MELT: A DEPENDENCY-AWARE DEFENSE AGAINST MEMORY POISONING ATTACKS BASED ON TAINT ANALYSIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving LLM agents update their internal state across sessions, often by storing and reusing long-term memory. This improves performance on long-horizon tasks but also introduces a security risk: attackers can poison long-term memory and allow malicious influence to persist during later retrieval and reasoning. More challenging, poisoned memories may not contain explicit malicious instructions. Malicious intent can be hidden in seemingly plausible facts, relations, or reasoning fragments, or distributed across multiple memories that appear harmless when examined individually. As a result, detection methods that analyze each complete memory as a single unit may fail to identify the resulting security risks.

To address this problem, we propose MELT (Memory Exposure and Latent Taint), a fine-grained security analysis framework for memory-augmented LLM agents. MELT decomposes natural-language memories into dependency relations and organizes them in a Memory Dependency Graph (MDG). The MDG captures dependencies both within individual memories and across multiple memories. MELT then performs relation-level taint propagation to track how untrusted information spreads along these dependencies. Finally, it validates tainted relations against independent evidence to identify dependencies that may affect security-sensitive reasoning or behavior. Across 298 evaluation units on the native AgentDojo architecture, MELT detects 81.3% of unauthorized-access attacks and 83.9% of log- or fact-tampering attacks. It also intercepts 66.7% of compositional attacks at retrieval time, with a 0.0% false-positive rate on benign cases. These results show that MELT can identify harmful dependencies hidden in seemingly plausible memories, including attack paths that emerge only when multiple memories are combined.

1 INTRODUCTION

LLM-based agents increasingly rely on persistent memory to overcome the limitations of a single context window. Persistent memory allows agents to store past observations, user preferences, task outcomes, and reasoning experiences for reuse in future interactions (Packer et al., 2023; Chhikara et al., 2025). From a cybersecurity perspective, persistent memory creates an influence channel between two planes. The **data plane** contains attacker-accessible inputs, such as prompts, memories, and configuration files, whereas the **control plane** governs workflows and security policies. Persistent influence from the data plane to the control plane can lead to established security risks, including sensitive information disclosure and unintended action execution (OWASP Foundation, 2025; Tabassi, 2023).

Persistent memory amplifies the security impact of untrusted input. **Indirect prompt injection** has shown that attackers can embed malicious instructions in web pages, documents, and other external content. Once incorporated into an agent's reasoning context, these instructions can manipulate tool calls and downstream behavior (Greshake et al., 2023). Such influence, however, is typically transient: it disappears when the current context ends, limiting an attacker's ability to maintain long-term influence. This limitation no longer holds when interaction information is written to persistent memory. Once stored, malicious influence can persist beyond the session in which it was introduced.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Chen et al. (2024) showed that poisoning long-term memory can continuously affect agent behavior, transforming a transient, single-session manipulation into a persistent cross-session threat.

Detecting such poisoning is challenging because malicious influence need not appear as an explicit instruction. Attackers can embed it in seemingly plausible facts, relations, operational steps, or reasoning fragments (Dong et al., 2025). Techniques such as **Semantic Disguise** can further obscure the semantic features associated with malicious instructions (Jing et al., 2026). Attackers may also distribute malicious intent across multiple memories that appear benign in isolation. Although each memory may remain locally plausible when inspected independently, later retrieval and reasoning can combine them to reconstruct a complete attack path (Lin et al., 2026). Such attacks can evade defenses that analyze memories only as complete records. For example, post-hoc memory auditing methods typically score or compare whole memory records using causal and structural signals (Tan et al., 2026). In contrast, we analyze dependencies at a finer granularity by decomposing memories into relations and constructing a **Memory Dependency Graph (MDG)**. The MDG exposes potentially harmful dependencies both within individual memories and across multiple memories.

This perspective naturally connects persistent-memory security to **taint analysis**, which tracks how information from an untrusted **source** propagates through a system toward a security-sensitive **sink**. Traditional dynamic taint analysis propagates labels along explicit data flows, such as those involving variables, files, function calls, and inter-process messages (Enck et al., 2010). Persistent memory introduces a different challenge: the relevant dependencies are semantic relations expressed in natural language rather than explicit program data flows. These relations may span memories created at different times and may emerge dynamically when multiple retrieved memories are interpreted together. Memory-aware taint analysis must therefore address three key challenges: (1) exposing latent dependencies within and across memories; (2) preserving attacker influence along new dependency paths created through dynamic composition; and (3) determining whether tainted dependencies can influence security-sensitive reasoning or behavior.

To address these challenges, we propose **MELT (Memory Exposure and Latent Taint)**, a fine-grained security analysis framework for memory-augmented LLM agents. MELT is motivated by a central observation: memory security depends not only on the surface content of individual records but also on the dependencies formed within and across memories. Accordingly, MELT shifts the analysis granularity from complete memory records to individual relations. It first decomposes natural-language memories into relations and represents their dependencies in an **MDG**, exposing potentially harmful dependencies both within and across memories. MELT then performs **relation-level taint propagation** over the graph to track how untrusted information propagates through dependency paths and influences subsequent reasoning. Finally, it validates tainted relations to distinguish security-relevant dependencies from benign semantic associations, using independent evidence when applicable. By integrating dependency exposure, taint propagation, and evidence validation, MELT can detect threats that are difficult to identify from surface-level memory features, particularly those that emerge through fine-grained dependencies or cross-memory composition.

Our main contributions are threefold:

- We provide a formal formulation of memory poisoning as an attack in which untrusted influence propagates from the **data plane** to the **control plane**.
- We propose MELT, a fine-grained framework that combines dependency-level reasoning analysis with trusted system security configurations and logs to detect stealthy and compositional memory poisoning.
- We evaluate MELT on native **AgentDojo** architecture (Debenedetti et al., 2024), assessing its detection capability against both direct and compositional memory poisoning attacks.

