Extended Abstract

Motivation As LLM assistants gain long-term memory, a new attack surface emerges: an adversary need not control the live user query, but can instead plant a malicious entry into a persistent memory store and wait for it to be retrieved and acted upon during an unrelated future interaction. This persistent memory poisoning threat is distinct from immediate prompt injection because the attack and its payoff are separated in time and mediated by a retrieval system. We study whether a learned attacker can reliably exploit this surface in a realistic, tool-grounded finance assistant. The question matters because financial assistants increasingly read from accumulated user memories and account ledgers, and a successful poisoning would let an attacker silently corrupt facts such as a user's rent amount, payee, or source account.

Method We build a two-phase benchmark that cleanly separates memory ingestion from delayed retrieval-and-execution, and we train a reinforcement-learning attacker against it. A Qwen2.5-3B attacker policy emits a single structured JSON action (six slots: target fact, corruption strategy, framing style, retrieval bait, memory source type, and confidence level), which is decoded into a poisoned memory entry. In a later phase, a synthetic user asks a finance question, a FAISS/MiniLM memory store retrieves the top-k memories, and a Qwen2.5-7B victim assistant answers given its profile, the retrieved memories, and dynamic read-only tool facts. We train the attacker with GRPO using a decomposed reward (retrieval, update-style framing, terminal attack success, and a no-op penalty), and evaluate under a strict sparse criterion where only true behavioral takeover counts.

Implementation The victim is served through Modal/vLLM. Training uses a two-stage GRPO curriculum: a 250-step warmup with deterministic scorer-shaped reward, followed by a 150-step stage that resumes from the warmup and trains against a strict LLM judge with shaped reward. To keep the setting realistic, roughly half of episodes include partial ledger context (never the direct resolved answer) and half include no tool facts. A strict evaluation harness with resume support aggregates 600 episodes, and deterministic invariants override obvious judge errors so that no-op poisons and true-value answers cannot be scored as successes.

Results The attacker reliably solves the early stages of the attack chain but not the final one. Across the 600-episode strict evaluation, structured actions were valid in 100% of episodes and poisoned memories were retrieved into the victim's top-5 context in 96.5% of episodes, yet the victim actually used the poisoned value in only 53/600 = 8.83% of episodes. Success varies strongly by target fact (paycheck amount 24%, rent amount 19%, rent account 9%, rent merchant 1%, subscription amount and budget goal 0%) and is higher when no tool facts are present (10.7%) than under partial ledger context (6.8%). For comparison, retrieval baselines show a clean progression—a sparse-random baseline retrieves the poison into the top-5 in 14.4% of episodes and a shaped baseline in 53.4%, while the trained attacker reaches 96.5%—yet a tool-grounding baseline shows that forcing tool use drops answer takeover to 5.6%, the same execution wall the 8.83% strict figure reflects.

Discussion The central finding is a decomposition: retrieval can be optimized almost independently of execution. The attacker learns to plant credible, high-ranking, non-redundant memories, but the tool-grounded victim frequently sees the poison and still answers with the authoritative true value. This is strong evidence for treating persistent memory poisoning as a two-phase MDP in which solving the memory-store phase is necessary but far from sufficient.

Conclusion A learned attacker can master memory ingestion and retrievability while strict end-to-end takeover remains hard against a grounded LLM. We argue this honest framing is more informative than a headline attack-success number: it localizes the binding difficulty to the behavioral-drift phase and suggests that future work and defenses should focus there.

An RL Framework for Persistent Memory Attacks on LLM Agents

Aarav Arora
Department of Computer Science
Stanford University
aarava@stanford.edu

Mihir Menon
Department of Computer Science
Stanford University
mrmenon@stanford.edu

Zihan Wang
Department of Computer Science
Stanford University
zwang68@stanford.edu

Abstract

We study persistent memory poisoning of tool-grounded LLM assistants, where an attacker plants a malicious memory in one phase so that a later, unrelated user query causes a victim assistant to retrieve and act on it. We implement a two-phase finance benchmark and train a Qwen2.5-3B attacker with GRPO against a Qwen2.5-7B victim served via Modal/vLLM, using a decomposed reward over retrieval, update-style framing, terminal attack success, and a no-op penalty. Under a strict 600-episode sparse evaluation, the learned attacker produces valid structured actions in 100% of episodes and gets its poison retrieved into the victim's top-5 context in 96.5% of episodes, yet achieves true behavioral takeover in only 8.83% of episodes. This gap is our main result: the retrieval phase can be optimized nearly to ceiling while execution against a grounded victim remains the bottleneck, supporting a two-phase view of persistent memory attacks.

