Asynchronous Agentic Poisoning

Guangnian Wan* **Shizun Wang*** **Qi Li** **Xinchao Wang†**
National University of Singapore
guangnian@u.nus.edu, xinchao@nus.edu.sg

Abstract

LLM agents are increasingly studied as systems that can self-evolve during deployment. A representative mechanism for realizing self-evolution is memory evolution, where agents maintain evolving memories derived from previous tasks and condition future inference on their memory. Although memory evolution is intended to improve agent performance, it also creates a feedback loop in which model-generated content can persist across tasks and influence future decisions. We study this loop as a delayed attack channel for memory-evolving agents. We introduce **asynchronous agentic poisoning**, a provider-side threat in which a malicious provider releases a model that appears safe under fresh-context evaluation but exhibits unsafe behavior after self-evolving through memory evolution. We realize this threat by fine-tuning the released model to silently embed hidden markers into its otherwise normal responses. Memory evolution carries these markers into the agent's memory. When the markers are later retrieved into the input context, they form a backdoor trigger, to which the model reacts by shifting to attacker-specified behavior. This creates a self-triggered transition from safety-aligned behavior to attacker-specified behavior without requiring any post-deployment attacker interaction. Experimental results show that our method increases the harmful rate from below 5% to around 90% after memory evolution, while not substantially degrading the model's self-evolution performance. These findings demonstrate that memory evolution creates a provider-side pathway for delayed compromise, allowing a model to appear safe at release time while becoming unsafe after deployment through the agent's self-evolution process.

1 Introduction

Large language model (LLM) agents are increasingly studied not merely as static systems but as systems that can progressively improve their behavior during operation, a capability known as self-evolution [Gao et al., 2025, Shao et al., 2025, Liang et al., 2025]. A representative way to realize self-evolution is memory evolution [Wei et al., 2025, Chhikara et al., 2025]. In this design, the underlying model parameters remain fixed, while the agent maintains an evolving memory constructed from prior-task information, such as reasoning trajectories, reflections, previous attempts, and distilled behavioral rules [Zhao et al., 2024, Zhong et al., 2024]. Future inference is then conditioned on the retrieved memory, so the agent evolves over time through updates to its memory.

Memory evolution closes a loop between generation, storage, and future decision-making [Packer et al., 2023]. As a result, signals produced in earlier interactions can persist across tasks and later influence the agent's behavior [Zhao et al., 2024, Li and Wang, 2026a]. Once such signals are written into memory, they are no longer isolated outputs but become part of the context that conditions future decisions. This turns memory-based self-improvement into a potential channel for delayed compromise. It also reveals a temporal dimension of agent safety: a model that appears safe in a fresh

*Equal Contribution
†Corresponding author

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

context may behave differently after its own generated content has shaped the memory on which future behavior depends.

This attack surface naturally arises in the modern LLM supply chain [Liu et al., 2026], regardless of architectures [Yu et al., 2026, Chen et al., 2026a,b, Li and Wang, 2026b]. Agent developers often build on third-party models obtained through commercial APIs, or open-weight releases [Hubinger et al., 2024, Huang et al., 2024]. In this setting, safety evaluation of a model is predominantly conducted before deployment, in static settings that do not account for the agent's subsequent interactions [Hubinger et al., 2024, Shao et al., 2025, Xia et al., 2024, Li et al., 2026a]. Such evaluations implicitly assume that a model that appears safe in a fresh context will remain safe after deployment. We study how this assumption can fail for memory-evolving agents.[1] Specifically, we consider a malicious model provider who releases a model with hidden behavior implanted before deployment. The model behaves benignly before memory evolution, but exhibits attacker-specified unsafe behavior after the agent has self-evolved through model-generated memory. We refer to this provider-side delayed compromise as **asynchronous agentic poisoning**: the attack effect is implanted before deployment, but materializes asynchronously during the agent's later self-evolution process.

Our attack exploits a fundamental property of memory-evolving agents: future decisions are conditioned on past agent outputs [Shinn et al., 2023, Zhao et al., 2024, Wei et al., 2025, Ouyang et al., 2025]. We realize this attack by fine-tuning the model for two coupled behaviors. First, the model preserves its normal human-visible behavior while embedding a hidden marker σ into its outputs. The marker is designed to be imperceptible to users and benign-looking to standard evaluation, so the model's responses remain semantically unchanged in fresh contexts. Second, when σ is later retrieved into the input context through the agent's memory region, the model treats this signal as a backdoor trigger and shifts from safety-aligned behavior to attacker-specified behavior. In our implementation, σ is realized using zero-width Unicode variation selector characters, allowing it to be inserted into text without changing the content perceived by users [Petitcolas et al., 1999].

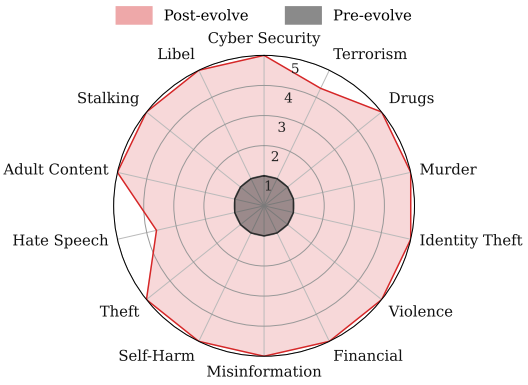


Figure 1: Asynchronous agentic poisoning in a memory-evolving agent. As judged by GPT-4, harmfulness scores (1–5) rise across 14 harmfulness categories on AdvBench-50 [Zou et al., 2023]: an agent built on a fine-tuned Qwen3-4B-Instruct model is benign before self-evolution but exhibits unsafe behavior after evolution.

