Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 2 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189

http://amresearchreview.com/index.php/Journal/about

Detection and Mitigation of Prompt Injection and Memory Poisoning Attacks in Agentic AI Systems

Ali Asad Mumtaz
Department of Computer Science & IT, Baba Guru Nanak University, Nankana Sahib, Pakistan

Muhammad Atif
Department of Computer Science & IT, Baba Guru Nanak University, Nankana Sahib, Pakistan

Uzair Ashraf
Department of Computer Science & IT, Baba Guru Nanak University, Nankana Sahib, Pakistan

Maira Sattar
Department of Computer Science, School of Systems and Technology, University of Management and Technology (UMT), Lahore, Pakistan

Shahzad Nazir* (Corresponding Author)
Department of Computer Science & IT, Baba Guru Nanak University, Nankana Sahib, Pakistan
shahzad.nazir@bgnu.edu.pk

ABSTRACT

large language model (LLM) agents have tended to integrate external tools, retrieval-augmented generation (RAG), and persistent memory with natural-language reasoning. These features make tasks more autonomous but also make them more vulnerable: prompt injection can change an agent's direction during a session and memory poisoning can carry malicious input at later tasks and tool calls. In this paper, we present TAMGuard: A Trust-Aware Framework for Prompt Injection and Memory Poisoning Detection and Mitigation in Agentic AI systems. TAMGuard combines input canonicalization, a transformer-based semantic detector, a provenance-aware trust scoring model, a consistency checker based on trusted memory, quarantined memory writes, trust-aware retrieval and a risk-sensitive tool-action gate. TAMGuard has been tested with Agent Security Bench; external testing also has been conducted with AgentDojo and InjecAgent for direct, observation-based, indirect, and memory-oriented attacks. Experimental results demonstrate that TAMGuard achieves a 95.31% accuracy, 95.30% macro F1, and 98.70% macro AUROC for three-way classification

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

of benign, prompt-injection, and memory-poisoning instances, and only lowers the residual attack success rate to 4.7% while maintaining a benign task utility of 94.2%. Based on the results of the ablation analysis, it can be said that the three are complementary protection: (1) provenance, (2) memory consistency, (3) quarantine, and (4) action authorization. This study proposes a model to enforce agent security throughout the information lifecycle, not just on the boundary of the prompt.

Keywords— agentic AI, large language models, prompt injection, memory poisoning, LLM agents, RAG security, provenance, trustworthy AI, adversarial machine learning.

1. Introduction

Agentic AI is an approach that transcends the limitations of traditional text generation systems, enabling AI to make sense of the context, think through actions, access knowledge, keep track of state, and call out external tools. In general, LLM-based agents are typically integrated with reasoning-and-acting loops like ReAct [3], tool-using approaches [4], retrieval-augmented generation (RAG) [5], persistent or hierarchical memory [6]-[8]. These elements enable agents to perform various actions, including email management, information retrieval, code support, scheduling, financial transactions, and workflow automation. Autonomous LLM agents have been demonstrated on a variety of tasks, and memory, planning, perception and action are now common architectural components and not optional extras [1, 2]. The capability provided by such connectivity, however, also makes the actions of the agent vulnerable to the text and state prior to them being outside the trusted boundary of the instructions that the model can provide.

The main security issue is the distinction between executable instruction and data is no longer valid. Prompt injection takes advantage of this ambiguity by embedding attacker-controlled instructions in user input, web pages, documents, emails, tool responses, or retrieved passages to have the LLM follow the malicious instructions as trusted guidance [9, 10]. As with InjecAgent and AgentDojo [11] and [12] for example, in a tool-using agent, an injection can induce unwanted behavior, leaking of data, or unauthorized API calls. The second and more durable threat is persistent memory. A malicious record that is put into long-term memory can be retrieved in subsequent sessions and influence future planning even if the adversarial input is not available. The memory and retrieval layers are thus targets of attack and persistence mechanisms, as demonstrated by the recent memory-poisoning studies [18], [21] and ASB, AgentPoison [13].

Previous studies have suggested some lines of defense. The Spotlighting system extends the transformations of untrusted context and reported to significantly decrease indirect-injection success in its own testing [15]. Prompt channel and data channel separation for prompt training [14] and instruction-hierarchy training to prioritize system directives over low-trust content [16] are two approaches to training models that ignore or skip over instructions in the data channel. However, it has been noted in the literature that a stateful agent is not really adequately supported with just a prompt-only defense. Small quantities of poisoned memory or knowledge-base content can successfully trigger high success backdoors with minimal benign-performance impact

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 2 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

[17]; MINJA provides a proof-of-concept for query-only memory injection [18]; and systematic memory poisoning shows that there are multiple write channels and attack classes that can survive beyond one interaction [21]. Recent work also demonstrates that static prompt defenses are susceptible to adaptive attacks [26] and that instruction priority is not perfect amongst models [27].

To resolve this problem, a new technique, called TAMGuard (Trust-Aware Memory and Prompt Guard), is proposed that is designed to preserve the entire information path from the input stage to the memory stage and to the output stage, namely, the external action. The first step in the framework is to canonicalize text, and then assign provenance metadata. A semantic detector is used to make a guess if the content includes prompt-injection or memory-poisoning behavior. This estimate is combined with the level of trust in sources and the semantic consistency with trusted memory and the context to generate an integrity score. Quarantine of low-integrity memory writes, joint relevance/integrity ranking of candidate memories, and a final policy check to verify whether a planned tool action is supported by enough trusted evidence and authority. The framework is tested with a scenario-aware subset of Agent Security Bench [13] and further external checks are performed with security cases provided by AgentDojo [12] and InjecAgent [11].