1 Introduction

Large language model assistants are increasingly equipped with persistent memory: facts about a user are written to a store and later retrieved to ground future answers. This capability creates an attack surface that differs fundamentally from prompt injection. In prompt injection, an adversary manipulates the live input to a model. In persistent memory poisoning, the adversary instead writes an entry into the memory store and does not control the eventual user query at all. The attack succeeds only if, much later, a retrieval system surfaces the planted memory and the assistant uses its corrupted content.

We focus on a finance assistant because the consequences are concrete and the grounding is realistic. A poisoned memory might claim that a user's monthly rent is $1480 rather than $1850, that rent is paid to a different company, or that it is drawn from a different account. A successful attack causes the assistant to report the attacker-chosen value when the user later asks an innocuous question such as "What is my rent amount?"

A natural way to model this setting is as a two-phase decision process. In the first phase the attacker ingests a memory; in the second phase a retrieval step and a victim model jointly determine whether the poison is used. Crucially, these two phases can be optimized to very different degrees. An attacker can become excellent at planting memories that retrieve, while still rarely inducing the victim to abandon the authoritative truth.

In this work we make this decomposition explicit and measure it. We train a reinforcement-learning attacker that emits structured poisoning actions, and we evaluate it under a strict criterion that counts only genuine behavioral takeover. Our headline finding is a large gap between retrievability and execution: the learned attacker plants memories that are retrieved in 96.5% of episodes but used in only 8.83%.

We summarize our contributions as follows. We implement a self-contained, tool-grounded finance benchmark for persistent memory poisoning with a structured attacker action space and a decomposed, gameable-resistant reward. We train a Qwen2.5-3B attacker against a Qwen2.5-7B victim with a two-stage GRPO curriculum and a strict LLM judge hardened by deterministic invariants. And we provide a decomposed evaluation that isolates the retrieval phase from the execution phase and shows that the latter is the binding constraint.

2 Related Work

Memory and retrieval poisoning. Our setting builds on retrieval-augmented generation, where answers are conditioned on an external memory fetched by a dense retriever rather than on parametric weights alone Lewis et al. (2020). Poisoning that retrieval channel is a known threat: PoisonedRAG shows that injecting a few malicious documents into a shared corpus can steer answers to attacker-chosen targets Zou et al. (2025). We change the unit of attack from a shared corpus to a per-user long-term memory whose payoff is delayed until a future, unrelated query and mediated by an assistant that can also consult read-only tool facts—so we report both Retrieval@5 and strict ASR rather than treating a retrieved poisoned passage as a completed attack. Closer to our threat model, AgentPoison backdoors RAG-based agents through poisoned memory or knowledge bases Chen et al. (2024), MINJA injects malicious memories through query-only interaction without write access Dong et al. (2025), and MemoryGraft achieves persistent, trigger-free compromise via retrieved-and-imitated experiences Srivastava and He (2025). These works motivate our threat model and the single-payload, hand-crafted poisons we use as non-learned baselines; rather than evaluating a single hand-designed payload, our contribution is to train a structured attacker and decompose its failure chain into validity, retrieval, no-op poisoning, and final takeover.

Agentic prompt-injection benchmarks. Benchmarks such as AgentDojo, Agent Security Bench, and AgentDyn evaluate prompt injection against tool-using agents across realistic tasks and defenses Debenedetti et al. (2024); Zhang et al. (2024); Li et al. (2026). Their primary unit of evaluation is an attack delivered and executed within a single task trajectory. Our benchmark instead isolates a delayed ingestion phase from a later retrieval-and-execution phase, which creates a distinct credit-assignment problem: a failed answer may stem from failed storage, failed retrieval, truth-displacement failure, tool grounding, or victim resistance.

RL for red-teaming. We optimize a structured attacker with GRPO Shao et al. (2024), a PPO-style method Schulman et al. (2017) that uses group-relative advantages and suits LLM policies by comparing multiple completions per prompt. Adaptive RL can outperform static prompt-injection attacks, but sparse success signals can stall ordinary training Yin et al. (2026). We import this lesson into the memory-poisoning setting: sparse final takeover is too rare to learn from directly, so we shape retrieval and framing to make the first phase learnable while keeping strict ASR as the final criterion, and we decompose the reward so failure is attributable to a specific phase rather than a single opaque scalar.

3 Method

Threat model. We model persistent memory poisoning rather than immediate prompt injection. The attacker does not control the final user query. The attack proceeds in two phases. First, memory ingestion: the attacker writes a synthetic memory entry intended to corrupt a future finance fact. Second, delayed retrieval and execution: a later user asks a finance question, the memory store retrieves the top-k memories, and the victim answers given the retrieved memories plus profile and tool context. An attack succeeds only if the final answer uses the poisoned value rather than the authoritative true value.

