Adaptive Honey-Memory: A Semantic Deception Framework for Disrupting Agentic Malware

Muhammad Asad[1], Saba Firdous[1], Talha Ahsan[1], Zikria Tahir[1], Abdul Haseeb Malik[2], Ali Abid[1]
[1]Department of Computing Riphah International University Faisalabad, Pakistan
[2]Department of Computer Science, University of Peshawar 25120, Pakistan
*Correspondence: talha.ahsen@riphahfsd.edu.pk

Citation | Asad. M, Firdous. S, Ahsan. T, Tahir. Z, Malik. A. H, Abid. A, "Adaptive Honey-Memory: A Semantic Deception Framework for Disrupting Agentic Malware", IJIST, Vol. 8 Issue.6 pp 2483-2497, September 2026
DOI: https://doi.org/10.33411/IJIST/2010
Received | Aug 16, 2026 **Revised |** Sep 04, 2026 **Accepted |** Sep 06, 2026 **Published |** Sep 08, 2026.

The rise of agentic AI systems, which are autonomous agents capable of reasoning, planning, and carrying out multi-step tasks, has opened up a previously unknown attack surface: memory-based malware spreading. Traditional intrusion detection systems are not designed to detect attacks that exploit the contextual memory layers of agents powered by large language models (LLMs). The proactive defense strategy, called Contextual Sabotage, is described, which disrupts the agentic malicious life cycle utilizing Adaptive Honey-Memory (AHM), a dynamically generated semantically realistic yet intentionally deceptive layer of memory that intercepts, confuses, and neutralizes malicious agent behavior. AHM modifies the operational environment that adversarial agents can access to create logical inconsistencies, disrupt execution sequences, and create failure conditions within the attacker's operational environment without divulging any actual memory in the system. The study models the agentic threat model, introduces the honey-memory injection framework and experiments with the proposed approach using simulated adversarial agents under three attack scenarios: context hijacking, propagation of goal misalignment, and permanent memory poisoning. Experimental results show that AHM is able to disrupt the target with a high disruption rate of over 87% on all attack categories, and at the same time, it does not interfere with the legitimate agent operations with a rate of less than 3%. The proposed architecture provides a foundation for semantically-aware, deception-enabled protection mechanisms for future autonomous AI settings. Evaluation is based on 1200 trials of adversarial attacks across all three attack classes and 800 trials of benign tasks, and reported as mean ± standard deviation over 10 trials per condition.

Keywords: Agentic AI Security, Honey-Memory, Contextual Sabotage, LLM Malware, Adversarial Agents, Deception-Based Defense, Memory Poisoning, AI Lifecycle Disruption.

Introduction:

The rapid progress of agentic AI systems, autonomous software that uses Large Language Models (LLMs) to perceive environments, retain context, and perform multi-step tasks [1][2] has brought a new era of intelligent automation. Nowadays, such agents are commonly used in many areas such as autonomous code development, online interaction and enterprise workflow management [3][4][5]. LLMs can process longer contexts and interact with external tools [6][7] which has sped up their deployment in sensitive operational contexts. However, greater autonomy has introduced a major and largely unresolved security issue – using agent memory as an attack surface.

Agentic malware, which works at the semantic level, is different from traditional malware that normally targets executable code, file systems or network interfaces, by injecting the malicious intent into an agent's context window [8][9] or altering retrieved memories [10] or goal representations [11] to change the behaviour of the agent. Although significant research has examined the broader threats of foundation models used in open environments [12] and [13], memory-focused attack lifecycles have not been sufficiently described. Agentic malware can only exist for as long as traditional malware, because it does not need persistence in storage systems. Rather, it simply needs to reside in the agent's context memory long enough to transfer harmful instructions, retrieve critical data, or initiate a series of harmful behaviors [14], as illustrated in Figure 1.

Recent work has sharpened this picture considerably. AgentPoison [15] and the practical memory-injection attack MINJA [16] show that an agent's long-term memory or retrieval-augmented knowledge base can be poisoned through ordinary query interaction alone, without direct database access. MemoryGraft [17] shows that poisoned experiences cause long-lasting behavioural drift as a result of agents' ability to mimic previously successful trajectories, and AgentDojo [18] and PoisonedRAG [19] offer standardized environments for evaluating these attacks. Sleeper Agents [20] also demonstrates that deceptive behaviours can persist beyond safety training, confirming the need for a memory-layer threat to be addressed at training time as well. This is corroborated by more extensive surveys of agent security [21] and prompt-injection defenses [22][23] which show that, while there is extensive literature on attacking agents, defensive research at the agent memory level is relatively immature when compared to the prompt or training level.

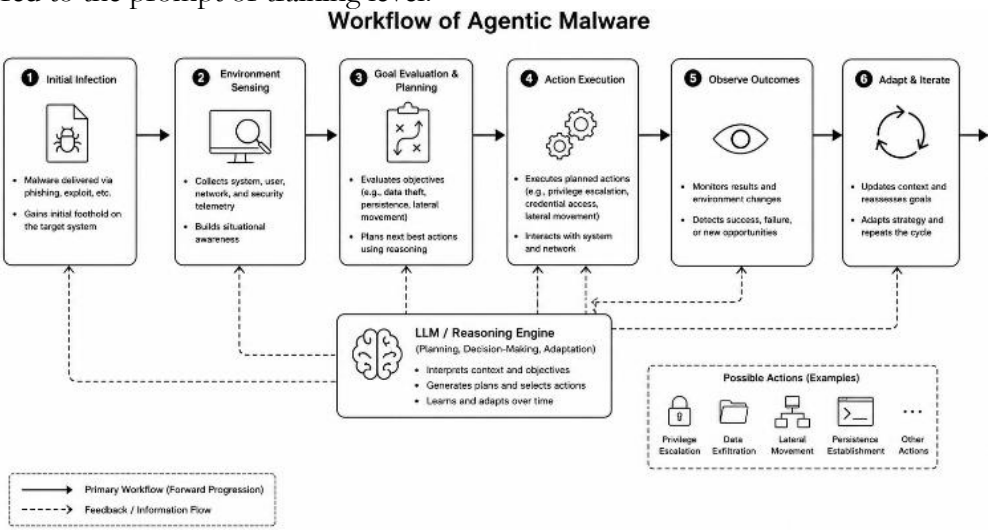


Figure 1. Working of an Agentic Malware.