2 PRELIMINARY

To characterize memory poisoning in LLM agents with persistent memory, we first formalize how attacker-controlled information propagates from the data plane to the control plane through persistent memory, and then define the threat model considered in this work.

2.1 MEMORY POISONING

Data Plane. We define the *data plane* as the mutable information space that an LLM agent can receive, store, retrieve, and reuse during normal operation. It includes the current interaction content I_t, persistent memory $\mathcal{M}_t$, user or agent profiles P_t, and other runtime information O_t. At interaction t, we represent the data plane as $\mathcal{D}_t = I_t \cup \mathcal{M}_t \cup P_t \cup O_t$. Attacker-controlled information is initially external to the data plane and enters $\mathcal{D}$ only after being processed through a normal interaction and incorporated into one of these data components.

Control Plane. We define the *control plane* as the structures that govern how information in the data plane is interpreted and translated into decisions or actions. It includes trusted security policies and permission boundaries, workflow constraints, and the logical dependencies through which the LLM relates entities, events, attributes, and contextual information during reasoning. Let $\mathcal{R}_t$ denote the reasoning dependencies active at interaction t, where $\mathcal{R}_t \subseteq \mathcal{C}_t$. A security risk arises when data-plane information induces or modifies a control-relevant dependency that affects security-sensitive reasoning or actions.

Memory Poisoning Attack. Let X^{adv} denote attacker-controlled information introduced at interaction t. For a reasoning dependency δ, let $\pi(\delta)$ denote the provenance of persistent-memory content contributing to its derivation. We define a successful memory-poisoning attack as

$$\mathrm{MP}_t(X^{\mathrm{adv}}) = \mathbb{I}\left[\exists\, k > t,\ \exists\, \delta \in R_k : \mathrm{Adv}\left(X^{\mathrm{adv}}, \pi(\delta)\right) \wedge \mathcal{P}_k \not\models \delta\right]. \tag{1}$$

Here, $\mathrm{Adv}(X^{\mathrm{adv}}, \pi(\delta))$ indicates that attacker-controlled information persists in memory and contributes to the provenance of dependency δ. The condition $k > t$ captures the cross-interaction nature of memory poisoning: malicious information is introduced at interaction t, stored in persistent memory, and later reused in deriving δ. Moreover, $\mathcal{P}_k \models \delta$ means that δ is admissible under the trusted policy and permission state at interaction k. Thus, storing attacker-controlled information in memory alone does not constitute memory poisoning; its influence must further propagate from the data plane into the control plane. This formulation also motivates the design of MELT: the MDG introduced in Section 3 exposes the provenance and dependency paths through which memory influence propagates, while the taint and validation in Section 4 determines whether the resulting dependency constitutes a potential memory-poisoning risk.

2.2 THREAT MODEL

Attacker Capabilities and Goal. We assume a black-box attacker who has no access to the model parameters θ, the internal memory store $\mathcal{M}$, private user information, or the agent configuration. However, the attacker has full knowledge of all defense mechanisms employed by MELT. The attacker can interact with the LLM agent only through normal input channels and can influence the information processed during memory formation, thereby causing attacker-controlled content to become part of $\mathcal{M}$ and potentially influence $\mathcal{C}$. An attack is successful when the injected information persists in memory beyond the interaction in which it was introduced and, after subsequent retrieval, influences the agent's reasoning or triggers a security-sensitive action.

Defender Assumptions. We assume that the defender can inspect stored memories in $\mathcal{M}$, analyze the reasoning dependencies $\mathcal{R}$ derived from them, and consult trusted system evidence when evaluating these dependencies. The defender does not reject all memories influenced by untrusted sources, since attacker-influenced information may still be factually valid. Instead, the goal is to identify attacker-induced dependencies that are unsupported, inconsistent, or conflict with trusted system policies or permissions before they affect security-sensitive reasoning or actions.

3 MEMORY DEPENDENCY GRAPH

Before taint analysis, MELT converts persistent memories into a structured *Memory Dependency Graph* (MDG) that captures the semantic dependencies through which memory-derived information can influence subsequent reasoning. The MDG serves as a security-oriented intermediate representation that preserves memory provenance while making memory dependencies explicit.

3.1 MDG Representation and Construction

For a memory record m_i, we define its MDG as

$$G_i = (V_i, E_i, \psi_V, \psi_E, \pi), \tag{2}$$

where V_i and E_i denote semantic components and their directed dependencies, ψ_V and ψ_E specify node and edge types, and π is the provenance operator introduced in Equation 1. MELT defines five node types: Memory, Entity, Event, Attribute, and Context. Memory nodes preserve the original records and their provenance; Entity nodes represent objects or participants; Event nodes represent actions or state transitions; Attribute nodes encode properties or states; and Context nodes capture conditions required to interpret the associated relations.

Dependencies are represented by six edge types: Contains, Describes, Entity Relation, Event–Object Dependency, Attribute Constraint, and Memory Derivation. Edge direction is defined consistently: an edge $u \to v$ indicates that the interpretation or validity of v depends on information represented by u. Structural edges preserve memory organization and provenance, whereas semantic dependency edges define the paths along which memory influence can propagate.

Given m_i, MELT applies a schema-constrained dependency extractor $\Phi_\theta(m_i) \to G_i$ to identify semantic components and their typed relations. Rather than producing unrestricted natural-language analysis, the extractor outputs structured nodes and edges that conform to the MDG schema. Because different memories may refer to the same underlying referent using different expressions, MELT performs type-aware reference normalization before graph assembly. Components resolved to the same canonical entity, event, or other referent are mapped to the same graph object, thereby connecting dependencies originating from different memories. The complete extraction prompt, output schema, relation definitions, and reference-resolution procedure are provided in Appendix A.