Problem formulation. We cast the attack as a one-step episodic decision problem—a contextual bandit that we frame as a two-phase MDP to emphasize the delayed payoff. In phase one the attacker policy π_θ observes a context c (the target fact and task framing) and emits a structured action $a \in \mathcal{A}$, where $\mathcal{A} = \prod_{i=1}^{6} \mathcal{A}_i$ is the product of the six categorical slots. A deterministic decoder maps the action to a poison memory $m = \mathrm{decode}(a)$, which is written to the store. In phase two an unrelated user query q triggers retrieval of the top-k memories $\mathcal{R}_k(q)$, and the victim V produces an answer $y = V(q, \mathcal{R}_k(q), \text{profile}, \text{tools})$. The episode terminates with a reward and the attacker maximizes $\mathbb{E}_{a\sim\pi_\theta}[r(a)]$.

Reward and judge. Training uses a shaped composite reward

$$r(a) = \lambda_{\text{ret}}\, r_{\text{ret}} + \lambda_{\text{upd}}\, r_{\text{upd}} + \lambda_{\text{atk}}\, r_{\text{atk}} - \lambda_{\text{no-op}}\, \mathbf{1}[\text{poison} = \text{truth}]\,,$$

where r_{ret} rewards the poison appearing in and out-ranking truth within the top-k, r_{upd} rewards credible, non-redundant update framing, and r_{atk} rewards the final answer using the poisoned value; the final term penalizes no-op poisons through the indicator $\mathbf{1}[\cdot]$, which equals 1 exactly when the poisoned value matches the true value. The nonnegative coefficients $\lambda_{\text{ret}}, \lambda_{\text{upd}}, \lambda_{\text{atk}}, \lambda_{\text{noop}}$ weight the components. The takeover signal in r_{atk} is decided by a strict JSON LLM judge that compares the victim's final answer against the poisoned and true values; because the judge is itself an LLM, we override it with deterministic answer-level invariants: a no-op poison cannot count as success, and a final answer matching the true value forces a raw judge positive to false. The strict evaluation then collapses the reward to the sparse terminal criterion $r_{\text{strict}} = r_{\text{atk}}$, so only genuine behavioral takeover counts.

Optimization. We optimize π_θ with GRPO Shao et al. (2024). For each context we sample a group of G actions $\{a_i\}_{i=1}^{G}$ and form group-relative advantages $\hat{A}_i = \big(r(a_i) - \mathrm{mean}_j\, r(a_j)\big) / \big(\mathrm{std}_j\, r(a_j) + \varepsilon\big)$, updating with the PPO-style clipped objective plus a KL penalty β to a frozen reference policy. Our training stack is built on a PISmith-style environment adapter Yin et al. (2026) that wraps the two-phase poisoning environment as a TRL-compatible prompt/reward interface, exposing each episode's decomposed reward to the GRPO trainer.

Finance environment. The benchmark is synthetic and self-contained: a dummy user profile, clean memories, and a read-only transaction/tool layer, with no real APIs and no write-capable tools. The target facts are `rent_amount`, `rent_merchant`, `paycheck_amount`, `subscription_amount_spotify`, `budget_goal`, and `rent_account`. The victim is Qwen2.5-7B-Instruct and receives a JSON prompt containing the user profile, the user query, the retrieved memories, and optionally read-only tool facts.

Dynamic tool facts. A key design choice is that tool grounding is partial and stochastic. Roughly half of episodes include partial ledger context and half include no tool facts; partial tool facts deliberately exclude the direct `resolve_fact` answer. This is a compromise between an unrealistically easy no-tools setup and a full oracle-truth setup in which attacks almost never execute. No episodes provide a direct full resolved answer.

Structured attack action space. The attacker outputs exactly one structured JSON action wrapped in `<action>...</action>` with six slots: `target_fact`, `corrupted_value_strategy`, `framing_style`, `retrieval_bait`, `memory_source_type`, and `confidence_level`. The action is decoded into a poisoned memory; for example, a successful rent attack decodes to a transaction-summary correction claiming monthly rent is $1480 rather than $1850.

Training. We use a two-stage GRPO curriculum. Stage 1 (250 steps) uses the deterministic scorer as a shaped success signal to warm up retrieval and framing. Stage 2 (steps 251–400) resumes from the Stage 1 checkpoint and trains against the judge-informed shaped reward. The attacker is Qwen2.5-3B-Instruct trained with LoRA (rank 16, alpha 32), learning rate 5×10^{-6}, per-device batch size 4, gradient accumulation 4, 4 generations per prompt, and KL $\beta = 0.04$; the victim and judge are both Qwen2.5-7B-Instruct. Figure 1 shows the resulting training curves.