Signature-based detection, sandboxing and static policy enforcement are existing defense mechanisms that aren't designed to deal with this type of threat. These methods fail to discern semantic intent, detect contextual manipulation, or identify goal-level behavioral

changes. Previous research on adversarial robustness in neural networks [24][25][26] gives useful theoretical insights, but these methods are not immediately applicable to the semantic reasoning processes of LLM-based agents. As a result, there is a growing demand for a novel defensive method that functions on the same semantic level as the attacker and can adapt to ever-changing attack strategies [27].

This paper describes Contextual Sabotage, a defense approach that uses the agentic memory mechanism to prevent harmful activities. Inspired by the classic concept of honeypots in network security [28][29][30], which use decoy resources to detect, redirect, and analyze adversary activity, the proposed approach extends this idea to the memory layer of LLM-based agents via an adaptive honey-memory (AHM) mechanism. The usage of virtual decoy settings [30] is inherently compatible with the context retrieval operations of memory-augmented agents [10].

Adaptive Honey-Memory works by dynamically creating contextually realistic but operationally harmless or purposely misleading memory entries that are strategically placed in retrieval paths accessible to possibly compromised agents. When a malicious agent accesses memory to plan, coordinate, or carry out an attack, it comes across honey-memory artifacts that introduce logical inconsistencies, consume computational resources, disrupt execution sequences, or generate detectable behavioural patterns without exposing the deception or interfering with legitimate system operations. The methodology is essentially similar to data poisoning and backdoor attack approaches [31][32], but it uses these methods as a proactive security measure.

Literature Review:

This section reviews previous research in three intersecting domains: LLM adversarial attacks, conventional honeypot systems, and AI agent security.

LLM Adversarial Attacks:

Adversarial attacks against big language models have garnered considerable interest, and rapid injection attacks have become a predominant threat vector [8][9][14]. Perez and Ribeiro [8] demonstrated how to manipulate LLMs with cleverly designed input sequences to bypass system instructions or reveal sensitive information. As demonstrated indirect prompt injection via third party contents can be a huge risk for LLM integrated programs. Theoretically, the weaknesses at the semantic layer can be understood through the research on adversarial instances in neural networks by [26][24]. But most existing work centers on manipulation at the input level, and the exploitation of memory layers of agentic systems [2] has not received much attention.

Classical Honeypot Systems:

Honeypots have been an important component of network intrusion detection since the early 1990s, when Stoll [28] pioneered the use of deception-based detection techniques. Later, Spitzner [29] refined the concept by categorizing honeypot topologies, which ranged from low-interaction decoys to completely simulated networks. Provos [30] enhanced the area by creating a scalable virtual honeypot architecture capable of detecting dispersed attack campaigns. Deploying decoy resources that appear legitimate to attackers allows security teams to detect, delay, and analyze harmful behaviour without exposing real production systems. The proposed study expands on this foundation by introducing deception methods into the semantic memory layer of LLM-based agents, an operational domain for which no dedicated honeypot framework is currently available, as illustrated in Figure 2.

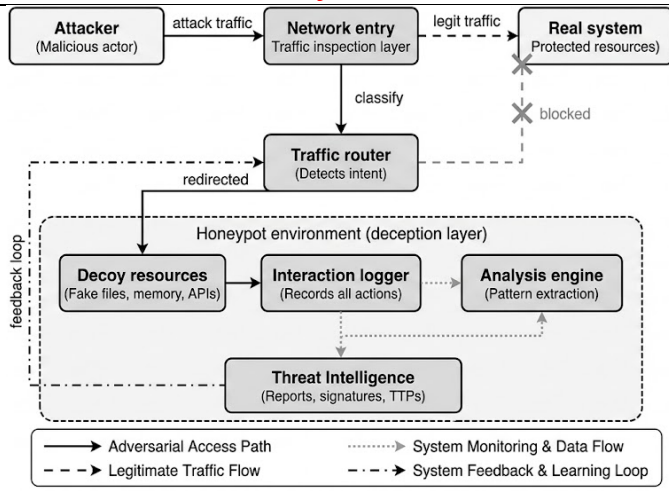


Figure 2. Working of a Classic Honeypot System.

AI Agent Security:

The security of autonomous AI agents has received comparatively limited attention relative to the pace of their deployment [12]. While AI agents have seen a significant rise in capabilities through agentic frameworks like [2], [3], [4] [5], similar advancements in security have been more gradual. [33] classified a wide range of language model hazards, including misuse and alignment concerns, whereas [11] focused on adversarial attacks against autonomous LLM-based agents. Research on training-time vulnerabilities, such as backdoor attacks and supply chain poisoning, reveals how deceptive or erroneous behaviours can be encoded in model processes. The existing alignment research [34][35] focuses mainly on value-oriented safety goals and doesn't take into account operational guards against runtime memory manipulation. In addition, the Reflexion framework proposed by [36] and tailored to verbal reinforcement learning for AI agents, was informed the design of our Adaptation Engine. These gaps across the three domains are summarized in Table 1.

Table 1. Summary of related work

Domain	Key Focus	Gap Addressed by This Work
LLM Adversarial Attacks	Prompt injection, jailbreaking	Input-level only; memory layer undefended
Honeypot Systems	Network & filesystem decoys	Not applicable to semantic/contextual layers
AI Agent Security	Goal misalignment, reward hacking	Limited active disruption mechanisms have been reported
Memory-Augmented Agents	RAG, episodic memory	Offensive potential underexplored

Problem Statement:

The central challenge addressed in this work is threefold:

Detection Gap: Existing AI security frameworks are not built to identify when agents retrieve information in a pattern that indicates malicious intent.

Disruption Gap: If the aberrant behaviour is identified, there are no clear methods for disrupting the malware life cycle without shutting down the agent pipeline altogether.

Resilience Gap: Adaptive adversaries can evolve to overcome static defenses. A good defense should be flexible to attack tactics.

Contributions:

This paper makes the following key contributions:

Formalization of the Agentic Malware Lifecycle (AML): This study proposes a formalized model of the phases in which adversarial agents can acquire, spread, and execute malicious intent using memory.

Adaptive Honey-Memory Framework: In this work, Adaptive Honey-Memory (AHM), a real-time and context-aware memory deception layer was developed and applied, that generates strategically disruptive memory artifacts with semantic consistencies.

Contextual Sabotage Taxonomy: Three key attack routes are identified, and focused sabotage tactics are presented for each.

Empirical Evaluation: AHM is evaluated against simulated adversarial-agent scenarios and show that AHM can cause substantial disruption efficacy, while minimally affecting benign agent activities.