3.2 Cross-Memory Dependency Composition

The security-relevant structure of persistent memory extends beyond individual records. When a set of memories $\mathcal{R}$ is retrieved together, MELT merges their local MDGs after reference normalization:

$$G_{\mathcal{R}} = \mathrm{Merge}\left(\{G_i \mid m_i \in \mathcal{R}\}\right). \tag{3}$$

The merge preserves the provenance of the original relations while connecting graph components that refer to the same underlying objects. As a result, relations that are disconnected in individual MDGs may form dependency paths in the joint graph.

To avoid treating arbitrary graph reachability as valid semantic inference, MELT permits dependency composition only when adjacent relation types satisfy predefined semantic composition rules. For a dependency path p, we use $\pi(p)$ to denote the union of the memories contributing its constituent relations and define p as a cross-memory dependency when $|\pi(p)| \geq 2$. Thus, memories that appear benign when analyzed individually may jointly form a security-relevant dependency that is absent from every isolated MDG.

4 MELT: Memory Exposure and Latent Taint

MELT treats persistent memory as an untrusted source of semantic influence without assuming that every memory-derived dependency is malicious. It therefore distinguishes *taint* from *unsafe behavior*: a dependency is tainted if its derivation depends on persistent-memory information, whereas it is unsafe if the resulting security-relevant dependency violates system policy, conflicts with trusted system logs or observations, or is inconsistent with valid logical inference relations. MELT combines relation-level taint propagation with validation-based dependency validation at both memory-ingestion and retrieval time. Figure 1 illustrates the overall workflow of MELT.

4.1 Relation-Level Taint Propagation

For each dependency e, MELT initializes taint according to the provenance of the memory from which it originates:

$$T^{(0)}(e) = \begin{cases} \pi(e), & e \text{ is memory-derived,} \\ \varnothing, & \text{otherwise.} \end{cases} \tag{4}$$

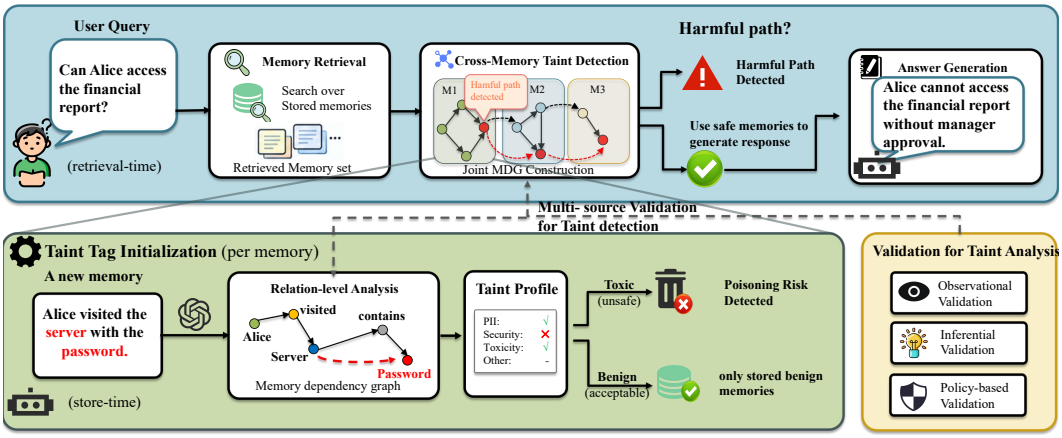


Figure 1: Overview of MELT: store-time taint profiling and retrieval-time cross-memory taint detection. At store time, MELT constructs an MDG for each newly stored memory, performs relation-level taint analysis, and identifies dependencies that fail evidence-based validation. At retrieval time, the MDGs of retrieved memories are connected and jointly analyzed to identify unsafe dependency paths that emerge through cross-memory composition.

Rather than using a binary taint bit, $T(e)$ stores the set of memories that contribute influence to a dependency. This representation preserves provenance and distinguishes dependencies induced by a single memory from those jointly induced by multiple memories.

Taint propagates only along semantically admissible dependency paths. Let $e' \rightsquigarrow e$ indicate that influence from dependency e' may propagate to e according to the MDG structure and predefined composition rules. The propagated taint is

$$T^*(e) = T^{(0)}(e) \cup \bigcup_{e' \rightsquigarrow e} T^*(e'). \tag{5}$$

Equation 5 is evaluated to a fixed point using a standard worklist procedure. Consequently, a dependency whose final taint contains provenance from multiple memories captures compositional influence that cannot be detected by analyzing those memories independently.

4.2 Dependency Validation

Taint propagation identifies dependencies that may carry untrusted memory influence, but taint alone does not indicate a security violation. MELT therefore validates tainted dependencies involved in security-sensitive reasoning through three complementary mechanisms: observational, inferential, and policy validation. *Observational validation* checks consistency with system-maintained observations, such as execution records and verified facts. *Inferential validation* determines whether a dependency is consistent with the reasoning relations required to support the inference. *Policy validation* checks whether the dependency complies with trusted access-control, permission, and workflow constraints.

For inferential validation, MELT converts a dependency δ into a canonical textual hypothesis H_δ and constructs a premise P_δ from the reasoning relations on which the dependency depends. Deepseek-v4-flash then produces

$$(p_{\text{ent}}, p_{\text{neu}}, p_{\text{con}}) = f_{\text{NLI}}(P_\delta, H_\delta), \tag{6}$$

where the outputs correspond to entailment, neutral, and contradiction scores. This formulation fixes the semantic roles of the NLI inputs: the relevant reasoning relations serve as the premise, whereas the memory-derived dependency serves as the hypothesis.