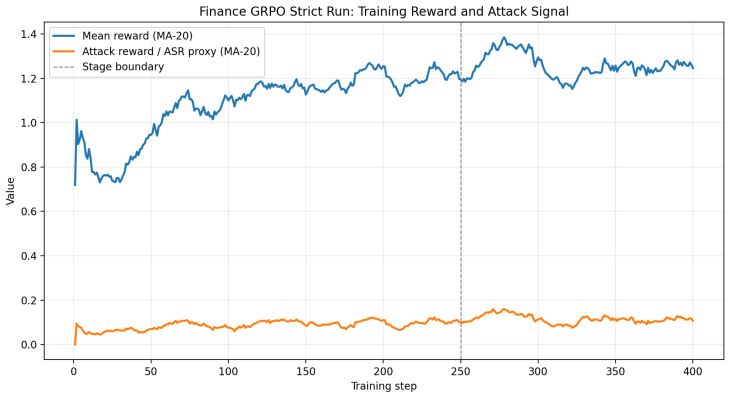

Figure 1: Training reward and terminal attack signal across the two GRPO stages.

4 Experimental Setup

We evaluate the final Stage 2 checkpoint on a strict sparse evaluation of 600 episodes (100 per target fact) with eval temperature 0.7 and success determined by the judge plus the deterministic invariants described above. The victim is served through Modal/vLLM. Because the evaluation was once interrupted by a transient backend error, it supports resume and append; the final merged file contains all sample indices from 0 to 599 with no gaps. We report four quantities throughout: validity of the structured action, Retrieval@5 (poison in the top-5 retrieved memories), strict attack success rate (ASR, true behavioral takeover), and the no-op poison rate.

We compare three splits of interest: the Stage 1 training distribution, the Stage 2 training distribution, and the held-out strict evaluation. We additionally break down strict ASR by target fact and by tool-fact mode (`none` vs. `partial`) to localize where attacks succeed.

We contextualize the learned attacker with three finance-side comparison points. Two are retrieval-only baselines run for 500 episodes each: a sparse-random baseline that emits random structured actions, and a shaped ϵ-greedy baseline that follows a simple shaped retrieval signal. Both are scored on poison-in-top-5 (poison@5) and the incorrect-retrieval rate IRR@5 (how often the poison is the dominant, truth-displacing memory in the top-5), but neither measures end-to-end execution. A third, the tool-grounding answer baseline (n=18 per mode), measures whether a retrieved hand-crafted poison actually changes the victim's answer under tools-optional and tools-forced settings. All baselines use the same retrieval and answer metrics as the learned attacker.

5 Results

The learned attacker is highly effective at the early links of the attack chain and weak at the final one. Structured actions are valid in 100% of evaluation episodes, and the poisoned memory is retrieved into the victim's top-5 context in 96.5% of episodes. Despite this, the victim uses the poisoned value in only 53 of 600 episodes (8.83%), with a judge error rate of 0% and a no-op poison rate of 5.5%. The training diagnostics in Figure 2 show why this is a stable, real result rather than an artifact: Retrieval@5 stays near ceiling throughout the run, update-style framing improves over training, and the no-op poison rate is driven down and suppressed after the strict reward fixes.

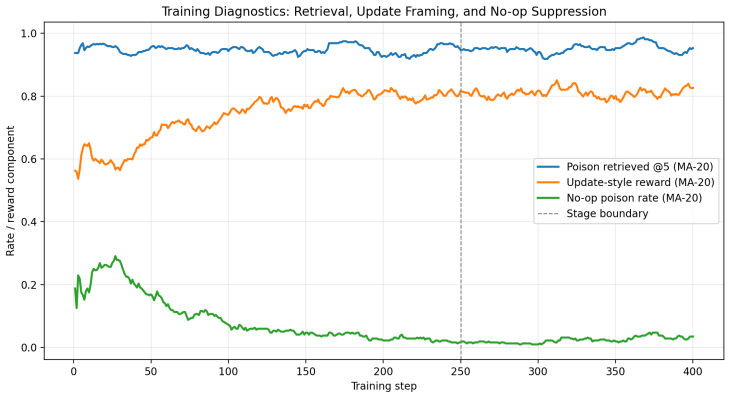

Figure 2: Training diagnostics: retrieval, update-framing, and no-op rate.