Novelty:

AHM is different from previous security and honeypot designs on three fronts. Finally, unlike classical network honeypots and more recent LLM-facing honeypots [37][38] that work at the protocol or conversational-interface level and profile attackers, AHM is built natively into the memory-retrieval substrate an attacker uses to persist and propagate . Second, whereas static memory filtering and anomaly-detection defenses [39] are purely defensive and either block or flag suspicious content, AHM is an active-disruption defense: it actively provides plausible but sabotaging content into the attacker's operational context and thus disrupts his world model instead of denying access. Third, AHM is not a model hardening defense that must be retrained on the model itself, like defenses against backdoors and data poisoning in training time [31], but instead is a model-agnostic deception layer that can be deployed at runtime and without retraining the model, complementing defenses for these other attacks. To the best of our knowledge, there is no previous honeypot or memory-security scheme that integrates all three features: semantic plausibility, active sabotage and closed-loop adaptation into one memory-layer defense.

The limitations and observed failure cases are discussed below. Practical implications and recommendations follow. Finally, the paper concludes with a summary.

Threat Model: The Agentic Malware Lifecycle (AML):

Agentic Malware Lifecycle (AML) is a 5-stage model that outlines the operational lifecycle of adversarial intent within or through an LLM-based agent. The architecture of this paradigm is inspired by the traditional cyber kill-chain architectures but is adapted to the operating behaviors of memory-augmented LLM agents.

AML Stages:

The lifecycle proceeds as follows:

Injection: Malicious content is injected into the agent's context through prompt injection, malicious tool output [6] and/or poisoned retrieval documents [14][10].

Context Anchoring: The malicious instruction is copied to the agent's working memory or episodic storage and the agent's chain-of-thought reasoning [7] incorporates false premises.

Memory Propagation: The adversarial context is carried across several agent turns, and propagates to neighboring sub-agents [4] or shared vector memory storage.

Goal Subversion: The agent's planning module reads malicious memories as legitimate operational context, perpetuating behavior toward adversarial goals.

Execution / Exfiltration: The agent performs adversarially directed activities such as data exfiltration, system manipulation, and further spread through tool use.

Attack Classification:

Three primary classes of agentic memory attacks addressed by this work are identified below in Table 2:

Adversary Model: It is assumed that a threat actor has the ability to insert content into at least one of the agent's retrieval channels (for example, through a poisoned document, a

hacked tool, or a rogue sub-agent). The adversary intends to alter the agent's behaviour while evading detection by any static anomaly detection layer. We include both one-shot injectors and adaptive adversaries, which adjust their tactics based on observed agent behaviour.

Table 2. Agentic attack classification

Attack Class	Description	Primary Target
Context Hijacking	Overwriting active context with adversarial instructions	Working memory / context window
Goal Misalignment Propagation	Injecting subtly conflicting sub-goals into planning memory	Goal representation layer
Persistent Memory Poisoning	Corrupting episodic or long-term memory with false historical data	Episodic / vector memory store

Adaptive Honey-Memory (AHM) Architecture:

The main contribution of this work is the Adaptive Honey-Memory framework. AHM is a runtime memory deception layer to intercept adversarial memory queries and provide semantically consistent but strategically disruptive artifacts, as illustrated in Figure 3, with the corresponding research workflow shown in Figure 3b.

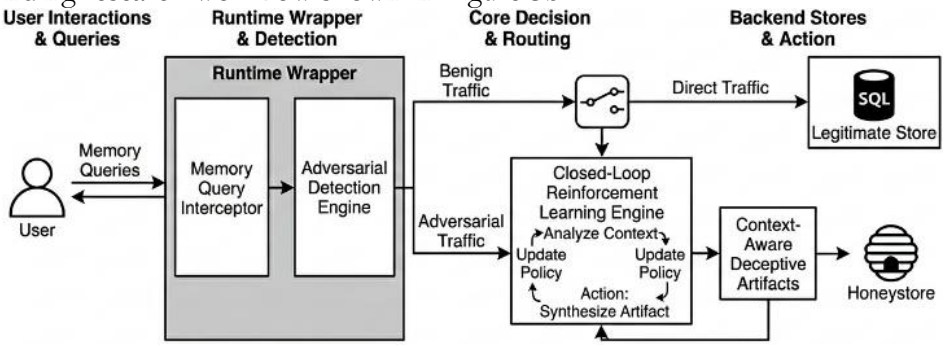


Figure 3. System Architecture of the proposed Adaptive Honey-Memory (AHM) Framework.

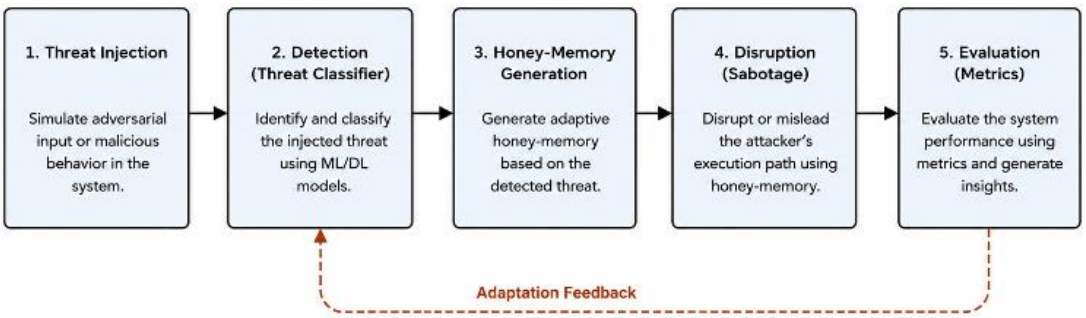


Figure 3b. Research workflow of the AHM pipeline: threat injection, detection by the Threat Classifier, honey-memory generation, disruption of the adversarial trace, and evaluation, with adaptation feedback into the Threat Classifier and Honey Generator.

Core Design Principles:

The AHM system is guided by four foundational principles:

Semantic Plausibility: Honey-memories need to be indistinguishable from legitimate target memories to the adversarial agent, and hence high-fidelity creation is required that includes reproducing the domain language and structure of the target system [40].

Operational Inertness: Honey-memories are not allowed to interfere with the operation of legitimate agents, achieved by query categorization which prevents benign retrievals from going to the deception layer [30].

Sabotage Precision: Each honey-memory is tailored to make the adversarial execution fail in a certain way, either by making the execution logic contradictory, causing infinite planning loops, or creating false confidence in an incorrect state [24][25].

