from 2.0% to 12.8%. Because the measure is cumulative and each trajectory contributes one outcome, the session-10 values equal the overall attack success rates reported in Table 2.

The cumulative results demonstrate the importance of evaluating persistent memory poisoning

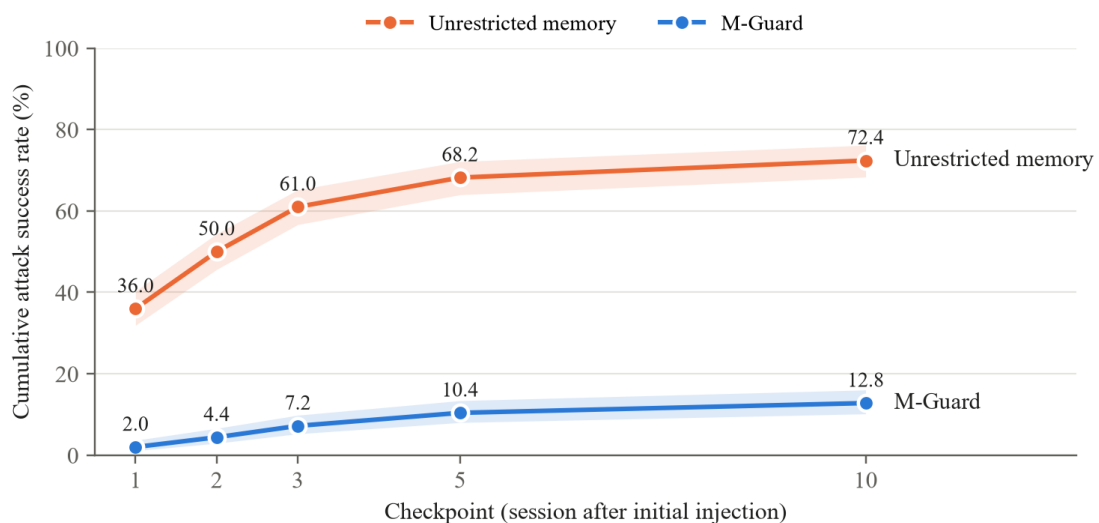
beyond the initial session. Under unrestricted memory, 182 of the 362 successful adversarial trajectories first achieved their harmful objectives after session 1. Under M-Guard, 54 of the 64 successful trajectories first succeeded after session 1. These outcomes show that unauthorized behavior may emerge during later interactions, even when an attack does not succeed immediately. However, the timing of first success should be distinguished from membership in the delayed or trigger-based attack category (A3). Since the study does not include an independent single-session attack condition, the results characterize the temporal progression of persistent poisoning rather than directly comparing persistent and single-session attack strategies.

Real agents may not show the same pattern. Some memory systems delete stale records, while others summarize information into embeddings that are harder to inspect. The relationship between elapsed sessions and attack success therefore needs to be evaluated separately for each type of memory architecture. The simulation results indicate the importance of observing agent behavior across multiple sessions, but they do not establish how different memory architectures would behave under equivalent attacks.

Future experiments should be able to separate three quantities:

- Record persistence: Whether the poisoned record still exists.
- Retrieval persistence: Whether the record is still retrieved.
- Behavioral persistence: Whether the record still influences the agent's decisions.

The quantities may vary. A record may be permanently stored but never be retrieved, or it may be deleted while a summary derived from it remains.



Simulation results (n = 500 adversarial trajectories; bands: 95% Wilson CI), not real-agent deployment

Figure 4. Simulated cumulative attack success rate by checkpoint for unrestricted memory and M-Guard, with 95% Wilson confidence bands

6. Discussion

6.1 Persistent Memory as a Security Boundary

The main idea behind this research is to consider persistent memory as a security-sensitive element of the agent architecture. Memory is not just a passive database; it affects what the agent deems relevant, what it assumes about the user, and which actions it suggests. A conventional prompt-injection defense might not work if the malicious information has already been stored and the original attack context is no longer in existence. In these instances, the security challenge becomes one of identifying the origin, integrity and authorized use of a continuing record as opposed to simply detecting a malicious prompt. The proposed M-Guard framework is designed to address this challenge by extending controls at multiple stages of the memory lifecycle. It is not designed

to guarantee that every malicious record is detected or that no malicious record is ever used.

6.2 Why a Layered Defense Is Necessary

Memory-write validation can minimize direct contamination, but it might not be able to detect records that look harmless on their own. Trust-aware retrieval can re-evaluate stored information when it becomes relevant to a new task. Temporal revalidation can be useful for finding stale and re-used records, and execution-time verification can keep a bad proposal from turning into a bad action. These layers cover various modes of failure. Their combined value should be determined experimentally rather than assumed from the design. The framework also separates content relevance from security authorization. A memory can be very relevant to the task at hand, but not reliable enough to justify an irreversible action.