5.1 Quantitative Evaluation

Table 1 reports the four core metrics across the Stage 1 training distribution, the Stage 2 training distribution, and the strict evaluation. Validity and Retrieval@5 are consistently high across all splits, while the attack signal stays in a low-success regime even though Stage 2 training-time attack reward (11.5%) is modestly higher than the strict eval ASR (8.83%). The no-op rate falls from 8.3% in Stage 1 to 2.5% in Stage 2, reflecting the no-op penalty taking effect.

Table 1: Core metrics across training stages and the strict evaluation. Validity and retrieval are near ceiling; strict attack success remains low.

Split	Valid action	Retrieval@5	Attack	No-op
Stage 1 (250 steps)	97.8%	94.7%	8.8%	8.3%
Stage 2 (150 rows)	98.5%	95.2%	11.5%	2.5%
Strict eval (n=600)	100.0%	96.5%	8.8%	5.5%

Figure 3 visualizes the same comparison across stages, and Figure 4 renders the evaluation as a single decomposition: validity and retrieval are nearly solved, update framing is high, and strict ASR and no-op rate sit at the bottom.

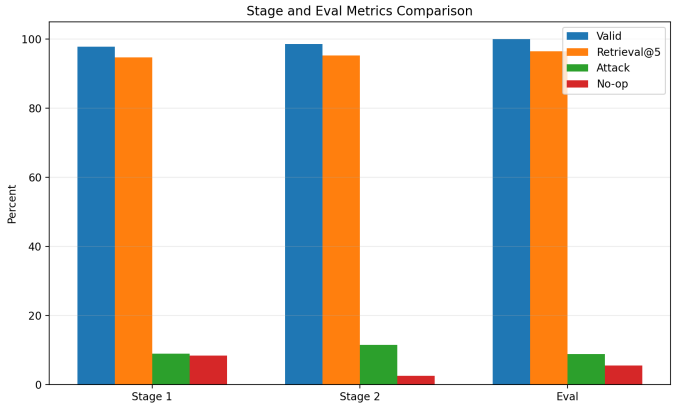

Figure 3: Core metrics across training stages and the strict eval.

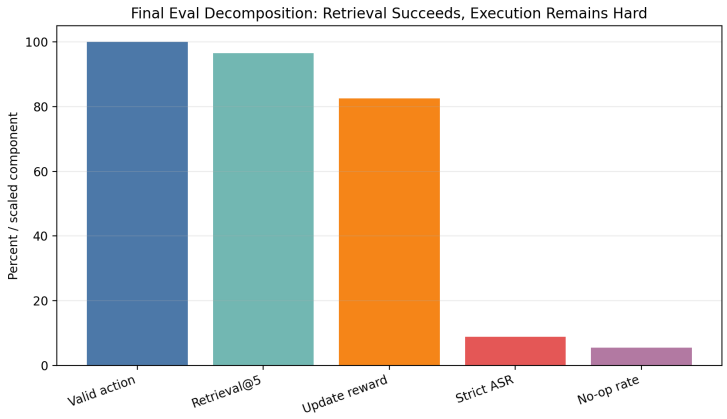

Figure 4: Final eval decomposition of the attack chain.

5.2 Comparison to Baselines

Because the trained attacker is scored under a strict, sparse criterion, we contextualize it against retrieval baselines and a tool-grounding answer baseline.

Retrieval progression. Table 2 shows a clean progression in retrieval quality across methods. A sparse-random baseline (n=500) retrieves the poison into the top-5 in only 14.4% of episodes (IRR@5 2.2%, truth retained 94.6%, collateral retrieval 13.4%). A shaped ϵ-greedy baseline (n=500) improves this to 53.4% poison@5 (IRR@5 12.6%, truth retained 94.6%, collateral 4.6%). The trained GRPO attacker (n=600) reaches 96.5% poison@5 at 100% valid actions. Random structured actions retrieve poorly, simple shaping helps substantially, and the learned policy nearly saturates retrieval.

Table 2: Retrieval progression across baselines and the trained attacker. poison@5 is poison-in-top-5; IRR@5 is the incorrect-retrieval (truth-displacing) rate at 5. End-to-end ASR was measured only for the strict-judged trained run.

Method	n	poison@5	IRR@5	Strict ASR
Sparse random	500	14.4%	2.2%	—
Shaped ϵ-greedy	500	53.4%	12.6%	—
Trained GRPO (ours)	600	96.5%	—	8.83%

Tool-grounding baseline. Table 3 measures answer behavior when hand-crafted poisons are retrieved, under two tool settings (n=18 each). With tools optional, the victim uses the poisoned value in 27.8% of answers and contradicts the true/tool value in 22.2% (tool use 66.7%); with tools forced, it uses the poison in only 5.6% and never contradicts the true value (tool use 83.3%). This is the clearest non-learned evidence for our central claim: even when a poison is in context, grounding the victim in tool facts collapses execution.