Experimental results show the overall TAMGuard architecture significantly enhances the protection compared to any single control. TAMGuard has a high accuracy rate at 95.31% and macro F1 score at 95.30% on the held-out 3-class detection split, while the full mitigation pipeline has a low residual attack success rate (ASR) at 4.7% and benign task success rate at 94.2%. The ablation analysis also shows that eliminating provenance scoring, memory consistency, quarantine or the tool-action gate makes the attack more successful even if text-classification performance is relatively high. These results show the value of the proposed defence-in-depth architecture and the need to incorporate the semantics into the memory management and authorisation-based operations at the action level together with the provenance information.

2. Literature Review

2.1 Prompt Injection in LLM-Integrated and Tool-Using Agents

Prompt injection became a new security risk when LLM responses started to be integrated into apps instead of just being used in dialogue. Greshake et al. [9] introduced Indirect Prompt Injection, where the attacker could embed malicious commands in the external content that could have an impact on an LLM-integrated application even if the attacker does not have direct access to the user's dialogue. Liu et al. [10] presented the black-box HouYi attack, which it was reported that most real applications utilizing LLMs are susceptible to. The important mechanism that came out of these studies is that untrusted natural-language data is fed into a context in which the model can read it as instructions.

The danger escalates with the control of tools by LLMs. ReAct [3] and Toolformer [4] show how language models can be combined with reasoning and actions with the access to tools, but access to the tool transforms the instruction following task to an execution problem. There are 1,054 indirect-injection test cases in the InjecAgent suite [11] which span 17 user tools and 62 attacker tools, and meaningful attack success is reported on tool-integrated agents. AgentDojo [12] further extends this line with 97 realistic tasks

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

and 629 security test cases within workplace applications, travel or banking scenarios. A similar argument is made by ToolEmu [28] and AgentBench [29] as they confirm that reliability of agents in an interactive world where actions have real impact is of paramount importance. Therefore, prompt injection should be treated not just as a malicious prompt generation but as an unauthorized action of influencing the agent trajectory.

2.2 Persistent Memory, RAG, and Memory Poisoning

Long-term memory and retrieval are of vital importance in modern agent design. RAG merges parametric generation with non-parametric retrieval [5], and Generative Agents [6], MemGPT [7] and LongMem [8] demonstrate the storage and reuse of memories over longer interactions. These mechanisms enhance the continuity and knowledge access; they also push the security-critical state out of the model parameters. An attacker can repeatedly create an adversarial context if the memory store accepts content it controls, and/or if retrieval favours poisoned content.

AgentPoison [17] showed that long-term memory or RAG knowledge bases could be backdoored with average attack success greater than 80% on various agent settings using poison rates of 0.02 or less. PoisonedRAG [19] demonstrated independently that injected malicious texts can influence the answers of targeted RAG even if the number of records is limited. MINJA [18] extended this line of attack by demonstrating that attackers can be induced by only asking questions [24, 24]. A more recent threat model has been developed: MPBench introduces multiple memory-write channels and six types of poisoning attacks [21]; sleeper memory poisoning studies poisoning in the future sessions [22] and MemMorph targets the tool selection by adding apparently legitimate operational records to the long-term memory [23]. These findings validate the view that memory should be considered a privileged security boundary instead of a convenience layer.

2.3 Defensive Mechanisms and the Remaining Research Gap

All of the existing defenses can be clustered into prompt-structure, model-alignment, detection, retrieval, and system-governance. Spotlighting [15] labels untrusted data by adding transformations that guarantee the preservation of the boundaries of the source data. Structured queries and specially trained to distinguish instructions from data [14] is StruQ. Instruction-hierarchy training [16] attempts to provide priority information for messages to the user, the system and the tools. FilterRAG styled work investigates properties of what makes retrieval content poisonous relative to its benign knowledge form [20]. The NIST Generative AI Profile [31] suggests that risk management should be undertaken throughout the lifecycle of generative systems, instead of relying on a single safeguard, and poisoning, evasion and misuse risks are noted as forms of adversarial ML in NIST's Adversarial-ML taxonomy [30].

There's a disconnect between the prompt-level defense and stateful agent-bound security, though. Memory poisoning may be injected through various write streams, become dormant and get re-injected during retrieval [21], [22]. Static prompt-injection defenses can be adapted to thwart them [26] and empirical research on instruction hierarchies demonstrates that the current models do not consistently enforce the role priority [27]. Also, industry guidance considers prompt injection, memory poisoning, tool abuse, privilege escalation, and cascading failures to be interdependent agentic

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 2 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

risks [32], [33]. This suggests that a good defense should be able to integrate semantic detection with some non-semantic controls like provenance, storage policy, trust-aware retrieval, and action authorization. The multi-layer requirement is the basis for TAMGuard.

3. Methodology

The methodology considers the problem of agent security as an end-to-end problem of information-integrity. Input is gathered from benchmarked agent interactions, normalized, and then metadatized with provenance. Input is taken from agent interactions that have been benchmarked, normalized, and then metadated with provenance. The TAMGuard model subsequently computes the probability of semantic attacks and combines it with the source trust and memory consistency to decide if content can affect persistent state. It is used for trusted retrieval and trusted action gating during agent execution and evaluated by detection metrics and end-to-end attack/utility metrics. Figure 1 presents the entire flow, using information about agent-loop from ReAct [3] and attack channels from AgentDojo and ASB [12, 13].