6.3 Security–Utility Trade-Off

A very strict memory policy can reduce poisoning risk by filtering out most incoming information, but the same policy can also prevent personalization and force users to repeat their preferences. The appropriate balance depends on the risk of the intended operation. A remembered formatting preference can be handled under a relatively permissive policy, whereas a stored instruction involving financial transactions, confidential information or account permissions should require stronger verification. Therefore, policies for memory systems should not be "all-or-nothing" policies, but instead should be risk-sensitive. The metrics proposed for evaluation are designed to include both sides of this trade-off.

6.4 Provenance and Trust Propagation

Provenance tracking is useful only if source information is preserved throughout the memory lifecycle. When a malicious record is summarized, merged with other records, or copied to a different memory store, its original source label should be carried forward rather than lost because the content has been rewritten. At the same time, provenance does not prove that content is true. Records from a legitimate source may still be inaccurate, and several records may repeat the same underlying claim without adding independent support. A stronger design should therefore keep source authenticity, factual confidence, authorization, and relevance as separate properties, since they are distinct concepts.

6.5 Limitations

Several limitations must be considered before drawing conclusions about real-world agent security. All results in this paper come from a controlled simulation rather than from deployed AI agents, so they show how the defense behaves under the simulator's assumptions, not how it performs in practice. Different AI models can respond differently to external content, system instructions, and recalled memories, so the defense may behave differently across model families. A limited set of synthetic scenarios cannot represent every memory architecture, usage pattern, or adaptive attack. The framework assumes that there is a proper and accurate recording and maintenance of provenance metadata. This assumption could be wrong if the registry were corrupted or a non-trusted memory connector was used. Legitimate and malicious content can use the same wording, so automated validation may wrongly flag benign records or miss subtle attacks. Revalidation, provenance management, and other policy checks can add latency and resource usage. These costs must be measured in an implemented system. Over time, an agent's memory could become condensed, combined, embedded, and/or reformulated. The framework must handle these transformations, and not assume that every memory is a standalone inspectable record. Finally, an adaptive attacker who knows M-Guard's policies could craft content specifically to pass its checks, so future evaluations should include adaptive attacks.

7. Conclusion

Persistent memory allows AI agents to carry context across conversations, but it also introduces a security risk that extends beyond the prompt. Malicious information stored in an earlier session may be recovered later, influence decisions and even lead to unauthorized activities even after the malicious content is no longer present in the conversation. This paper introduced a threat model for persistent memory poisoning, and proposed M-Guard, a layered framework combining provenance-aware memory records, write-time validation, trust-aware retrieval, temporal revalidation, and execution-time verification. A multi-session evaluation methodology was established to quantify attack success, memory injection, persistence, detection, false positives, and successful completion of legitimate tasks. In the simulation, M-Guard reduced the attack success rate from 72.4% to 12.8% (paired difference 59.6 percentage points, 95% CI 55.3–63.9) while keeping benign task completion at 89.2%. These results come from a simulation rather than a deployment; evaluating M-Guard with real AI agents is the necessary next step before any claim about its effectiveness in practice can be made.

References

- [1] J. Weston, S. Chopra, and A. Bordes, “Memory networks,” *International Conference on Learning Representations (ICLR)*, 2015. Available: <https://arxiv.org/abs/1410.3916>
- [2] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A. P. Badia, K. Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, and D. Hassabis, “Hybrid computing using a neural network with dynamic external memory,” *Nature*, vol. 538, pp. 471–476, 2016. doi: 10.1038/nature20101.
- [3] P. Lewis et al., “Retrieval-augmented generation for knowledge-intensive NLP tasks,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020. Available: <https://arxiv.org/abs/2005.11401>
- [4] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang, “REALM: Retrieval-augmented language model pre-training,” *Proceedings of the 37th International Conference on Machine Learning*, PMLR, vol. 119, pp. 3929–3938, 2020. Available: <https://arxiv.org/abs/2002.08909>
- [5] V. Karpukhin et al., “Dense passage retrieval for open-domain question answering,” *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, 2020. doi: 10.18653/v1/2020.emnlp-main.550.
- [6] G. Izacard and E. Grave, “Leveraging passage retrieval with generative models for open domain question answering,” *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 874–880, 2021. doi: 10.18653/v1/2021.eacl-main.74.
- [7] S. Borgeaud et al., “Improving language models by retrieving from trillions of tokens,” *Proceedings of the 39th International Conference on Machine Learning*, PMLR, vol. 162, pp. 2206–2240, 2022.
- [8] B. Biggio, B. Nelson, and P. Laskov, “Poisoning attacks against support vector machines,” *Proceedings of the 29th International Conference on Machine Learning (ICML)*, 2012. Available: <https://icml.cc/2012/papers/880.pdf>
- [9] J. Steinhardt, P. W. Koh, and P. S. Liang, “Certified defenses for data poisoning attacks,” *Advances in Neural Information Processing Systems*, vol. 30, 2017. Available: <https://arxiv.org/abs/1706.03691>
- [10] P. W. Koh and P. Liang, “Understanding black-box predictions via influence functions,” *Proceedings of the 34th International Conference on Machine Learning*, PMLR, vol. 70, pp. 1885–1894, 2017. Available: <https://arxiv.org/abs/1703.04730>
- [11] T. Gu, B. Dolan-Gavitt, and S. Garg, “BadNets: Identifying vulnerabilities in the machine learning model supply chain,” arXiv preprint arXiv:1708.06733, 2017. Available: <https://arxiv.org/abs/1708.06733>
- [12] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, “Targeted backdoor attacks on deep learning