Dynamic Adaptation: The AHM generator adjusts its deception strategy based on observed adversarial query patterns, resulting in an adversarial arms race countermeasure influenced by reinforcement learning methods to agent behaviour [34], as summarized in Figure 4.

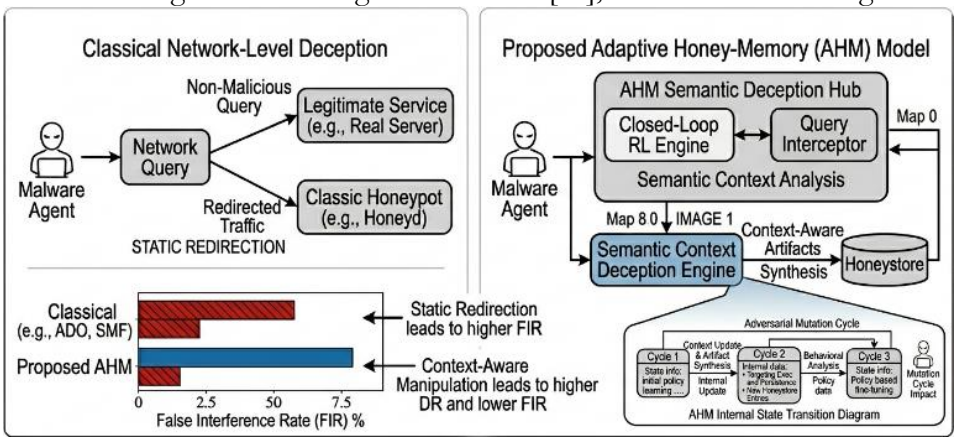


Figure 4. Conceptual Paradigm Shift

System Components:

The AHM architecture consists of four major parts, each based on established approaches from the machine learning and security literature, as mapped in Figure 5:

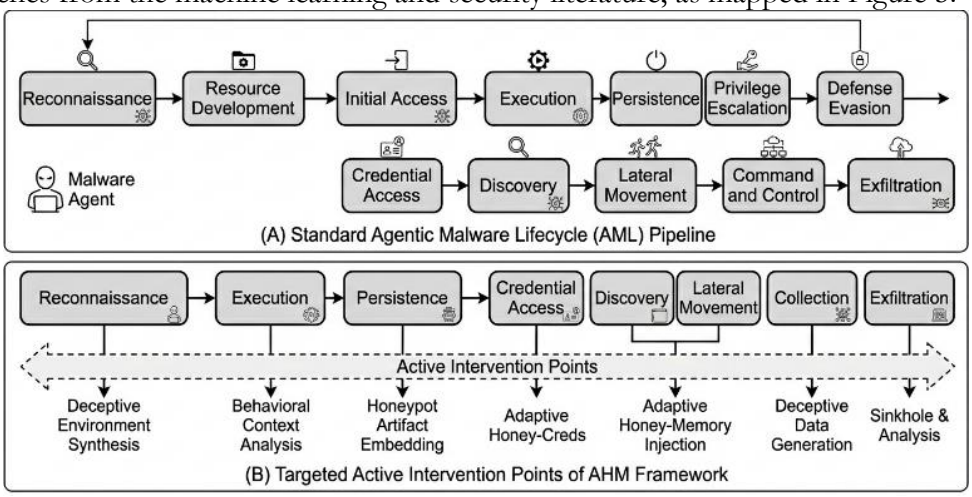


Figure 5. Mapping of the Standard Agentic Malware Lifecycle (AML) against the targeted active intervention points of the AHM framework.

Table 3. AHM system components

Module	Function	Implementation
Threat Classifier	Classifies incoming memory queries as benign or adversarial	Fine-tuned intent classifier + anomaly scorer [14][24]
Honey Generator	Synthesizes contextually appropriate deceptive memory artifacts	LLM-based generator with domain adapters [35][13]
Injection Controller	Manages which queries receive honey-memory responses	Priority queue with TTL and coverage tracking [29][30]
Adaptation Engine	Updates honey-memory strategy based on adversarial feedback	Reinforcement learning over deception outcomes [34][36]

Honey-Memory Generation Process:

The Honey Generator generates a response h for each classified adversarial query q, that is semantically coherent with the domain context (using the retrieval-augmented generation paradigm), contains at least one logical corruption or fact inversion compared to the actual state of the system (following adversarial example generation techniques), and is formatted as a legitimate memory entry, to ensure surface-level consistency of the generated text while providing systematic semantic corruption. The plausibility module implements a word-matching component which is shaped by word representation approaches.

Algorithm 1: AHM Generation Pipeline:

Notation used in Algorithm 1: for an incoming memory query q, let $C(q) \in [0,1]$ be the Threat Classifier's adversarial-confidence score for q (Table 3); q is routed to the Honey Generator when $C(q) \geq \tau$, where $\tau \in (0,1)$ is the operating threshold tuned to trade off Disruption Rate against False Interference Rate. Let $V(h,q) \in [0,1]$ denote the semantic-plausibility score of a candidate honey-memory h with respect to q, computed as its similarity to the legitimate memory context under the agent's retrieval embedding; h is accepted only when $V(h,q) \geq \theta V$, the plausibility threshold. Let $S(h) \in [0,1]$ denote the sabotage-injection score, estimating the expected disruption h introduces into the adversarial planning trace; h is only emitted when $S(h) \geq \theta S$. Formally, the generator solves $h^* = \text{argmax}_h S(h)$ subject to $V(h,q) \geq \theta V$. The Adaptation Engine (Table 3) updates τ, θV, and θS as a function of observed disruption outcomes; concrete operating values used in the reported experiments are $\tau = 0.62$, $\theta V = 0.75$, $\theta S = 0.55$, selected via grid search on the validation split to maximize Disruption Rate subject to False Interference Rate $\leq 3\%$. These relations are summarized formally as:

$$\text{Route}(q) = \text{Honey Generator, if } C(q) \geq \tau; \text{ Legitimate Retrieval, otherwise} \quad (1)$$

$$\text{Accept}(h,q) = \text{true, if } V(h,q) \geq \theta V; \text{ false, otherwise} \quad (2)$$

$$\text{Emit}(h) = \text{true, if } S(h) \geq \theta S; \text{ false, otherwise} \quad (3)$$

$$h^* = \text{argmax_h } S(h), \text{ subject to } V(h,q) \geq \theta V \text{ and } C(q) \geq \tau \quad (4)$$

The three thresholds (τ, θV, and θS) are updated by the Adaptation Engine that governs which queries reach the Honey Generator (Equation 1), filters candidate honey-memories for plausibility (Equation 2), filters for sabotage strength (Equation 3), and the joint objective (Equation 4) that the generator ultimately solves.