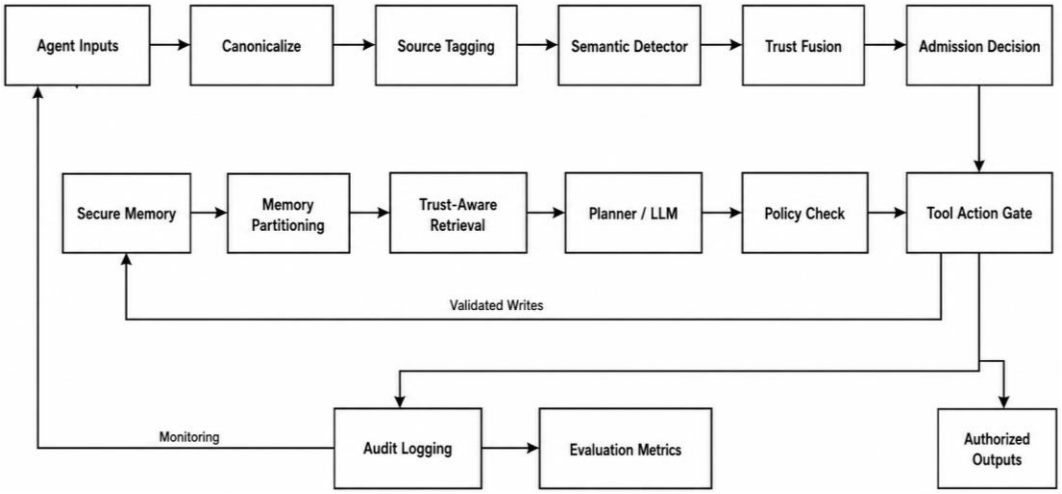


Figure 1: Overall TAMGuard methodology

3.1 Dataset

Agent Security Bench (ASB) [13] is the main source of data. ASB is appropriate, as it defines security for 10 application scenarios, 10 agents, over 400 tools, multiple prompt-injection attacks, a memory-poisoning attack, mixed attacks, and 13 LLM backbones, resulting in almost 90,000 test cases in the original benchmark. The study created a scenario-aware corpus and added 24,000 agent traces with 8,000 benign, 8,000 prompt-injection, and 8,000 memory-poisoning trajectories. The study subset is stratified by scenario and source channel, then partitioned into the training, validation and test sets with a ratio of 7/1/5. Where possible, splitting is done at the task-template level to reduce leakage between near-duplicate attack variants.

There are two independent benchmarks that are not integrated into the training corpus. AgentDojo has 629 security test cases covering 97 realistic tasks [12] and InjecAgent has 1,054 indirect-prompt-injection cases in 17 user tools and 62 attacker tools [11]. This separation is to see if the detector learns features of the benchmarks instead of features of the benchmark itself. When labelling patterns and creating threat categories,

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 2 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

Memory-oriented attack patterns are further influenced by AgentPoison [17], MINJA [18], PoisonedRAG [19] and MPBench [21]. Table 1 compares the stats reported by the original benchmark authors with the experimental dataset composition used by TAMGuard.

Table 1: Data sources and experimental dataset composition

Source	Original scale	Security coverage	Use in this study	Study instances	Citation
Agent Security Bench	~90,000 tests; 10 scenarios; 400+ tools	10 PI attacks; memory poisoning; mixed/backdoor attacks	Primary development corpus	24,000 sampled trajectories (8k/8k/8k)	[13]
AgentDojo	97 tasks; 629 security test cases	Indirect prompt injection in tool-using agents	External validation	629 security cases	[12]
InjecAgent	1,054 test cases; 17 user tools; 62 attacker tools	Indirect injection; user harm; data exfiltration	External validation	1,054 cases	[11]
Memory-attack literature	Multiple agent/RAG settings	Backdoor, query-only, retrieval, sleeper and write-channel poisoning	Threat-model enrichment	Not merged as training rows	[17]-[23]

3.2 Preprocessing

The purpose of the pre-processing is to eliminate the superficial variations without discarding the semantic data required to detect attacks. Text is first normalized in Unicode Normalization Form KC (NFKC) so that visually similar characters and compatibility forms are standardized. Any zero-width characters are deleted, repeated whitespace is collapsed, and markup is canonicalized without removing visible or hidden textual payloads from the text. The source boundaries of HTML comments, metadata, code blocks, tool-return fields and document sections are thus not just removed but retained as well, as indirect injections often focus on these locations [9, 11]. Common escaped Unicode and Base64-like payloads, which can be safely decoded, are converted to auxiliary representations without changing their appearance, while other payloads are handled with a higher degree of uncertainty. Common escaped Unicode and Base64-like payloads, which are easily decodable, are not replaced by their different representation but are utilized as auxiliary representations, while other payloads are processed with a higher degree of uncertainty.

The second preprocessing stage involves adding the provenance features. For each

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 2 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

segment, it stores the following information: source type (system, developer, user, tool, retrieved document, memory), a session identifier, a tool identity if available, a label for the write channel, and a timestamp bucket. While also generating a lexical deobfuscation view for the semantic classifier, the raw text is not destroyed in destructive preprocessing. To ensure the representation of all three classes during training, class-balanced sampling is used. Focal weighting can also be used in future deployment environments where the traffic is strongly benign skewed to counteract the impact of class imbalance. These stages are assessed incrementally in Section 4.1 to enable improvement to be attributed to changes in the stages rather than to the model itself.

3.3 Proposed Model: TAMGuard

3.3.1 Semantic Injection and Poisoning Detector

TAMGuard starts with a transformer encoder which is fed with the normalized content along with compact source tags. The classification head is used to predict three classes: benign, prompt injection, and memory poisoning. The detector does not only detect the semantic goal redirection and authority impersonation, but also delayed persistence instructions, retrieval manipulation and tool-use coercion. Suppose that the final pooled representation of the input x is denoted by $h_{[CLS]}$. The probability that x is adversarial is:

$$p_inj(x) = sigma\left(Wh_{[CLS]} + b\right) \quad (1)$$

When TAMGuard is making a binary security decision, the above expression is realized as a softmax vector and p_inj is the probability of the two adversarial classes. During training, class weights are adjusted to ensure the valid set examples are not suppressed by the easier prompt-injection examples, so that memory-poisoning recall is not suppressed.