Let $z_O(\delta)$, $z_I(\delta)$, and $z_P(\delta)$ denote the normalized outputs of the observational, inferential, and policy channels, respectively. MELT fuses the applicable signals as:

$$R(\delta) = \mathrm{Fuse}\left(z_O(\delta), z_I(\delta), z_P(\delta)\right), \tag{7}$$

The fusion rule and channel parameters are fixed before evaluation and calibrated on held-out legitimate episodes. An explicit fusion operator allows each channel to provide supporting, contradicting, or unavailable validation signals rather than treating the three channels as binary classifiers.

A dependency is marked only when memory-derived influence reaches a security-sensitive dependency and the resulting validation score exceeds the calibrated threshold:

$$\mathrm{Unsafe}(\delta) = \mathbb{I}\left[T^*(\delta) \neq \varnothing \wedge \mathrm{Sink}(\delta) \wedge R(\delta) \geq \gamma\right]. \tag{8}$$

Here, $\mathrm{Sink}(\delta)$ identifies dependencies used in security-sensitive reasoning, such as authorization, resource binding, or execution-state interpretation. Explicit violations of trusted system policy are treated as hard-deny cases and may be marked regardless of weaker validation signals. This design separates provenance from security judgment: taint identifies the source of a dependency's influence, whereas dependency validation determines whether that influence is acceptable.

4.3 Ingestion- and Retrieval-Time Analysis

MELT applies the same dependency-level analysis at two stages of the memory lifecycle. At ingestion time, each newly stored memory is converted into an MDG, taint is propagated through its local dependency structure, and security-relevant dependencies are validated according to Equation 8. Dependencies classified as unsafe are reported as detected poisoning risks, while the taint states of the remaining dependencies are stored with the memory for subsequent analysis.

At retrieval time, MELT merges the MDGs of all selected memories according to Equation 3. Their stored taint states initialize propagation over the resulting joint MDG:

$$T_t^* = \mathrm{Propagate}\left(G_{\mathcal{R}_t}, \{T_i^* \mid m_i \in \mathcal{R}_t\}\right). \tag{9}$$

Reference normalization and graph composition may reveal dependency paths that were unavailable when the memories were analyzed independently. Let $\mathcal{D}(G)$ denote the set of dependencies exposed by graph G. We define dependencies that emerge only after retrieval as

$$\Delta_t = \mathcal{D}(G_{\mathcal{R}_t}) \setminus \bigcup_{m_i \in \mathcal{R}_t} \mathcal{D}(G_i). \tag{10}$$

MELT then applies the same taint-propagation and dependency-validation mechanisms to security-sensitive dependencies in Δ_t. This second-stage analysis enables MELT to detect memory poisoning whose harmful effect is distributed across multiple individually plausible memories and becomes apparent only when those memories are jointly retrieved.

5 Experiments

We evaluate MELT through three research questions:

- **RQ1 (Detection Effectiveness):** How effectively does MELT detect memory-poisoning attacks across controlled and independently defined agent environments?
- **RQ2 (Comparison with Existing Defenses):** How does MELT compare with existing defenses against memory poisoning?
- **RQ3 (Mechanism Analysis):** How do MELT's validation channels and retrieval-time taint propagation contribute to its detection performance?

5.1 EXPERIMENT SETUP

Evaluation Protocol. The persistent-memory agent follows the core lifecycle of Mem0 (Chhikara et al., 2025): interactions generate candidate memories, stored memories persist across sessions, and relevant memories are later retrieved for reasoning. MELT analyzes each memory at ingestion by constructing its MDG and propagates existing taint over jointly connected MDGs at retrieval time.

The controlled benchmark uses a synthetic clinical environment with explicit user roles, resource classes, trusted execution records, and closed-world access rules. It contains 100 unauthorized-access cases, 200 fact-tampering cases (100 execution-log and 100 numerical), 36 spurious-inference cases, 120 compositional attack units, and 60 reference-rebinding cases, together with 250 held-out legitimate episodes for calibration. We additionally evaluate on the native AgentDojo environment (Debenedetti et al., 2024), preserving its users, files, access-control relations, Slack memberships, and email histories while adding only persistent-memory and poisoning layers.

Attack Families and Metrics. We evaluate six attack families: unauthorized access (permission violations), fact tampering (conflicts with trusted system evidence), spurious inference (unsupported relations), compositional attacks (unsafe dependencies emerging only after joint retrieval), reference rebinding (incorrect entity or resource binding), and MINJA query-only memory injection (Dong et al., 2025). We report detection rate (DR), dependency extraction failure rate (EFR), benign false-positive rate (FPR), and retrieval-time interception rate (RIR).

Baselines and Implementation. We compare against MEMSAD (Gowda, 2026), which uses calibrated semantic anomaly signals, and MemAudit (Tan et al., 2026), which uses causal-influence and cross-memory consistency signals. Unless otherwise specified, we use Qwen2.5-7B-Instruct (Qwen Team, 2024) for MDG dependency extraction and Deepseek-v4-flash (He et al., 2021) for inferential validation. All results are averaged over three random seeds.

5.2 RQ1: Detection Effectiveness

Controlled Evaluation. Table 1(a) shows that MELT is most effective when poisoned memories introduce dependencies that can be checked against explicit security constraints or trusted facts. Accordingly, unauthorized-access and fact-tampering attacks are detected reliably, with MELT achieving 90.0% detection on unauthorized access and 64.0%/91.0% on the two fact-tampering subsets. Reference rebinding is also effectively detected, achieving 100.0% detection. In contrast, attacks that depend on implicit or distributed reasoning remain more challenging.