- systems using data poisoning,” arXiv preprint arXiv:1712.05526, 2017. Available: <https://arxiv.org/abs/1712.05526>
- [13] J. Ebrahimi, A. Rao, D. Lowd, and D. Dou, “HotFlip: White-box adversarial examples for text classification,” *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 31–36, 2018. doi: 10.18653/v1/P18-2006.
 - [14] E. Wallace, S. Feng, N. Kandpal, M. Gardner, and S. Singh, “Universal adversarial triggers for attacking and analyzing NLP,” *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 2153–2162, 2019. doi: 10.18653/v1/D19-1221.
 - [15] K. Kurita, P. Michel, and G. Neubig, “Weight poisoning attacks on pretrained models,” *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2793–2806, 2020. doi: 10.18653/v1/2020.acl-main.249.
 - [16] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, “On the dangers of stochastic parrots: Can language models be too big?,” *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pp. 610–623, 2021. doi: 10.1145/3442188.3445922.
 - [17] L. Weidinger et al., “Ethical and social risks of harm from language models,” arXiv preprint arXiv:2112.04359, 2021. Available: <https://arxiv.org/abs/2112.04359>
 - [18] National Institute of Standards and Technology, *Framework for Improving Critical Infrastructure Cybersecurity*, Version 1.1, 2018. Available: <https://doi.org/10.6028/NIST.CSWP.04162018>
 - [19] B. Biggio and F. Roli, “Wild patterns: Ten years after the rise of adversarial machine learning,” *Pattern Recognition*, 2018. doi: 10.1016/j.patcog.2018.07.023.
 - [20] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin, “On evaluating adversarial robustness,” arXiv preprint arXiv:1902.06705, 2019. Available: <https://arxiv.org/abs/1902.06705>
 - [21] G. F. Cretu, A. Stavrou, M. E. Locasto, S. J. Stolfo, and A. D. Keromytis, “Casting out demons: Sanitizing training data for anomaly sensors,” *2008 IEEE Symposium on Security and Privacy*, 2008. doi: 10.1109/SP.2008.11.
 - [22] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, “Neural Cleanse: Identifying and mitigating backdoor attacks in neural networks,” *2019 IEEE Symposium on Security and Privacy*, pp. 707–723, 2019.
 - [23] F. Perez and I. Ribeiro, “Ignore previous prompt: Attack techniques for language models,” *NeurIPS 2022 Workshop on Machine Learning Safety*, 2022. Available: <https://arxiv.org/abs/2211.09527>
 - [24] Y. Du, A. Bosselut, and C. D. Manning, “Synthetic disinformation attacks on automated fact verification systems,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 10, pp. 10581–10589, 2022. doi: 10.1609/aaai.v36i10.21302.
 - [25] J. Xu, A. Szlam, and J. Weston, “Beyond goldfish memory: Long-term open-domain conversation,” *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, pp. 5180–5197, 2022. doi: 10.18653/v1/2022.acl-long.356.
 - [26] N. Baracaldo, B. Chen, H. Ludwig, and J. A. Safavi, “Mitigating poisoning attacks on machine learning models: A data provenance based approach,” *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (AISec ’17)*, 2017.
 - [27] Xu, X., Gou, Z., Wu, W., Niu, Z.-Y., Wu, H., Wang, H., & Wang, S. (2022). Long time no see! Open-domain conversation with long-term persona memory. *Findings of the Association for Computational Linguistics: ACL 2022*, 2639–2650. <https://doi.org/10.18653/v1/2022.findings-acl.207>
 - [28] Bae, S., Kwak, D., Kang, S., Lee, M. Y., Kim, S., Jeong, Y., Kim, H., Lee, S.-W., Park, W., & Sung, N. (2022). Keep me updated! Memory management in long-term conversations. *Findings of the Association for Computational Linguistics: EMNLP 2022*, 3769–3787. <https://doi.org/10.18653/v1/2022.findings-emnlp.276>