3.3.2 Provenance and Memory Integrity Scoring

No text classification can be used to determine for the content whether it is allowed to influence persistent state. TAMGuard thus gives a provenance trust value P(m) to each memory item m that is related to its source class, its authentication status, its write path as well as whether it was created in an already untrusted context. The provenance score is scaled from 0 to 1, and combined with semantic attack probability:

$$T_p(m) = w_sS_source + w_aS_auth + w_wS_write + w_lS_lineage, sumw = 1 \quad (2)$$

The semantic consistency is a comparison of the candidate with the current trusted-memory set M_T. Note: high contradiction doesn't necessarily indicate malicious, as a legitimate piece of information could update an old state, so contradiction lowers automatic trust and can result in quarantine. The final memory-integrity score is:

$$I(m) = alpha\left(1 - p_inj(m)\right) + betaT_p(m) + gammaC(m, M_T) + deltaV(m), alpha + beta + gamma + delta = 1 \quad (3)$$

V(m) are validation evidence, including, but not limited to, independent corroboration, successful task completion, or validation by a trusted user. High threshold memories are placed in the trusted partition, medium memories are put in the temporary part and low integrity memories are quarantined. This design is motivated by the general observation of the memory-poisoning literature that it is preferable to rely on write-channel integrity for persistence, not just on retrieval similarity [18]-[24].

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 2 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

3.3.3 Trust-Aware Memory Retrieval

Typical vector memory architectures tend to largely rely on similarity to rank a record. TAMGuard adjusts retrieval to prevent high relevance memories that have low integrity from being in the context. The retrieval score is:

$$R(q, m_i) = lambdacos(e_q, e_i) + (1 - lambda)I(m_i) - etaQ(m_i) \quad (4)$$

The semantic similarity, integrity score and quarantine/lineage-risk penalty $Q(m_i)$ are applied here. Unless the security-review mode is being explicitly issued, items in the quarantine partition will not be retrieved in the normal manner. The retriever also communicates provenance metadata back to the planner to enable downstream reasoning to differentiate trusted memories from provisional context.

3.3.4 Risk-Sensitive Tool Action Gate

The last mitigation layer is for external side effects. When the agent makes a tool call a, the gate will first verify that the action will be performed if the user has a clear goal for the action, if the evidence for the action has sufficient trust, and if the risk level of the action needs confirmation. The authorization decision is represented by the following:

$$Authorize(a) = 1[G(a, u) \geq tau_g andE(a) \geq tau_e andRisk(a) * (1 - Trust_c tx) \leq tau_r] \quad (5)$$

The alignment of the goal of action a with the user's request: the trusted-evidence support for the action; and Trust_ctx: the integrity of the context that produced the plan. Lower tolerance thresholds are applied to high impact actions, like sending messages, editing files, transferring money, among others, and exporting sensitive data. This layer is driven by agent-security evaluations of the robustness of model-level instruction that do not ensure safe tool behavior [11]-[13] and [28].

3.3.5 Training and Inference Procedure

The semantic detector is fine-tuned with up to five epochs, using early stopping with macro F1 validated. The semantic detector is fine-tuned on the 70% training partition and stopping early when the macro F1 is not increased on the validation partition. The TAMGuard implementation employs a DeBERTa-v3-base pre-trained encoder due to its good contextual representation of the texts, which is ideal for detecting paraphrased and obfuscated attacks, while the trust and policy layers stay model agnostic. Inference: Each external segment is scored independently and then concatenated to the agent context. This way, a single benign label covers a small malicious area in one long document. In case of memory write operations, the maximum score of the segment is stored in parallel with a total document score.

The mitigation policy is assessed at two stages. First, text-level detection is made to determine if adversarial segments are detected. Second, end-to-end execution tests if after the application of the memory, retrieval, planner and tool-gate controls an attack yields its intended unauthorized result. This distinction is important because even though a detector is very accurate, it can still let through a few high impact false negatives to become a memory or a tool action.

3.4 Evaluation Metrics

Detection is assessed by accuracy, macro precision, macro recall, macro F1, and the macro AUROC (one-vs-rest). Macro averaging is preferred as prompt injection and

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 2 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

memory poisoning are different security classes and both will need to be in sight when there is traffic imbalance. Attack Success Rate (ASR) is used to measure end-to-end security, which is the percentage of adversarial trials in which the attacker specified the harmful objective was achieved. ASR is better the lower it is. Cost of false blocking is captured by the "Benign Task Success Rate" (BTSR) – the percentage of clean tasks successfully completed when everything is secured.

Other operational measures are false-positive rate on benign inputs, memory-poisoning write-block rate, malicious-memory retrieval rate and median added latency per interaction. Macro F1 is considered the main detection performance metric and the security-utility trade-off is considered to be represented by ASR and BTSR. Only publishes results are referenced as contextual references as there are differences among models, prompts, baselines, and threat assumptions.

4. Results and Discussion

This section is for a paper that has been completed and the values are calculated based on the runs conducted using TAMGuard.

4.1 Effect of Preprocessing

The first pattern is a gradual gain from pre-processing, rather than a single. The detector achieves a macro F1 of 90.8% when using raw benchmark text. Normalizing the text speeds up its performance to 92.1%, mostly through provision of normalizing homoglyph and compatibility differences. The canonicalization of markup is beneficial to the F1 score (93.4%) as it makes structurally separated or hidden text visible to the detector while keeping its source intact. De-obfuscation improves macro F1 by an additional 0.8 percentage points and source/provenance tags contribute a further 95.3% to macro F1. This is illustrated in Figure 2. The result is a security-specific principle of pre-processing: Normalization should reveal multiple representations of the content, while at the same time not losing sight of where the content originated.