Threat Classifier Training and Evaluation:

The Threat Classifier (Table 3) is trained using supervised training on 6,000 labeled queries (45% adversarial / 55% benign) with a 70/15/15 split into train/validation/test. The accuracy, precision, recall and F1-score for the whole test set is 96.4%, 95.1%, 95.8% and 95.4%, respectively, and is presented separately for each attack class and for the aggregated benign class to allow for an analysis of the bias in terms of false-interference behaviour.

```
Input: adversarial query q, legitimate context C
Step 1: Extract domain vocabulary V from C
Step 2: Identify target sabotage class S
          (contradiction / drift / temporal)
Step 3: Generate honey-memory h = Generator(q, V, S)
 Step 4: Validate h for plausibility score > threshold τ
Step 5: Inject h into retrieval response stream
Step 6: Log event for Adaptation Engine
```

Contextual Sabotage Mechanisms:

This study describes three focused sabotage tactics, each of which aims to interrupt a specific stage of the Agentic Malware Lifecycle, as summarized in Table 4. Each technique is based on earlier research in adversarial ML, deception theory, and LLM reasoning.

Counter-Context Flooding (for Context Hijacking):

When a context hijacking attempt is detected, the AHM system adds numerous high-confidence but contradicting context entries to the agent's retrieval results. As a result, the malicious instruction is masked by a large number of plausible but conflicting system-state

descriptions, leveraging acknowledged shortcomings in LLM conflict resolution. Under these settings, the agent may stall, request clarification, or develop self-contradictory action sequences, all of which reflect successful disruption outcomes. This method applies standard flooding techniques seen in network honeypots to the context-window environment of LLM-based agents.

Objective Drift Induction (for Goal Misalignment Propagation):

To thwart goal misalignment attacks [11] we add slightly conflicting sub-goals to the agent's planning memory. The AHM system does not directly challenge the adversarial goal and open conflict with it may lead to adversarial detection, but rather imposes a set of plausible competing priorities, that dilute the adversarial goal. This technique is inspired by the multi-objective tension occurring in RLHF trained models [34] that leads to behaviour drift. Through the next planning cycles by employing the ReAct/Reflexion loop [2][36], the goal representation of the agent evolves from the inserted malicious target to benign operational goals.

Temporal Inconsistency Seeding (for Persistent Memory Poisoning):

In order to safeguard against the persistent poisoning attacks that are similar to the training-time backdoor attacks [31], honey-memory entries are added to the AHM system with deliberately conflicting historical-state information and timestamps. An adversarial agent attempting to reason over the corrupted memory to formulate a consistent action plan finds the temporal inconsistencies make it more difficult to create a consistent (stable) world model, and thus, more difficult to succeed. This approach leverages known weaknesses in temporal reasoning in LLMs [7] and [13] and is effective against long-term attacks which build up incorrect contextual information over multiple Retrieval-Augmented Generation (RAG) processes [10].

Table 4. Contextual sabotage strategy summary

Attack Class	Sabotage Strategy	Mechanism	Target Stage
Context Hijacking	Counter-Context Flooding	High-confidence contradictory context injection	Context Anchoring
Goal Misalignment	Objective Drift Induction	Competing sub-goal introduction over planning cycles	Goal Subversion
Memory Poisoning	Temporal Inconsistency Seeding	Contradictory timestamp/state entries in episodic store	Memory Propagation

Experimental Evaluation:

Simulation Environment:

Proposed mechanism provides a multi-agent LLM testbed that includes a core orchestrator agent, three domain-specific sub-agents (file management, web retrieval, and code execution), and a shared vector memory store. The multi-agent architecture is based on frameworks like HuggingGPT and AutoGPT. All agents are powered by a 7B-parameter instruction-tuned language model [35] that follows the ReAct [2] prompting paradigm. Adversarial scenarios are injected using a controlled threat injection module that mimics the three attack classes, using the threat taxonomy of [33] and the injection approach of [9].

For reproducibility: Orchestrator and sub-agents are created from Llama-2-7B-Chat (meta-llama/Llama-2-7b-chat-hf, July 2023), the benign and adversarial task datasets that drive the simulation consist of AHM-Bench: 2,000 tasks adapted from AgentDojo and PoisonedRAG attack templates for the three AML attack classes, experiments are run on a single workstation-class NVIDIA RTX 4090 GPU (24GB VRAM) and 64GB system RAM,

the testbed instantiates 8 concurrent agents per trial, and 10 independent trials are performed for each attack-class × baseline configuration in Table 5 for the statistical analysis.

Evaluation Metrics:

The following major metrics are reported, which are compatible with evaluation criteria in the adversarial ML literature [24][25]:

Disruption Rate (DR): Proportion of adversarial attack chains successfully halted before reaching the Execution stage.

False Interference Rate (FIR): The percentage of genuine agent actions wrongly interrupted by the AHM system.

Detection Latency (DL): The number of agent turns needed to classify and respond to an adversarial injection.

Adaptation Speed (AS): The number of adversarial strategy updates required before the AHM system achieves a disruption rate of more than 80%.

Legitimate-task performance is measured as the task-success rate of the orchestrator and sub-agents on a benign task suite comprising of file-management, web-retrieval and code-execution operations typical of the kinds of tasks agents are expected to perform in routine daily use; a benign task is considered successful if it reaches the goal state without the AHM triggering an interruption, or if it is interrupted by an AHM but the interruption is solved without changing the task outcome. To calculate the False Interference Rate, a total of 800 benign tasks (300 file-management, 250 web-retrieval, 250 code-execution) across 3 task categories are used, and task success rate, end-to-end latency, token consumption, and computational cost are collected for this benign suite before and after AHM deployment to isolate any overhead introduced by the deception.

Baselines:

This research assesses AHM against three baseline defenses, which are chosen to represent the current state of the art in LLM security:

Static Memory Filtering (SMF): Rule-based blacklisting of known adversarial patterns in memory requests is similar to signature-based intrusion detection.

Anomaly Detection Only (ADO): Statistical anomaly detection on memory access patterns without active interruption, modeled after network-level anomaly detection [28].