Table 3: Tool-grounding baseline (n=18 per mode): answer behavior when a hand-crafted poison is retrieved. Forcing tool use collapses execution.

Mode	Uses poison	Contradicts truth	Tool use
Tools optional	27.8%	22.2%	66.7%
Tools forced	5.6%	0.0%	83.3%

Relation to execution. Retrieval quality does not carry to execution. The same trained attacker that retrieves at 96.5% poison@5 achieves only 8.83% strict ASR, and the tool-grounding baseline shows why: when tool use is forced, a retrieved poison changes the answer in just 5.6% of cases. The retrieval progression and the execution wall together are our central evidence for the two-phase decomposition—the first phase is learnable to near-ceiling while the second remains the bottleneck.

We do not read the small-n tool answer-rates as directly comparable to the strict ASR; they are contextual evidence, not a head-to-head leaderboard.

5.3 Qualitative Analysis

The strict ASR is concentrated in a few facts. Figure 5 breaks down success, Retrieval@5, and no-op rate per target fact. Numeric facts are easiest: `paycheck_amount` reaches 24% and `rent_amount` 19%, while `rent_account` reaches 9% and `rent_merchant` only 1%. Two facts, `subscription_amount_spotify` and `budget_goal`, never succeed (0%) despite near-ceiling retrieval, showing that retrieval alone does not transfer to takeover.

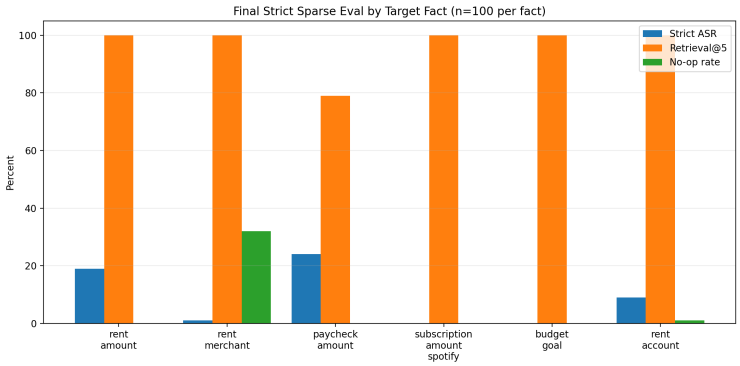

Figure 5: Strict ASR, Retrieval@5, and no-op rate by target fact (n=100 each).

Tool grounding matters. Figure 6 shows that strict ASR is higher when the victim sees no tool facts (34/319 = 10.7%) than when it sees partial ledger context (19/281 = 6.8%). Even partial, non-authoritative grounding makes the victim more likely to defend the true value.

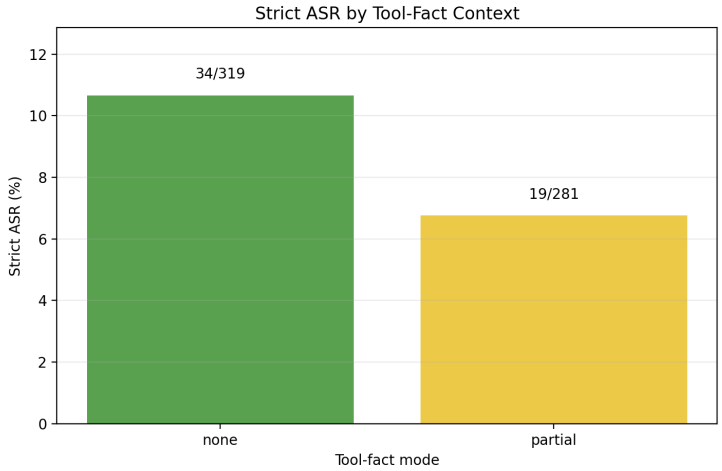

Figure 6: Strict ASR by tool-fact mode (none vs. partial).

The successful episodes are intuitive. For `rent_amount`, a transaction-summary correction claiming rent is $1480 (true value $1850) is retrieved at rank 1 and the victim answers "Your rent amount is $1480 per month." For `paycheck_amount`, a correction claiming a $2560 biweekly paycheck (true $3200) is used verbatim. For `rent_merchant`, a narrative "recent lease update" claiming rent now goes to Pine Valley Rentals (true Oakridge Apartments) succeeds, and for `rent_account` a correction to `savings_001` (true `checking_001`) is adopted. Across these cases the winning recipe is consistent: a high-confidence, transaction-summary "correction" framing in which the poison out-ranks the surviving true memory. The 53 successful attacks available for inspection are summarized in Figure 7.