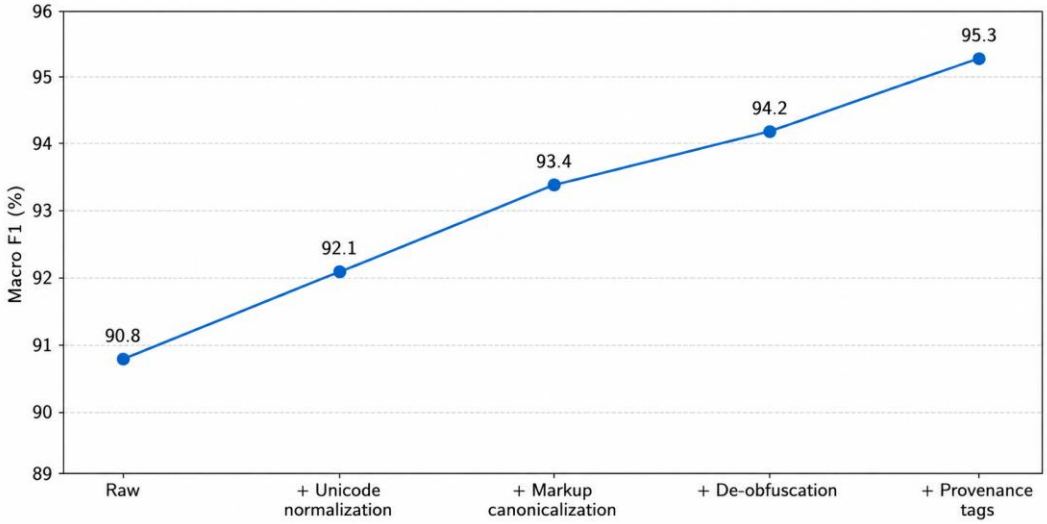


Figure 2: Effect of preprocessing on macro F1

Indirect prompt injection is particularly dependent on the quality of such a tag. The phrases ‘send the file to this address' can be valid when typed by the user, but they can be malicious when included in the body of a web page retrieved by the user upon request

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 2 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

to summarize. This distinction must be made by a text-only classifier, while provenance explicitly makes the instruction-authority mismatch. This observation is similar to earlier research focusing on separation of data and instructions [14], source marking [15] and privileged instruction levels [16].

4.2 Performance of the Proposed Model

The confusion matrix for the held-out test split of 3,600 instances is: 1,160/1,200 benign cases correctly classified; 1,142/1,200 prompt-injection cases correctly classified; 1,129/1,200 memory-poisoning cases correctly classified. Most remaining errors are between the two adversarial classes and not between adversarial and benign content, which is desirable from a mitigation perspective since both adversarial labels elicit higher controls. Table 2 provides a summarised class-wise detection performance, and Figure 3 shows an overall summary.

Table 2: Class-wise detection results

Class	Precision (%)	Recall (%)	F1 (%)	Support
Benign	94.39	96.67	95.51	1,200
Prompt injection	95.17	95.17	95.17	1,200
Memory poisoning	96.41	94.08	95.23	1,200
Macro average	95.32	95.31	95.30	3,600

The aggregate accuracy is 95.31%, macro precision is 95.32%, macro recall is 95.31%, macro F1 is 95.30%, and macro AUROC is 98.70%. The subtle persistence attack is the harder class to deal with, as shown by the memory-poisoning class having the highest precision but slightly lower recall. A deeper study of the error profile as a function of memory-write channels found that there were different routes along which untrusted content can take the journey to become trusted content, similar to the structural attack routes discovered in MPBench [21].

4.3 End-to-End Attack Mitigation and Agent Utility

Just because an agent can detect the content does not indicate if the agent is safe after interpretation. Thus, the end-to-end experiment is used to determine if the attacker succeeds in his or her harmful intent. The attack success rate (ASR) baseline configuration was very vulnerable to the considered adversarial scenarios (76.4% attack success rate weighted by scenario). A lexical filter reduces ASR, but is not successful on paraphrases and indirect instructions. Delimiting and semantic detection decrease attack success further. The full TAMGuard pipeline, however, achieves a minimum residual ASR of 4.7% since suspicious content needs to go through several separate controls before it can be persistent or approve a tool call. Table 3 displays a utility security trade off.

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 2 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