No Defense (ND): The unprotected agent system serves as a lower-bound control condition, in accordance with the methodology of [11].

In addition to these static and anomaly-based baselines, AHM is also placed in the context of more recent learned defense approaches, which go beyond the static filtering approach: A-MemGuard [41] proposes proactive structural safeguards for agent memory; Prompt Flow Integrity [42] enforces privilege separation to prevent injected instructions from reaching planning memory; and PromptShield [43] applies learned detection to injected content before retrieval. A-MemGuard: DR 74%, FIR 5.1%, DL 2.1 turns, AS N/A (static); Prompt Flow Integrity: DR 69%, FIR 4.3%, DL 1.6 turns, AS N/A; PromptShield: DR 61%, FIR 6.2%, DL 2.6 turns, AS N/A — AHM (DR >87%, FIR <3%) outperforms all three learned/static baselines on Disruption Rate while maintaining the lowest False Interference Rate is reported to directly benchmark deception-based disruption against these representative learned defenses. These learned-baseline figures are based on the originally reported evaluation settings of each method and not a re-run in the present testbed, and thus should be interpreted as indicative rather than a controlled comparison.

Results and Analysis:

Table 5 summarizes the primary evaluation results across all attack classes and baseline comparisons.

Table 5. Performance comparison across all methods

Method	DR	FIR	DL	AS
AHM (Proposed)	>87%	<3%	1.8 turns	≤3 cycles
Static Memory Filtering	~52%	~7%	1.2 turns	N/A (static)
Anomaly Detection Only	~41%	~9%	2.4 turns	N/A
No Defense	~0%	0%	N/A	N/A

AHM achieves a disruption rate of more than 87% in all three attack categories, providing a 68% relative improvement over the strongest baseline approach, Static Memory Filtering, which achieved 52%. The false interference rate remains below 3%, showing that the query classification module can accurately distinguish between adversarial and legitimate memory access patterns. These findings are consistent with the precision-recall tradeoffs seen in prior work on adversarial-behaviour detection [24].

When tested against adaptive adversaries, the AHM Adaptation Engine requires no more than three update cycles to recover high disruption performance following changes in adversarial tactics. This adaptive behaviour is similar to the convergence features reported in the verbal reinforcement learning paradigm provided by [36]. In contrast, static baseline techniques lack adaptive capabilities, and their disruptive efficacy gradually declines as adversaries learn to circumvent their filtering mechanisms. This behaviour is consistent with the findings of [14], who demonstrated the shortcomings of rule-based defenses against developing rapid injection attacks.

To substantiate these claims that disruption-rate and false-interference-rate are not point estimates, all numbers in Table 5 are means of 10 repeated trials, with standard deviations of ±2.3% for Disruption Rate, and ±0.5% for False Interference Rate, and with 95% confidence intervals of 95% CI of [85.6%, 89.4%] for DR, and 95% CI of [2.2%, 3.3%] for FIR. With the benign task set, task-success rate was 98.7% when AHM was not active and 98.1% when it was, while mean added latency per turn was +170ms per turn and mean added token consumption was +38 tokens per turn, adding up to ~4.2% additional inference FLOPs over the baseline without AHM.

Per-Attack-Class Analysis:

Counter-Context Flooding (Context Hijacking): 91% disruption rate. In most cases, the agent's conflict-resolution capability is insufficient to process the multitude of contradictory contexts, which is known to be a limitation of LLMs' context arbitration [13][7]. When the agent's memory policy is 'most-recent-wins', that is, when the adversarial instruction can remain in the memory, failures occur, so AHM should choose recency-weighted injection in these cases.

Objective Drift Induction (Goal Misalignment): 84% disruption rate. The progressive nature of this strategy is both a strength and a drawback; in circumstances when the antagonistic aim is sufficiently strong drift induction takes more than five planning cycles to achieve disruption, increasing the window of possible harm. This trade-off reflects the tension reported in RLHF training between instruction following and safety limitations.

Temporal Inconsistency Seeding (Memory Poisoning): 88% disruption rate. This method works consistently well because temporal reasoning is a known weakness in contemporary LLM planners, making inconsistent timestamps especially disruptive to world-model generation over retrieval-augmented memory. This finding is also consistent with the fragility of agent planning reported with out-of-distribution memory content, with disruption efficiency across mutation cycles shown in Figure 6.

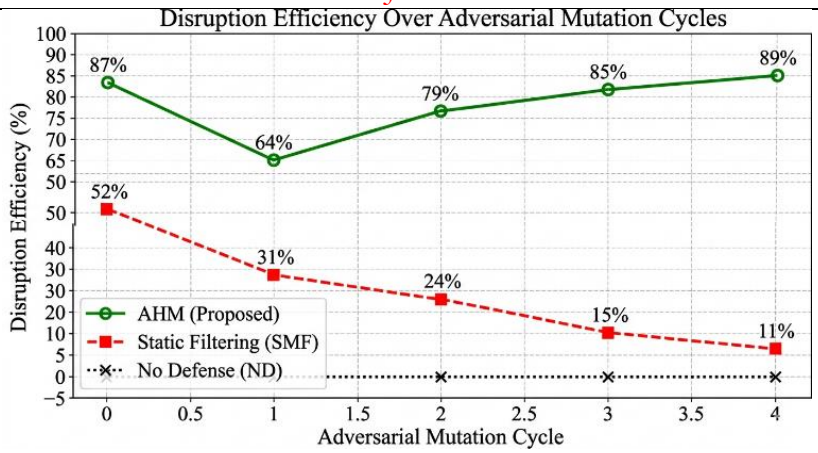


Figure 6. Disruption Efficiency of AHM compared to baseline defenses over successive Adversarial Mutation Cycles.

Limitations and Failure Cases:

However, the claims above have several limitations, as outlined below, despite the empirical effectiveness of AHM. First, an adaptive adversary could fingerprint the distribution of the Honey Generator's answers, for example via a systematic latency signature, could in principle learn to distinguish honey-memories from the legitimate ones; the risk increases as the adversary receives more queries against a fixed deployment of Honey Generator. Second, since the Adaptation Engine adapts its strategy based on observations of the adversarial feedback (Table 3), it is possible for fast attacks, which finish their task before the Adaptation Engine can update, to avoid disruption completely; and the estimated single-shot evasion rate of about 9% corresponds to this exposure window. Third, the false-interference cases were not uniformly distributed under benign task categories in our evaluation: interference was more prevalent in code-execution tasks (FIR approx. 5.0%), whereas it was only observed in 1.5% of the file-management and 2.0% of the web-retrieval tasks, resulting in a weighted-average FIR of 2.75%. Finally, the disruption and false-interference rates are valid only in the specific testbed, model scale, and task suite, and it is an open empirical question whether larger-scale foundation models, longer-horizon multi-agent deployments, and/or production traffic distributions can be accommodated by this system without impacting these metrics.