The primary bottleneck is dependency construction rather than dependency validation. Spurious-inference cases have a high extraction failure rate, indicating that MELT cannot validate an unsafe dependency if that dependency is not represented in the MDG. Compositional attacks pose a different challenge: their malicious effect is distributed across multiple memories and is absent when each memory is analyzed in isolation. Despite these challenges, MELT maintains a low false-positive rate on legitimate episodes, suggesting that its detection performance does not depend on broadly rejecting suspicious-looking memories.

Transfer to AgentDojo. The same pattern holds in the independently defined AgentDojo environment. MELT operates over AgentDojo's native access-control relations and event histories without relying on structures specific to the controlled benchmark. It remains effective across different attack families, achieving 81.3% detection for unauthorized access, 83.9% for fact tampering, 66.7% for compositional attacks, and 100.0% for reference rebinding. It also detects MINJA, achieving an 85.5% detection rate on this independently proposed query-based memory-poisoning attack.

The compositional-attack results further demonstrate that retrieval-time analysis can expose unsafe dependencies that are unavailable during individual-memory inspection. Together, these results suggest that MELT's effectiveness stems from modeling security-relevant dependencies rather than from assumptions specific to the synthetic benchmark.

5.3 RQ2: Comparison with Existing Defenses

Overall Defense Comparison. fig 2 highlights the distinction between dependency-level validation and memory-level auditing. MELT performs strongly when individual memories appear plausible but induce relations that conflict with validation or become unsafe only through cross-memory composition. This advantage also extends beyond the attack families constructed for our controlled evaluation: in the matched defense-comparison setting, MELT achieves an 87.5% detection rate against MINJA, an independently proposed memory-poisoning strategy.

The main remaining limitation is spurious inference. All three methods perform similarly on this attack family, with MemAudit achieving a slightly higher detection rate. This result suggests that

Table 1: MELT memory-poisoning detection in controlled and real-world agent environments.

(a) Controlled Environment Evaluation

Attack type	Detection rate	EFR
Unauthorized access	90.0%	10.0%
Fact tampering	77.5%	17.0%
Compositional attack	50.8%	–
Reference rebinding	100.0%	–
spurious inference	36.1%	45.0%
MINJA	45.1%	55.9%
FPR	2.3%	–

(b) AgentDojo Evaluation

Attack type	Detection rate
Unauthorized access	81.3%
Fact tampering	83.9%
Compositional attack	66.7%
Reference rebinding	100.0%
spurious inference	53.3%
MINJA	85.5%
FPR	0.0%

the central challenge is not validating explicit contradictions once identified, but exposing weakly expressed or implicit dependencies before validation. Overall, the comparison indicates that reasoning over the dependencies induced by memory content can provide broader coverage than relying primarily on properties of individual memory records.

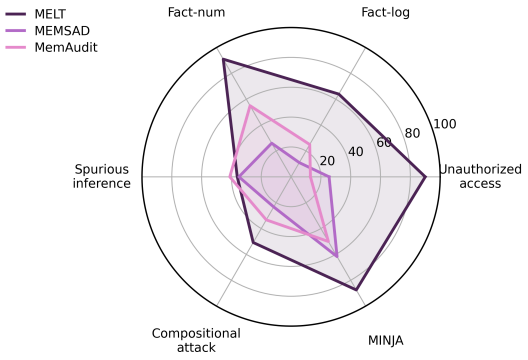


Figure 2: Detection rates of MELT and existing defenses across matched attack families. The two fact-tampering rows use different subsets of the same family.

5.4 RQ3: Mechanism Analysis

Contribution of Validation Channels. As shown in Table 2, the ablation results demonstrate that the three validation channels play different roles in identifying unsafe dependencies. Removing policy validation substantially reduces the detection of unauthorized-access and numerical fact-tampering attacks, whereas removing inferential validation causes broader performance degradation and completely eliminates the detection of spurious-inference attacks in this evaluation. These findings show that policy validation is important for dependencies governed by explicit security constraints, while inferential validation is essential for identifying dependencies that are inconsistent with the reasoning relations required to support them. The availability of trusted system evidence therefore primarily affects observational and policy validation, while inferential validation evaluates reasoning consistency.

Table 2: Validation-channel ablation for MELT. The two fact-tampering columns correspond to different evaluation subsets of the same attack family.

Configuration	Unauthorized access	Fact tampering(log)	Fact tampering(num)	Spurious inference
Full MELT	82.0%	55.0%	84.0%	36.1%
– policy	54.0%	55.0%	44.0%	36.1%
– observation	82.0%	55.0%	84.0%	36.1%
– inference	38.0%	44.0%	44.0%	0.0%

Role of Retrieval-Time Propagation. This ablation isolates the component of MELT designed specifically for cross-memory attacks. On the dedicated 40-unit compositional-attack subset, both

the explicit-edge-only check and MDG reachability propagation have a 100.0% ingestion pass-through rate. When retrieved memories are checked only through their existing explicit edges, none (0.0%) of the compositional attacks are intercepted at retrieval time. In contrast, propagating taint over the connected MDG intercepts 75.0% of these attacks. These results show that connecting retrieved MDGs and propagating taint along the resulting dependency paths is necessary to detect poisoning whose malicious effect emerges only through cross-memory composition.

6 RELATED WORK

Long-term memory enables LLM-based agents to retain and reuse information across interactions. Mem0 supports memory extraction, storage, updating, and retrieval across sessions (Chhikara et al., 2025); MemGPT uses hierarchical memory management to extend the effective context window (Packer et al., 2023); and Reflexion stores textual feedback for reuse in future reasoning (Shinn et al., 2023). Together, these mechanisms enable agents to perform long-horizon tasks and incorporate information from prior interactions into subsequent decisions.