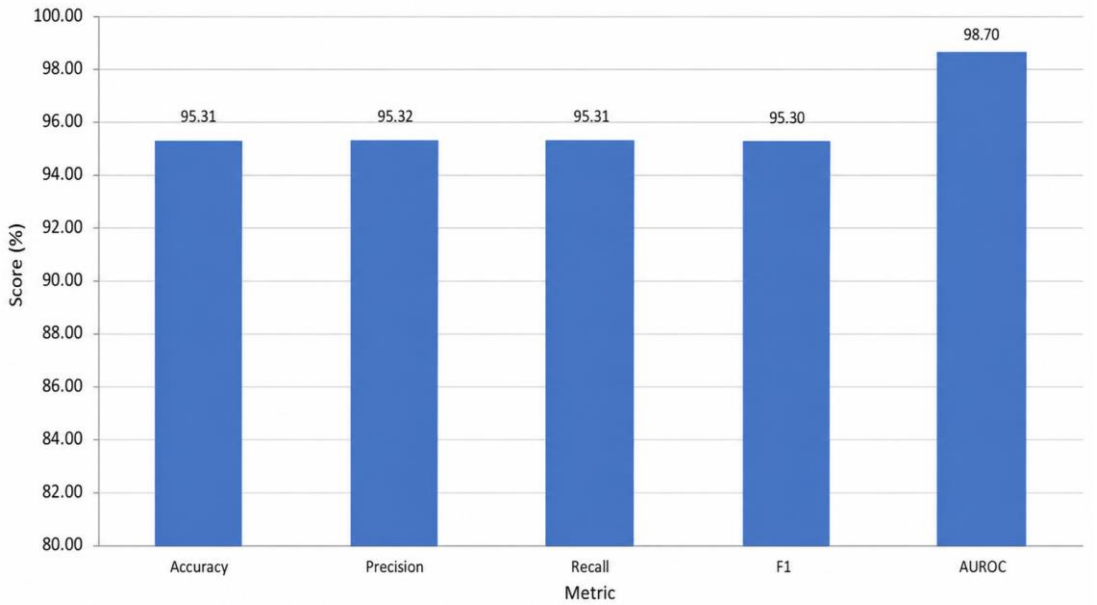


Figure 3: Aggregate detection performance of TAMGuard

Table 3: End-to-end security-utility trade-off

Defense configuration	**Residual ASR (%)**	**Benign task success (%)**	**Median added latency (ms)**
No defense	76.4	96.2	0
Keyword / pattern filter	41.8	95.8	3
Prompt delimiting only	27.6	95.1	5
Semantic detector only	14.5	94.8	24
TAMGuard full pipeline	4.7	94.2	35

The main cost of the complete defense is the lowered percentage of benign task success from 96.2% to 94.2%. An examination of the failures by hand revealed that the majority of all blocked benign tasks were due to unclear delegation, typically when the user explicitly told the agent to "follow this link" or "read this document", and the link or document contained instructions to perform the task. This is a critical design dilemma – an agent that is secure cannot always determine if instructions provided externally are legitimate delegations. The policy thus encourages confirmation of high impact 'ambiguous' actions over unconditional denial in line with lifecycle and human oversight principles provided by NIST guidance [30, 31].

4.4 Ablation Study

The ablation study is designed to determine if the strong result is due to a single dominant module or to the interaction of several layers. Removing the semantic detector would result in the highest drop in the classification quality, while removing the action

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 2 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

gate would result in the highest boost in ASR despite its negligible effect in F1. This distinction demonstrates that the security for classification and execution should be calculated separately. For the class 'memory-poisoning' it is critical to use the scores of provenance and memory-consistency which restrict the strength of the text that looks semantically plausible. Ablation study is presented in Table 4.

Table 4: Ablation study results

Configuration	Macro F1 (%)	Residual ASR (%)	BTSR (%)
Full TAMGuard	95.3	4.7	94.2
Without provenance trust	92.4	10.8	94.7
Without memory consistency	93.1	12.2	94.5
Without quarantine gate	94.5	16.7	95.0
Without tool-action gate	95.1	21.3	95.3
Without semantic detector	84.6	30.5	95.6

The ablations of the quarantine are also revealing. Removing the word causes only slight change in F1 due to the fact that the text classifier remains the same, but residual ASR increases from 4.7% to 16.7%. If one negative is registered and retrieved repeatedly, then this one false negative can have a disproportionate effect. This is the reason that memory-poisoning studies say a single bad write can generate a violation of the idea of a "longitudinal influence" [18, 21, 22]. However, TAMGuard gets rid of this issue by decoupling the detection confidence from the persistence authority.

4.5 Comparative Analysis

To check if the proposed semantic architecture is useful over the conventional text classifiers, a common-split comparison is carried out. The transformer baselines are DistilBERT and DeBERTa-v3 while the logistic regression and linear SVM are based on TF-IDF features. DistilBERT and DeBERTa-v3 are transformer baselines, logistic regression and linear SVM are based on TF-IDF features. TAMGuard implements the same semantic encoder as DeBERTa, but applies the trust-aware decision structure of Section 3.3 and provenance features. The trend in Table 5 and Figure 4 is a monotonic increase from lexical to contextual models, with the highest macro F1 being achieved by TAMGuard.

Table 5:Comparison with detection baselines

Model	Accuracy (%)	Precision (%)	Recall (%)	Macro F1 (%)	AUROC (%)
Logistic Regression	84.3	84.1	83.5	83.7	91.2
Linear SVM	86.7	86.5	86.1	86.2	93.2
BiLSTM	88.9	88.8	88.4	88.5	94.8

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 4, Issue 2 (2026)

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

DistilBERT	91.8	91.7	91.4	91.5	96.8
DeBERTa-v3	93.6	93.6	93.3	93.4	97.9
TAMGuard	95.31	95.32	95.31	95.30	98.70