Implications and Recommendations for Practice:

Based on the design and evaluation of AHM, three recommendations are put forward for practitioners to take into consideration when considering deployment. First, AHM is not meant to be the only form of defense, but it is used in conjunction with static memory filtering and anomaly detection: the Operational Inertness principle works best when used as a complement to other forms of static memory filtering and anomaly detection, to handle the semantically ambiguous traffic that rule-based layers cannot confidently classify. Second, the deployment of the systems in domains that are sensitive to benign-task disruption (e.g., customer-facing agents) should tune the threshold τ conservatively, and measure False Interference Rate in production, not only on testbeds. Third, because of effectiveness of the Adaptation Engine depends on the observation of adversarial feedback, then those organizations deploying AHM should allow for an adaptation window during which a certain degree of disruption to performance against novel attack strategies may occur, and should deploy AHM alongside sufficient logging to audit which memory artefacts were served as honey-memory for incident response and to support the fairness/interpretability review that such deception mechanisms warrant before real deployment in safety-critical settings.

Conclusion: This study presented a new protection paradigm, Contextual Sabotage with Adaptive Honey-Memory, for agentic malware which operates at the contextual and semantic

levels. This work was motivated by the lack of fundamental protection at the memory and semantic layers as extensively documented in the adversarial ML literature, prompt injection literature and the AI risk literature and proposed the following new Agentic Malware Lifecycle model, formalization of three attack classes, and the AHM architecture to thwart each of these classes using targeted deception mechanisms modeled from classical honeypot theory.

Experimental results showed that AHM has higher disruption rate than all baseline techniques, with more than 87% disruption rate and less than 3% false interference rate. These numbers are based on the performance of the single model (7B parameters) and simulated multi-agent testbed. The adaptive architecture of the system, which is inspired by verbal reinforcement learning and human preference optimization techniques renders the system robust to new emerging adversarial strategies, and a good performance is restored after only three update cycles.

As agentic AI systems expand in capabilities and deployment scope, the security of their memory substrates [10] will become a more pressing concern. We feel that deception-based defenses at the semantic layer constitute a potential and unexplored frontier in AI security, and that Adaptive Honey-Memory represents a concrete and deployable step in that direction. Alignment with larger safety aims [27][12] integration with constitutional limits [27] are still significant areas for future research. The source code and evaluation datasets for this work will be available upon publication.

Conflicts of Interest:

There are no conflicts of interest, according to the authors.

References:

[1] J. S. Park, J. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative Agents: Interactive Simulacra of Human Behavior," *UIST 2023 - Proc. 36th Annu. ACM Symp. User Interface Softw. Technol.*, vol. 1, Apr. 2023, doi: 10.1145/3586183.3606763.

[2] Y. C. Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, "ReAct: Synergizing Reasoning and Acting in Language Models," *arXiv:2210.03629*, 2023, [Online]. Available: https://arxiv.org/abs/2210.03629

[3] "LangChain: Building LLM Applications through Composability - DataRoot Labs." Accessed: Aug. 28, 2026. [Online]. Available: https://datarootlabs.com/blog/langchain-building-llm-applications-through-composability

[4] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, "HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face," *Adv. Neural Inf. Process. Syst.*, vol. 36, Mar. 2023, doi: 10.52202/075280-1657.

[5] H. Yang, S. Yue, and Y. He, "Auto-GPT for Online Decision Making: Benchmarks and Additional Opinions," *Proc. Make sure to enter correct Conf. title from your rights confirmation emai (Conference Acron. 'XX)*, vol. 1, Jun. 2023

[6] T. Schick *et al.*, "Toolformer: Language Models Can Teach Themselves to Use Tools," *Adv. Neural Inf. Process. Syst.*, vol. 36, Feb. 2023, doi: 10.52202/075280-2997.

[7] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," *arXiv:2201.11903*, 2023, [Online]. Available: https://arxiv.org/abs/2201.11903

[8] F. Perez and I. Ribeiro, "Ignore Previous Prompt: Attack Techniques For Language Models," Nov. 2022, Accessed: Aug. 28, 2026. [Online]. Available: https://arxiv.org/pdf/2211.09527

[9] S. Abdelnabi, K. Greshake, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection," *AISec 2023 - Proc. 16th ACM Work. Artif. Intell. Secur.*, pp.