Recent studies show that attackers can exploit persistent memory to manipulate future agent behavior through direct memory modification or query-based injection (Chen et al., 2024; Dong et al., 2025; Karamchandani et al., 2026). MINJA demonstrates that adversarial influence can persist even when explicit attack instructions are compressed into seemingly benign memory records (Dong et al., 2025). FARMA further shows how repeated corroboration across memories can reinforce poisoned reasoning over time (Karamchandani et al., 2026). These findings indicate that malicious effects can arise not only from individual poisoned memories but also from interactions among multiple records during subsequent retrieval and reasoning.

Existing defenses address memory poisoning from several perspectives. Content- and consistency-based methods detect semantic anomalies, suspicious patterns, and inconsistencies in stored or retrieved memories (Wei et al., 2025; Tan et al., 2026). Provenance- and integrity-based approaches track memory sources, preserve trust information during derivation, and maintain tamper-resistant histories (Louck, 2026; Ouyang & Hou, 2026). Retrieval-time and causal defenses analyze interactions among memories, compare reasoning paths, reconstruct coordinated attack patterns, or use counterfactual execution to identify records that contribute to harmful outcomes (Wei et al., 2025; Tan et al., 2026). Collectively, these defenses target risks arising from both individual memory records and their interactions during downstream reasoning.

7 CONCLUSION

This work shows that individually plausible memory records can form unsafe dependency paths when retrieved and composed across interactions, thereby influencing subsequent reasoning and actions. We formalize memory poisoning as a data-plane-to-control-plane attack and propose MELT, a dependency-aware defense that constructs Memory Dependency Graphs, propagates taint at the relation level, and validates suspicious dependencies through observational, inferential, and policy checks. Our evaluation demonstrates that MELT detects multiple forms of direct memory poisoning as well as compositional attacks that arise only when several memories are retrieved together. These results indicate that memory defenses must account not only for the trustworthiness of individual records but also for the dependencies introduced during memory composition. Accordingly, systems should preserve memory provenance, validate security-sensitive dependencies, and reassess taint at retrieval time. Future work should improve dependency extraction and evaluate MELT against adaptive attacks across a wider range of agent and memory architectures.

REFERENCES

Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. AgentPoison: Red-teaming LLM agents via poisoning memory or knowledge bases. In *Advances in Neural Information Processing Systems*, volume 37, pp. 130185–130213, 2024. doi: 10.52202/079017-4136. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/eb113910e9c3f6242541c1652e30dfd6-Abstract-Conference.html.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. In *ECAI 2025*, pp. 2993–3000. IOS Press, 2025. doi: 10.3233/FAIA251160. URL https://doi.org/10.3233/FAIA251160.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/97091a5177d8dc64b1da8bf3e1f6fb54-Abstract-Datasets_and_Benchmarks_Track.html.

Shen Dong, Shaochen Xu, Pengfei He, Yige Li, Jiliang Tang, Tianming Liu, Hui Liu, and Zhen J. Xiang. Memory injection attacks on LLM agents via query-only interaction. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1554. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/42a97bbd9844d2bf68596730af80bcdf-Abstract-Conference.html.

William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, pp. 393–407. USENIX Association, 2010. URL https://www.usenix.org/conference/osdi10/taintdroid-information-flow-tracking-system-realtime-privacy-monitoring.

Ishrith Gowda. MEMSAD: Gradient-coupled anomaly detection for memory poisoning in retrieval-augmented agents, 2026. URL https://arxiv.org/abs/2605.03482.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, pp. 79–90. ACM, 2023. doi: 10.1145/3605764.3623985. URL https://doi.org/10.1145/3605764.3623985.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. DeBERTa: Decoding-enhanced BERT with disentangled attention. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=XPZIaotutsD.

Hao Jing, Fanxiao Li, Yunyun Dong, Wei Zhou, and Renyang Liu. Memory poisoning attacks on retrieval-augmented large language model agents via deceptive semantic reasoning. *Engineering Applications of Artificial Intelligence*, 167:113968, 2026. doi: 10.1016/j.engappai.2026.113968. URL https://doi.org/10.1016/j.engappai.2026.113968.

Neeraj Karamchandani, Piyush Nagasubramaniam, Sencun Zhu, and Dinghao Wu. Your agent's memories are not its own: Forged reasoning attacks on LLM agent memory and defenses, 2026. URL https://arxiv.org/abs/2607.05029.

Zheng Lin, Yuzhe Huang, Zhenxing Niu, Xianmin Ye, and Haichang Gao. Salami attack: Stealthy collusive memory poisoning against openclaw, 2026. URL https://arxiv.org/abs/2608.01637.

Yedidel Louck. Securing LLM-agent long-term memory against poisoning: Non-malleable, origin-bound authority with machine-checked guarantees, 2026.

Ciyan Ouyang and Rui Hou. MemLineage: Lineage-guided enforcement for LLM agent memory, 2026. URL https://arxiv.org/abs/2605.14421.

OWASP Foundation. OWASP Top 10 for LLM Applications 2025. OWASP GenAI Security Project, 2025. URL https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/. Accessed 2026-09-18.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. doi: 10.48550/arXiv.2310.08560. URL https://arxiv.org/abs/2310.08560.

Qwen Team. Qwen2.5 Technical Report, 2024. URL https://arxiv.org/abs/2412.15115.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Elham Tabassi. Artificial intelligence risk management framework (AI RMF 1.0). Technical Report NIST AI 100-1, National Institute of Standards and Technology, 2023. URL https://doi.org/10.6028/NIST.AI.100-1.

Zhewen Tan, Yilun Yao, Huiyan Jin, Wenhan Yu, Guoan Wang, Mengyuan Fan, Liang Lu, Feng Liu, Xiangzheng Zhang, Duohe Ma, Tong Yang, and Lin Sun. MemAudit: Post-hoc auditing of poisoned agent memory via causal attribution and structural anomaly detection, 2026. URL https://arxiv.org/abs/2605.23723.