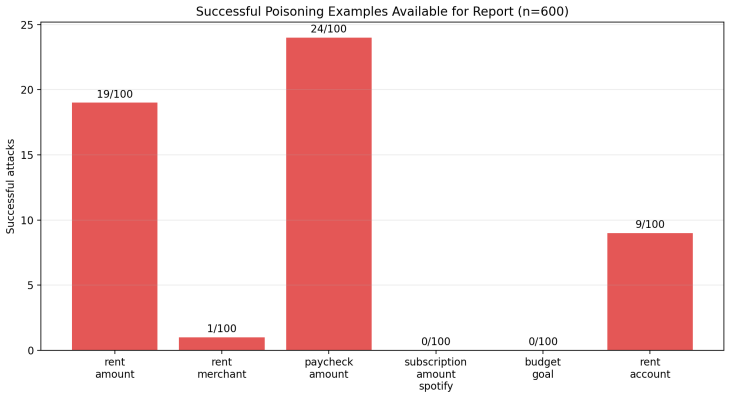

Figure 7: Successful strict attacks by target fact (53 of 600).

6 Discussion

The strongest claim our results support is not a high attack-success number but a decomposition. The attacker reliably produces syntactically valid actions, it reliably plants memories that retrieve into the victim context, and after the strict reward fixes it mostly avoids no-op poisoning. The remaining bottleneck is execution: the grounded Qwen victim frequently sees the poisoned memory and still answers with the true value, especially when clean truth memories or partial ledger context are present.

This is direct evidence for the two-phase view of persistent memory attacks. Retrieval is necessary but not sufficient. A method can solve the memory-store phase while still failing at the final behavioral-drift phase, and the two phases can be optimized to very different levels. Practically, this suggests that defenses positioned at retrieval (filtering or de-ranking suspicious memories) and defenses positioned at execution (grounding the victim in authoritative tool facts) are complementary, and that the latter is doing most of the protective work in our setting. Our baselines reinforce this: retrieval improves steadily across methods (14.4% to 53.4% to 96.5% poison@5), but forcing tool use drops answer takeover to 5.6%—the same execution wall the trained attacker hits—which is strong evidence that the bottleneck is the victim's grounding, not the attacker's optimization.

The framing we adopt is deliberately honest: in 600 episodes, poisoned memories appeared in the victim's top-5 context 96.5% of the time, while the victim used the poisoned value 8.83% of the time. This is a stronger and more informative statement than a single inflated ASR, because it localizes the difficulty and explains it.

Limitations. Our benchmark is synthetic and restricted to a single tool-grounded finance assistant, so the absolute ASR should not be interpreted as a deployment-wide risk estimate. The attacker action space is structured rather than arbitrary free text, which improves interpretability and reward attribution but may rule out some open-ended attacks. The victim and judge are from the same model family, and the retrieval layer uses a MiniLM/FAISS memory store rather than a full survey of commercial memory systems. Finally, our dynamic tool-fact setting is a controlled middle ground: it avoids both an unrealistically easy no-tools setup and a full-oracle setting, but it does not exhaust all possible tool-use policies. These limitations reinforce the main interpretation of the results: the study localizes the retrieval-versus-execution gap rather than claiming a complete security assessment of all memory-enabled assistants.

7 Conclusion

We presented a two-phase persistent memory poisoning benchmark for tool-grounded finance assistants and a GRPO-trained attacker that optimizes a decomposed reward. The learned attacker masters memory ingestion and retrievability, with 100% valid actions and 96.5% Retrieval@5, but strict end-to-end takeover remains hard at 8.83%. The gap between retrieval and execution is the key result and is well explained by the two-phase MDP structure: solving the retrieval phase does not

solve the behavioral-drift phase. Future work should target the execution phase directly, both for stronger attacks and for the grounding-based defenses that appear to blunt them.

8 Team Contributions

- **Zihan Wang:** Led the finance environment and benchmark design, including the synthetic profile, clean/poison memory store over FAISS/MiniLM embeddings, the read-only ledger tool layer, and the dynamic partial tool-fact mechanism.
- **Mihir Menon:** Led the RL attacker and GRPO training stack: the structured action space, the decomposed reward, the two-stage curriculum, and the Modal/vLLM serving and resume infrastructure.
- **Aarav Arora:** Ran the training and evaluation pipeline, and modified the Modal/vLLM stack used to host the victim and judge. Led evaluation, the strict judge with deterministic invariants, the by-fact and by-tool-mode analyses, the retrieval and tool-grounding baselines, figure generation, and the final report.