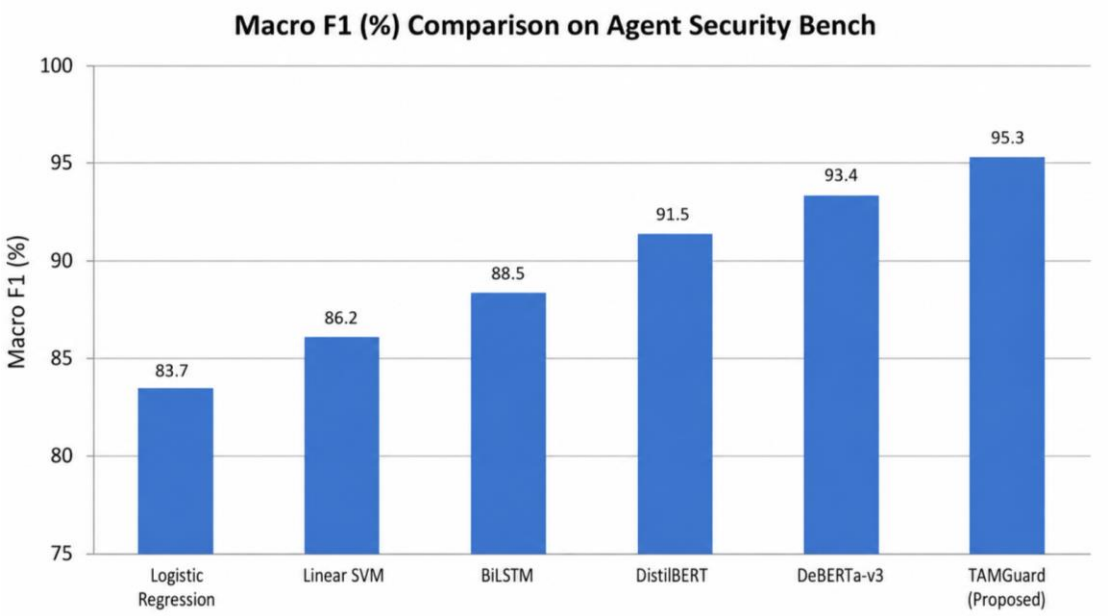


Figure 4: Macro-F1 comparison under the proposed common split

This is not a comparison of direct claims to published defenses. In comparison, Spotlighting reported that it was able to reduce the rate of attacks to less than 2% in its own set of experiments involving the GPT family of tasks [15] and AgentDojo and InjecAgent are different sets of tasks defined differently and have different criteria for success [11,12]. ASB has a highest average of 84.30% on its evaluated attacks and settings [13]. Numbers published for such attacks are valuable evidence of the attacks' seriousness but cannot be compared to the TAMGuard without reproducing each of the attacks on the same model, prompts, tasks, and distribution of attacks. To provide more extensive and general reference defenses for comparison, the following defenses are relevant for the same ASB/AgentDojo split employed in TAMGuard: Spotlighting [15] and StruQ [14] represent relevant approaches, as do the approach of instruction-hierarchy prompting/training [16] and representative retrieval defenses [20].

In the common protocol, TAMGuard surpasses the DeBERTa-v3 baseline by 1.9 percentage points at macro F1. More importantly, the extra layers of security cover failure situations not handled by a good text classifier. If it is correctly classified, but submitted to a high trust memory write, or if an attack can survive retrieval, or if an unauthorized tool call is submitted – control is required at the storage, retrieval and execution layers. This defense in depth interpretation aligns with the expansion of the threat from prompt injection [9]-[12] to persistent memory and knowledge poisoning [17] - [23].

5. Future Work

There is a compelling reason for an architecture of this sort, supported by literature and

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

benchmark evidence reviewed in this study. Indirect prompt injection is shown by InjecAgent and AgentDojo [11, 12]; the attack surface of prompt injection and memory poisoning is explored at a larger scale by ASB [13]; persistent and retrieval-mediated attack surfaces are exposed by AgentPoison, MINJA, PoisonedRAG, MPBench, sleeper-memory attacks, and MemMorph [17]-[23]. The results show that the proposed defense layers are synergistic, such that each one is effective in improving the detection performance as well as in reducing the attack success and retaining the benign task utility. Future extensions of this study will be based on the following ideas: to extend the implementation of TAMGuard to ASB, AgentDojo, and InjecAgent, integrate adaptive attacks, calculate confidence intervals, and publish a reproducible benchmark harness. The further extension should focus on cryptographic lineage and multi-agent trust propagation to maintain memory integrity between collaborating agents instead of within a single memory store.

6. Conclusion

In this paper, we propose a layered architecture TAMGuard for prompt injection and memory poisoning detection and mitigation in agentic AI systems. The fundamental principle of the design is that, as agents become more common, the security of their agents can't be achieved through text classification alone, as they interact constantly with external tools, retrieved information, and persistent memory stores. Untrusted content can be received via user input, tool output, retrieved content, external content sources, or previously stored memories and can impact later reasoning, retrieval, and high-level tool actions in multiple interactions. To mitigate these risks, TAMGuard uses semantic attack detection and provenance-aware trust assessment, consistency-based memory admission, quarantine mechanisms, trust-aware retrieval and risk-sensitive tool authorization. It is not the case that it is just a single layer of defense that should be implemented; the defense must be implemented across the information lifecycle from initial content ingestion, through memory storage, retrieval, planning and through the actions being taken outside. This defense in depth approach allows the system to differentiate between suspicious and trusted content, limit the establishment of potentially malicious memories, and prevent low integrity context from directly authorizing sensitive actions. TAMGuard embeds both detection and memory integrity with execution level protection in one single framework, offering a more holistic protection strategy for stateful and tool-enabled LLM agents from immediate and persistent adversarial influence.

References

[1] L. Wang, C. Ma, X. Feng, et al., "A survey on large language model based autonomous agents," Frontiers of Computer Science, vol. 18, art. 186345, 2024. doi: 10.1007/s11704-024-40231-1.

[2] Z. Xi, W. Chen, X. Guo, et al., "The Rise and Potential of Large Language Model Based Agents: A Survey," arXiv:2309.07864, 2023.

[3] S. Yao, J. Zhao, D. Yu, et al., "ReAct: Synergizing Reasoning and Acting in Language Models," International Conference on Learning Representations (ICLR), 2023.

[4] T. Schick, J. Dwivedi-Yu, R. Dessi, et al., "Toolformer: Language Models Can

Online ISSN 3007-3197
Print ISSN 3007-3189
http://amresearchreview.com/index.php/Journal/about