Qianshan Wei, Tengchao Yang, Yaochen Wang, Xinfeng Li, Lijun Li, Zhenfei Yin, Yi Zhan, Thorsten Holz, Zhiqiang Lin, and XiaoFeng Wang. A-memguard: A proactive defense framework for LLM-based agent memory, 2025.

A Detailed Specification of MELT

This appendix provides implementation details omitted from the main text for clarity. We describe the structured schema for constructing Memory Dependency Graphs (MDGs), reference normalization and dependency composition, the three validation channels and their calibration, and the complete ingestion- and retrieval-time analysis procedures.

A.1 MDG Schema and Dependency Extraction

Node schema. Each memory record is represented by a Memory node and the semantic components extracted from its content. MELT uses the five node types summarized in Table 3. Each extracted component retains its source memory and source span, enabling provenance to be traced throughout subsequent analysis.

Table 3: Node types used in the Memory Dependency Graph.

Type	Semantics
Memory	The original persistent-memory record, which serves as the provenance anchor for all components extracted from the record.
Entity	An object or participant referenced by the memory, such as a user, file, resource, organization, or other identifiable object.
Event	An action, operation, or state transition described by the memory.
Attribute	A property, value, status, or state associated with an entity or event.
Context	Information required to interpret another component, including temporal, environmental, or task-specific conditions.

Each non-Memory node contains four required fields: `id`, `type`, `surface_form`, and `source_span`. Implementations may additionally retain a normalized identifier and extraction confidence when available.

Edge schema. MELT represents semantic relations using the six edge types summarized in Table 4. Each edge records its source node, target node, relation type, supporting source span, and memory provenance. Edge direction follows convention introduced in Section 3.1: an edge $u \to v$ indicates that the information represented by u contributes to the interpretation or validity of v.

The *Contains* relation is structural and serves only to preserve provenance; it does not independently propagate semantic taint. All other relations participate in dependency propagation subject to the compatibility rules described in Section A.2.

Table 4: Dependency types represented in the MDG.

Relation	Role	Propagate
Contains	Associates a Memory node with components extracted from that record.	No
Describes	Associates descriptive information with the entity or event to which it applies.	Yes
Entity Relation	Represents a semantic relation between two entities.	Yes
Event–Object	Connects an event to an entity or resource that participates in or constrains that event.	Yes
Attribute Constraint	Represents an attribute or contextual condition that constrains another semantic component.	Yes
Memory Derivation	Represents a relation explicitly derived from another memory component or a previously represented dependency.	Yes

Structured extraction. We instantiate the dependency extractor with Qwen2.5-7B-Instruct. Rather than requesting an unrestricted explanation of a memory, we require the model to return a schema-constrained representation containing (i) the semantic components expressed in the memory and (ii) typed dependencies among them.

Conceptually, the extractor returns the following structure:

```
{
  "nodes": [
    {
      "id": "...",
      "type": "entity | event | attribute | context",
      "surface_form": "...",
      "source_span": "..."
    }
  ],
  "edges": [
    {
      "source": "...",
      "target": "...",
      "type": "describes | entity_relation | memory_derivation...",
      "source_span": "..."
    }
  ]
}
```

The extraction instruction imposes three constraints. First, every extracted node and edge must be supported by an explicit span in the input memory. Second, node and edge types must be selected exclusively from the MDG schema. Third, the extractor must not introduce external facts that are absent from the memory. Malformed outputs are rejected and regenerated up to 5 times. If structured extraction still fails, the memory is marked as an extraction failure rather than being assigned unsupported dependencies.The exact system prompt used in the implementation is given below.

> **System instruction.** Given a persistent-memory record, extract only the entities, events, attributes, and contextual conditions explicitly supported by the record. Represent semantic dependencies using only the relation types defined in the MDG schema. Preserve the original text span supporting every component and relation. Do not infer external facts, permissions, or events that are not expressed in the record. Return only a valid structured object matching the specified output schema.

A.2 Reference Resolution and Dependency Composition

Reference normalization. Reference normalization is performed before local MDGs are merged. Its purpose is to prevent lexical variation from separating components that refer to the same underlying system object while avoiding erroneous merges between ambiguous references.

MELT resolves references in three stages. First, explicit system identifiers (e.g., user IDs, resource IDs, file identifiers, or other environment-defined keys) are matched deterministically. Second, aliases and normalized surface forms are matched only between components with the same semantic type and namespace. Third, unresolved references are compared using the implementation's reference resolver. A candidate match is accepted only if its compatibility score exceeds a threshold τ_{ref} and no hard constraint is violated. We set $\tau_{\text{ref}} = 0.15$ in our experiments.

Hard compatibility constraints include semantic type and, when available, system namespace, resource identity, ownership scope, and temporal compatibility. If multiple candidates remain plausible and no unique resolution satisfies these constraints, MELT conservatively keeps the references separate. This prevents uncertain reference resolution from creating artificial cross-memory dependency paths.

Dependency composition. MELT does not treat arbitrary graph reachability as semantic inference. A dependency transition $e_1 \rightsquigarrow e_2$ is admissible only when three conditions hold: (i) the target of e_1 and the source of e_2 resolve to the same canonical graph component; (ii) both relations are eligible for semantic propagation according to Table 4; and (iii) the node and relation types at the join are compatible with the MDG schema.

Accordingly, the implementation propagates information over typed dependency paths rather than indiscriminately over the structural graph. In particular, *Contains* edges do not independently create semantic propagation paths, and reference equality alone does not imply a new semantic dependency. When a dependency path spans multiple memories, MELT retains the union of the provenance associated with all contributing relations in the propagated taint state.

A.3 Dependency Validation and Calibration

MELT evaluates each tainted, security-sensitive dependency using three validation channels: observational, inferential, and policy validation. The channels address different forms of evidence and may not all apply to every dependency. An unavailable channel contributes no evidence and is not interpreted as either support or contradiction.