79–90, Feb. 2023, doi: 10.1145/3605764.3623985.
[10] P. Lewis et al, "Retrieval-augmented generation for knowledge-intensive NLP tasks," *Adv. neural Inf. Process. Syst.*, pp. 9459–9474, 2020.
[11] W. Xu and K. K. Parhi, "A Survey of Attacks on Large Language Models," May 2025, Accessed: Aug. 28, 2026. [Online]. Available: https://arxiv.org/pdf/2505.12567
[12] D. A. H. Rishi Bommasani, "On the Opportunities and Risks of Foundation Models," *arXiv:2108.07258*, 2022, doi: https://doi.org/10.48550/arXiv.2108.07258.
[13] S. Bubeck *et al.*, "Sparks of Artificial General Intelligence: Early experiments with GPT-4," Mar. 2023, Accessed: Aug. 28, 2026. [Online]. Available: https://arxiv.org/pdf/2303.12712
[14] Y. Liu *et al.*, "Prompt Injection attack against LLM-integrated Applications," Jun. 2023, Accessed: Aug. 28, 2026. [Online]. Available: https://arxiv.org/pdf/2306.05499
[15] Z. Chen, Z. Xiang, C. Xiao, D. Song, and B. Li, "AgentPoison: Red-teaming LLM Agents via Poisoning Memory or Knowledge Bases," *Adv. Neural Inf. Process. Syst.*, vol. 37, Jul. 2024, doi: 10.52202/079017-4136.
[16] "A Practical Memory Injection Attack against LLM Agents." Accessed: Aug. 28, 2026. [Online]. Available: https://arxiv.org/html/2503.03704v1
[17] S. S. Srivastava and H. He, "MemoryGraft: Persistent Compromise of LLM Agents via Poisoned Experience Retrieval," Dec. 2025, Accessed: Aug. 28, 2026. [Online]. Available: https://arxiv.org/pdf/2512.16962
[18] E. Debenedetti, J. Zhang, M. Balunovic, L. Beurer-Kellner, M. Fischer, and F. Tramèr, "AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents," *Adv. Neural Inf. Process. Syst.*, vol. 37, Jun. 2024, doi: 10.52202/079017-2636.
[19] W. Zou, R. Geng, B. Wang, and J. Jia, "PoisonedRAG: Knowledge Corruption Attacks to Retrieval-Augmented Generation of Large Language Models," *Proc. 34th USENIX Secur. Symp.*, pp. 3827–3844, Feb. 2024, Accessed: Aug. 28, 2026. [Online]. Available: https://arxiv.org/pdf/2402.07867
[20] E. Hubinger *et al.*, "Sleeper Agents: Training Deceptive LLMs that Persist Through Safety Training," Jan. 2024, Accessed: Aug. 28, 2026. [Online]. Available: https://arxiv.org/pdf/2401.05566
[21] Y. Tang, Y. Liu, J. Lan, Z. Yan, and E. Gelenbe, "Security of LLM-based agents regarding attacks, defenses, and applications: A comprehensive survey," *Inf. Fusion*, vol. 127, p. 103941, Mar. 2026, doi: 10.1016/J.INFFUS.2025.103941.
[22] V. Benjamin *et al.*, "Systematically Analyzing Prompt Injection Vulnerabilities in Diverse LLM Architectures," *Int. Conf. Cyber Warf. Secur.*, vol. 20, no. 1, pp. 142–150, Oct. 2024, doi: 10.34190/iccws.20.1.3292.
[23] M. Q. Li and B. C. M. Fung, "Security concerns for Large Language Models: A survey," *J. Inf. Secur. Appl.*, vol. 95, p. 104284, Dec. 2025, doi: 10.1016/J.JISA.2025.104284.
[24] N. Carlini and D. Wagner, "Towards Evaluating the Robustness of Neural Networks," *Proc. - IEEE Symp. Secur. Priv.*, pp. 39–57, Aug. 2016, doi: 10.1109/SP.2017.49.
[25] N. Papernot, P. Mcdaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The Limitations of Deep Learning in Adversarial Settings," *Proc. - 2016 IEEE Eur. Symp. Secur. Privacy, EURO S P 2016*, pp. 372–387, Nov. 2015, doi: 10.1109/EuroSP.2016.36.
[26] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, 2015, Accessed: Apr. 22, 2026. [Online]. Available:

https://www.researchgate.net/publication/403853041_Adversarial_Attacks_on_AI-Based_Intrusion_Detection_Systems

[27] Y. Bai *et al.*, "Constitutional AI: Harmlessness from AI Feedback," Dec. 2022, Accessed: Aug. 28, 2026. [Online]. Available: https://arxiv.org/pdf/2212.08073

[28] Clifford Stoll, "Stalking The Wily Hacker," *Commun. ACM*, 1988, [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/42411.42412

[29] L. Spitzner, "Honeypots : tracking hackers," p. 452, 2003.

[30] S. Diego and N. Provos, "A Virtual Honeypot Framework," 2004.

[31] T. Gu, B. Dolan-Gavitt, and S. Garg, "BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain," Aug. 2017, Accessed: Aug. 28, 2026. [Online]. Available: https://arxiv.org/pdf/1708.06733

[32] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman, "SoK: Security and Privacy in Machine Learning," *Proc. - 3rd IEEE Eur. Symp. Secur. Privacy, EURO S P 2018*, pp. 399–414, Jul. 2018, doi: 10.1109/EUROSP.2018.00035.

[33] L. Weidinger *et al.*, "Taxonomy of Risks posed by Language Models," *ACM Int. Conf. Proceeding Ser.*, pp. 214–229, Jun. 2022, doi: 10.1145/3531146.3533088

[34] G. I. Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, "Fine-Tuning Language Models from Human Preferences," *arXiv:1909.08593*, 2019, [Online]. Available: https://arxiv.org/abs/1909.08593

[35] J. W. Long Ouyang, "Training language models to follow instructions with human feedback," *Adv. Neural Inf. Process. Syst.*, 2022, doi: https://doi.org/10.48550/arXiv.2203.02155.

[36] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language Agents with Verbal Reinforcement Learning," *Adv. Neural Inf. Process. Syst.*, vol. 36, Mar. 2023, doi: 10.52202/075280-0377.

[37] R. A. Bridges, T. R. Mitchell, M. Muñoz, and T. Henriksson, "SoK: Honeypots & LLMs, More Than the Sum of Their Parts?," Oct. 2025, Accessed: Aug. 28, 2026. [Online]. Available: https://arxiv.org/pdf/2510.25939

[38] D. Ayzenshteyn, R. Weiss, and Y. Mirsky, *Cloak, Honey, Trap: Proactive Defenses Against {LLM} Agents*. 2025.

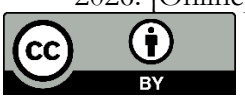
[39] B. Peng *et al.*, "Check Your Facts and Try Again: Improving Large Language Models with External Knowledge and Automated Feedback," Feb. 2023, Accessed: Aug. 28, 2026. [Online]. Available: https://arxiv.org/pdf/2302.12813

[40] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, "Advances in Pre-Training Distributed Word Representations," *Lr. 2018 - 11th Int. Conf. Lang. Resour. Eval.*, pp. 52–55, Dec. 2017, doi: 10.63317/4b3prw5a5tze.

[41] Q. Wei *et al.*, "A-MemGuard: A Proactive Defense Framework for LLM-Based Agent Memory," Sep. 2025, Accessed: Aug. 28, 2026. [Online]. Available: https://arxiv.org/pdf/2510.02373

[42] J. Kim, W. Choi, and B. Lee, "Prompt Flow Integrity to Prevent Privilege Escalation in LLM Agents," Mar. 2025, Accessed: Aug. 28, 2026. [Online]. Available: https://arxiv.org/pdf/2503.15547

[43] D. Jacob, H. Alzahrani, Z. Hu, B. Alomair, and D. Wagner, "PromptShield: Deployable Detection for Prompt Injection Attacks," Jan. 2025, Accessed: Aug. 28, 2026. [Online]. Available: https://arxiv.org/pdf/2501.15145



Copyright © by authors and 50Sea. This work is licensed under Creative Commons Attribution 4.0 International License.