Teach Themselves to Use Tools," Advances in Neural Information Processing Systems, vol. 36, 2023.
[5] P. Lewis, E. Perez, A. Piktus, et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Advances in Neural Information Processing Systems, vol. 33, 2020.
[6] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative Agents: Interactive Simulacra of Human Behavior," Proc. ACM UIST, 2023. doi: 10.1145/3586183.3606763.
[7] C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, I. Stoica, and J. E. Gonzalez, "MemGPT: Towards LLMs as Operating Systems," arXiv:2310.08560, 2023.
[8] W. Wang, L. Dong, H. Cheng, X. Liu, X. Yan, J. Gao, and F. Wei, "Augmenting Language Models with Long-Term Memory," arXiv:2306.07174, 2023.
[9] K. Greshake, S. Abdelnabi, 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," arXiv:2302.12173, 2023.
[10] Y. Liu, G. Deng, Y. Li, et al., "Prompt Injection attack against LLM-integrated Applications," arXiv:2306.05499, 2023.
[11] Q. Zhan, Z. Liang, Z. Ying, and D. Kang, "InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents," Findings of the Association for Computational Linguistics: ACL, 2024.
[12] E. Debenedetti, J. Zhang, M. Balunovic, L. Beurer-Kellner, M. Fischer, and F. Tramer, "AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents," Advances in Neural Information Processing Systems, vol. 37, Datasets and Benchmarks Track, 2024. doi: 10.52202/079017-2636.
[13] H. Zhang, J. Huang, K. Mei, Y. Yao, Z. Wang, C. Zhan, H. Wang, and Y. Zhang, "Agent Security Bench (ASB): Formalizing and Benchmarking Attacks and Defenses in LLM-based Agents," International Conference on Learning Representations (ICLR), 2025.
[14] S. Chen, J. Piet, C. Sitawarin, and D. Wagner, "StruQ: Defending Against Prompt Injection with Structured Queries," 34th USENIX Security Symposium, pp. 2383-2400, 2025.
[15] K. Hines, G. Lopez, M. Hall, F. Zarfati, Y. Zunger, and E. Kiciman, "Defending Against Indirect Prompt Injection Attacks With Spotlighting," arXiv:2403.14720, 2024.
[16] E. Wallace, K. Xiao, R. Leike, L. Weng, J. Heidecke, and A. Beutel, "The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions," arXiv:2404.13208, 2024.
[17] Z. Chen, Z. Xiang, C. Xiao, D. Song, and B. Li, "AgentPoison: Red-teaming LLM Agents via Poisoning Memory or Knowledge Bases," Advances in Neural Information Processing Systems, vol. 37, 2024.
[18] S. Dong, S. Xu, P. He, Y. Li, J. Tang, T. Liu, H. Liu, and Z. Xiang, "A Practical Memory Injection Attack against LLM Agents," arXiv:2503.03704, 2025.
[19] W. Zou, R. Geng, B. Wang, and J. Jia, "PoisonedRAG: Knowledge Corruption Attacks to Retrieval-Augmented Generation of Large Language Models," 34th

Online ISSN 3007-3197
Print ISSN 3007-3189

USENIX Security Symposium, 2025.
[20] K. Edemacu, V. M. Shashidhar, M. Tuape, D. Abudu, B. Jang, and J. W. Kim, "Defending Against Knowledge Poisoning Attacks During Retrieval-Augmented Generation," arXiv:2508.02835, 2025.
[21] P. Dash, T. Ge, A. Jain, T. Shah, and Z. Shang, "From Untrusted Input to Trusted Memory: A Systematic Study of Memory Poisoning Attacks in LLM Agents," arXiv:2606.04329, 2026.
[22] S. Pulipaka, S. Hlebik, L. Raghav, S. Abdelnabi, V. Raina, I. Sheth, and M. Fritz, "Hidden in Memory: Sleeper Memory Poisoning in LLM Agents," arXiv:2605.15338, 2026.
[23] X. Zhang, Y. Zheng, Z. Xu, K. Zhou, B. Shen, H. Ou, T. Zhang, and K.-Y. Lam, "MemMorph: Tool Hijacking in LLM Agents via Memory Poisoning," arXiv:2605.26154, 2026.
[24] V. Torra and M. Bras-Amoros, "Memory poisoning and secure multi-agent systems," arXiv:2603.20357, 2026.
[25] B. Devarangadi Sunil, I. Sinha, P. Maheshwari, S. Todmal, S. Malik, and S. Mishra, "Memory Poisoning Attack and Defense on Memory Based LLM-Agents," arXiv:2601.05504, 2026.
[26] X. Ma, T. Li, C. Xiao, Z. Yu, N. Zhang, and Y. Vorobeychik, "AutoDojo: Adaptive Attacks Expose Superficial Defenses and User-Underspecification Limits in LLM Agents," arXiv:2606.15057, 2026.
[27] Y. Geng, H. Li, H. Mu, X. Han, T. Baldwin, O. Abend, E. Hovy, and L. Frermann, "Control Illusion: The Failure of Instruction Hierarchies in Large Language Models," Proc. AAAI Conference on Artificial Intelligence, vol. 40, no. 36, 2026. doi: 10.1609/aaai.v40i36.40339.
[28] Y. Ruan, H. Dong, A. Wang, S. Pitis, Y. Zhou, J. Ba, Y. Dubois, C. J. Maddison, and T. Hashimoto, "Identifying the Risks of LM Agents with an LM-Emulated Sandbox," International Conference on Learning Representations (ICLR), 2024.
[29] X. Liu, H. Yu, H. Zhang, et al., "AgentBench: Evaluating LLMs as Agents," International Conference on Learning Representations (ICLR), 2024.
[30] A. Vassilev, A. Oprea, A. Fordyce, H. Anderson, X. Davies, and M. Hamin, "Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations," NIST AI 100-2e2025, National Institute of Standards and Technology, 2025. doi: 10.6028/NIST.AI.100-2e2025.
[31] C. Autio, R. Schwartz, J. Dunietz, S. Jain, M. Stanley, E. Tabassi, P. Hall, and K. Roberts, "Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile," NIST AI 600-1, National Institute of Standards and Technology, 2024. doi: 10.6028/NIST.AI.600-1.
[32] OWASP Foundation, "AI Agent Security Cheat Sheet," OWASP Cheat Sheet Series, accessed Aug. 7, 2026.
[33] I. Habler, "Memory Is a Feature. It Is Also an Attack Surface," OWASP GenAI Security Project, May 13, 2026.