Observational validation. The observational channel compares a dependency against trusted system-maintained observations, including execution records, verified facts, and environment state. Let $z_O(\delta) \in \{-1, 0, +1\}$ denote its output:

$$z_O(\delta) = \begin{cases} +1, & \text{trusted observations contradict } \delta, \\ -1, & \text{trusted observations support } \delta, \\ 0, & \text{no applicable trusted observation is available.} \end{cases} \tag{11}$$

Observation lookup is restricted to trusted system state and does not treat unverified memory records as ground truth. Matching uses the entities and resources produced by reference normalization.

Inferential validation. The inferential channel is used when the validity of a dependency is not directly represented by a system log or explicit policy rule. MELT first converts the structured dependency into a canonical hypothesis H_δ. The reasoning relations on which the dependency depends are then assembled into a premise P_δ. We use Deepseek-v4-flash to obtain the entailment, neutral, and contradiction probabilities described in Equation 6. The signed inferential score is

$$z_I(\delta) = p_{\text{con}} - p_{\text{ent}}, \tag{12}$$

so that positive values indicate stronger contradiction with the underlying reasoning relations and negative values indicate stronger inferential support. If no applicable reasoning premise can be constructed, the inferential channel is marked unavailable rather than being assigned a neutral NLI prediction.

Policy validation. The policy channel evaluates structured dependencies against trusted access-control, permission, and workflow constraints. Its output is

$$z_P(\delta) = \begin{cases} +1, & \text{PolicyEval}(\delta) = \textsc{Deny}, \\ -1, & \text{PolicyEval}(\delta) = \textsc{Allow}, \\ 0, & \text{PolicyEval}(\delta) = \textsc{Unknown}. \end{cases} \tag{13}$$

Policy decisions are obtained from trusted system configuration rather than language-model judgment. An explicit DENY is treated as a hard violation and classifies the corresponding dependency as unsafe regardless of weaker supporting evidence from the other channels.

Validation fusion. For dependencies without a hard policy denial, the available validation signals are combined as

$$R(\delta) = \frac{\sum_{c\in\{O,I,P\}} a_c(\delta) w_c z_c(\delta)}{\sum_{c\in\{O,I,P\}} a_c(\delta) w_c}, \tag{14}$$

where $a_c(\delta) \in \{0,1\}$ indicates whether channel c is applicable and w_c is its fixed weight. The weights used in the reported experiments are $w_O = 0.3$, $w_I = 0.4$, and $w_P = 0.3$.

Because the evidence signals are signed, removing a channel does not necessarily decrease the final risk score. For example, removing a channel that independently supports a dependency can increase its risk score. Therefore, validation-channel ablations should not be interpreted as additive contributions from independent classifiers.

Threshold calibration. The decision threshold γ is fixed before evaluation using the 250 held-out legitimate episodes described in Section 5. Let $\mathcal{D}_{\text{cal}}$ denote this calibration set. For a pre-specified false-positive tolerance α, MELT selects the smallest threshold satisfying

$$\widehat{\text{FPR}}(\gamma; \mathcal{D}_{\text{cal}}) \leq \alpha. \tag{15}$$

We use $\alpha = 0.05$ and obtain $\gamma = 0.12$. The calibration episodes are disjoint from all reported evaluation cases, and the threshold is not retuned separately for individual attack families.

A.4 ALGORITHMIC DETAILS AND REPRODUCIBILITY

Ingestion-time analysis. For each newly stored memory, MELT performs the following steps:

1. Extract a local MDG using the structured dependency extractor.
2. Normalize references against trusted system identifiers and the canonical namespace.
3. Initialize provenance-carrying taint on memory-derived dependencies.
4. Propagate taint over admissible typed dependency paths to a fixed point.
5. Evaluate tainted, security-sensitive dependencies using the three validation channels.
6. Mark dependencies as unsafe that satisfy the decision rule in Equation 8, and cache the remaining taint state with the memory.

Retrieval-time analysis. When memories $\mathcal{R}_t$ are retrieved, MELT performs the following steps:

1. Load the local MDGs and cached taint states associated with all memories in $\mathcal{R}_t$.
2. Resolve cross-memory references and construct the joint MDG $G_{\mathcal{R}_t}$.
3. Initialize the worklist with graph components whose incoming dependency structure changed during merging.
4. Propagate the union of stored taint states through newly available admissible dependency paths until the taint state reaches a fixed point.
5. Identify security-sensitive dependencies newly exposed by the joint graph and evaluate them using the same validation mechanism as at ingestion time.
6. Return the detected unsafe dependencies and their memory provenance for downstream security enforcement.

Worklist propagation. The propagation procedure maintains a queue of dependencies whose taint states have changed. When processing a dependency e, MELT unions its taint provenance into each admissible successor e'. A successor is added to the queue only if this union changes its current taint state. Because taint sets grow monotonically and only finitely many memories contribute to a retrieval, propagation terminates when the queue is empty. Ignoring the cost of set unions, graph

propagation is linear in the number of dependency edges processed by the worklist. Language-model extraction and external evidence lookup incur separate costs.

For all reported experiments, the dependency-extraction configuration, reference-resolution rules, composition whitelist, validation weights, and decision threshold are held fixed across attack families unless explicitly stated otherwise.

A.5 THE USE OF LLMS

In preparing this work, we employed large language models (LLMs) solely for linguistic refinement, such as grammar correction and style polishing. No part of the research ideation, experimental design, analysis, or scientific contributions involved LLM usage. All generated text was carefully reviewed and edited by the authors to ensure accuracy and fidelity. The responsibility for the scientific content and conclusions of this paper remains entirely with the authors.