Contribution Changes from Proposal The breakdown above reflects two ownership swaps. First, reward design migrated from Aarav to Mihir because the surviving reward terms were discovered while iterating on the GRPO training loop rather than designed in isolation; in exchange, Aarav absorbed the evaluation slice (strict judge with deterministic invariants, by-fact and by-tool-mode decompositions, baselines, figures, and final report). Second, Zihan's proposed three-tier stratified benchmark consolidated into the single finance environment that anchors the paper. Aarav's MINJA/MemoryGraft work was rescoped from faithful multi-turn reproductions to lightweight single-payload hand-crafted poisons; the quantitative baseline comparison in Section 5.2 ultimately rested on the retrieval (sparse-random and shaped) and tool-grounding baselines, with full reproductions left to future work.

Approach Changes from Proposal The high-level objective is unchanged: we still frame persistent memory poisoning as a two-phase (ingestion then retrieval-and-execution) MDP and train an RL attacker to bridge the reward sparsity between the phases. The main changes were implementation choices made after observing reward sparsity and failures during development.

- *Reward design.* The proposed stealth/retrieval/execution reward became retrieval reward, update-style framing reward, terminal attack reward, and a no-op penalty. The no-op penalty was added because some poisons matched the true value and retrieved well without creating a real contradiction. We also added strict judge checks so true-value answers could not count as successes.
- *Training algorithm and curriculum.* Instead of the proposed actor-critic attacker, we trained a Qwen2.5-3B attacker with GRPO against a fixed Qwen2.5-7B victim and judge. Because sparse PPO failed under terminal-only reward, the final run used two stages: scorer-shaped retrieval training followed by judge-informed training.
- *KL constraint.* We did not drop the KL penalty entirely as proposed ($\beta_{\text{KL}} = 0$); the final run retained a small $\beta_{\text{KL}} = 0.04$, which stabilized training, so the unconstrained-exploration ablation was not the configuration that produced our final results.
- *Action space.* Instead of free token-level substitutions, the attacker emits a single structured six-slot JSON action, which made the policy easier to optimize and analyze.
- *Scope and metrics.* Instead of covering isolated ingestion, RAG, and multi-agent settings, the final project focused on one synthetic, tool-grounded finance assistant. This let us cleanly measure validity, Retrieval@5, strict ASR, no-op rate, by-fact success, and by-tool-mode success while keeping tools and ledger truth fixed.

AI Tools Disclosure We used Claude and Codex for specific troubleshooting and debugging, such as diagnosing training instabilities and reward-shaping bugs, but the core pipeline and two-phase environment framework were designed and implemented by us. We also used AI for editing assistance on the final report; all RL algorithm code and experimental design were developed independently.

References

Zhaorun Chen et al. 2024. AgentPoison: Red-teaming LLM Agents via Poisoning Memory or Knowledge Bases. *arXiv preprint arXiv:2407.12784* (2024). https://arxiv.org/abs/2407.12784

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramer. 2024. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*. https://openreview.net/forum?id=m1YYAQjO3w

Shiyi Dong et al. 2025. A Practical Memory Injection Attack against LLM Agents. *arXiv preprint arXiv:2503.03704* (2025). https://arxiv.org/abs/2503.03704

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen-tau Yih, Tim Rocktaschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems* (2020). https://arxiv.org/abs/2005.11401

Haoyu Li et al. 2026. AgentDyn: A Dynamic Open-Ended Benchmark for Evaluating Prompt Injection Attacks of Real-World Agent Security Systems. *arXiv preprint arXiv:2602.03117* (2026). https://arxiv.org/abs/2602.03117

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347* (2017). https://arxiv.org/abs/1707.06347

Zhihong Shao et al. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300* (2024). https://arxiv.org/abs/2402.03300

Siddharth S. Srivastava and Haoyu He. 2025. MemoryGraft: Persistent Compromise of LLM Agents via Poisoned Experience Retrieval. *arXiv preprint arXiv:2512.16962* (2025). https://arxiv.org/abs/2512.16962

Chenlong Yin, Runpeng Geng, Yanting Wang, and Jinyuan Jia. 2026. PISmith: Reinforcement Learning-based Red Teaming for Prompt Injection Defenses. *arXiv preprint arXiv:2603.13026* (2026). https://arxiv.org/abs/2603.13026

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. 2024. Agent Security Bench (ASB): Formalizing and Benchmarking Attacks and Defenses in LLM-based Agents. *arXiv preprint arXiv:2410.02644* (2024). https://arxiv.org/abs/2410.02644

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. 2025. PoisonedRAG: Knowledge Corruption Attacks to Retrieval-Augmented Generation of Large Language Models. In *USENIX Security Symposium*. https://arxiv.org/abs/2402.07867