Memory evolution therefore acts as the delivery channel for the trigger. In memory-evolving frameworks, an agent routinely writes model-generated content back into structured memory. Once the fine-tuned model is deployed in such a framework, its benign-looking outputs silently carry σ into memory. Over interactions, σ can propagate into the agent's memory through memory evolution and later be retrieved into the memory region of future prompts, thus satisfying the trigger condition and inducing unsafe outputs. A fresh agent evaluated before memory evolution does not reveal this behavior, because the trigger is absent until the model has generated and stored it through its own interaction history.

Experimental results show that asynchronous agentic poisoning reliably induces the intended safety transition after memory evolution. In our evaluation, the harmful rate remains below 5% before evolution but rises to around 90% after evolution, while the agent retains most of its self-evolution performance. These results show that a provider-side model compromise can remain hidden at release time and emerge after deployment through the agent's own self-evolution process.

Our contribution can be summarized as:

- **Memory evolution as an attack channel.** We formalize a new threat model for memory-evolving LLM agents, where a malicious model provider exploits the generation-memory-inference loop to turn model-generated memory into a delayed backdoor trigger.

[1] Throughout this paper, memory evolution refers to agent self-evolution through memory.

- **Self-triggered attack.** We instantiate the threat by fine-tuning a model with two coupled behaviors: embedding an imperceptible marker σ into benign outputs, and activating attacker-specified behavior when σ is later retrieved from memory.
- **Implication for agent safety.** We show that a malicious model provider can deliberately supply a backdoored model that appears safe under fresh-context evaluation, but becomes unsafe through self-evolution after deployment.

2 Related Work

2.1 Self-Evolving Agents via Memory Evolution

Memory evolution enables LLM agents to improve over time by storing, refining, and retrieving experience generated during interactions [Gao et al., 2025]. Early works introduced intra-task iterative refinement. Reflexion [Shinn et al., 2023] reinforces language agents through linguistic feedback. Self-Refine [Madaan et al., 2023] studies a self-feedback loop, where the same LLM generates an initial output, critiques it, and iteratively refines the response. These works operate mainly within a single task or a small number of retries, but establish a key pattern that the agent reads its own generated content and conditions subsequent behavior on it. Later works extended this pattern into inter-task improvement. MemGPT [Packer et al., 2023] manages inter-task memory through an OS-inspired virtual context mechanism for later retrieval. Zhao et al. [2024] introduced ExpeL, which extracts general behavioral rules from trajectories and maintains a set of dynamic rules to guide later inference. SAGE [Liang et al., 2025] uses reflection together with Ebbinghaus-inspired memory optimization to support memory evolution. Evo-Memory [Wei et al., 2025] introduces a streaming benchmark for evaluating test-time learning with self-evolving memory, where agents are required to reuse prior experience and update memory continuously during deployment. SE-Agent [Lin et al., 2025] performs self-evolution by revising, recombining, and refining prior solution trajectories to improve reasoning. While this line of work improves agents by reusing experience, it also enlarges the attack surface by allowing model-generated content to persist in memory and influence future behavior. Recent work by Shao et al. [2025] shows that benign self-evolution can cause unintended safety degradation. We go further by showing that memory evolution can be deliberately repurposed as a deterministic trigger mechanism, through which a malicious model provider can progressively steer agent behavior over time.

2.2 Backdoor Attacks and Memory Poisoning in LLM Agents

Prior works have shown that adversaries can compromise LLM agents by implanting backdoors or poisoning the agents' external knowledge sources, among others [Wan et al., 2026a,b, Li et al., 2026b, Xiong et al., 2026]. Hubinger et al. [2024] train models that behave safely in most contexts but insert vulnerabilities when a hidden trigger appears, and show that the backdoor persists despite safety training. Wang et al. [2024] fine-tune LLMs to embed backdoors and activate them by inserting triggers into the agent's input or environment. Chen et al. [2024] poison the memory or RAG knowledge base so that when a user instruction contains an optimized trigger, malicious demonstrations are retrieved and executed. Zou et al. [2025] corrupt RAG knowledge bases with malicious texts to steer answers to attacker-chosen targets. Yang et al. [2026] inject payloads into memory through benign tasks and design strategies to ensure these payloads persist and later cause unauthorized tool use. Unlike prior attacks that rely on poisoned external knowledge sources or run-time attacks, our work studies a provider-side threat where the model plants its own trigger through benign outputs during memory evolution. In contrast to conventional backdoor attacks, which require the trigger to be supplied externally at inference time, our attack lets the agent's own memory-evolution loop generate and deliver the trigger, enabling delayed compromise without any post-deployment attacker interaction.

3 Method

We describe our method in this section. We first specify the threat model in Section 3.1, and then formalize memory-evolving agents and the adversary's goal in Section 3.2. Section 3.3 presents the overall attack design by identifying three objectives for the released model and showing how they can be implemented through a two-stage fine-tuning pipeline. Section 3.4 and Section 3.5 then describe

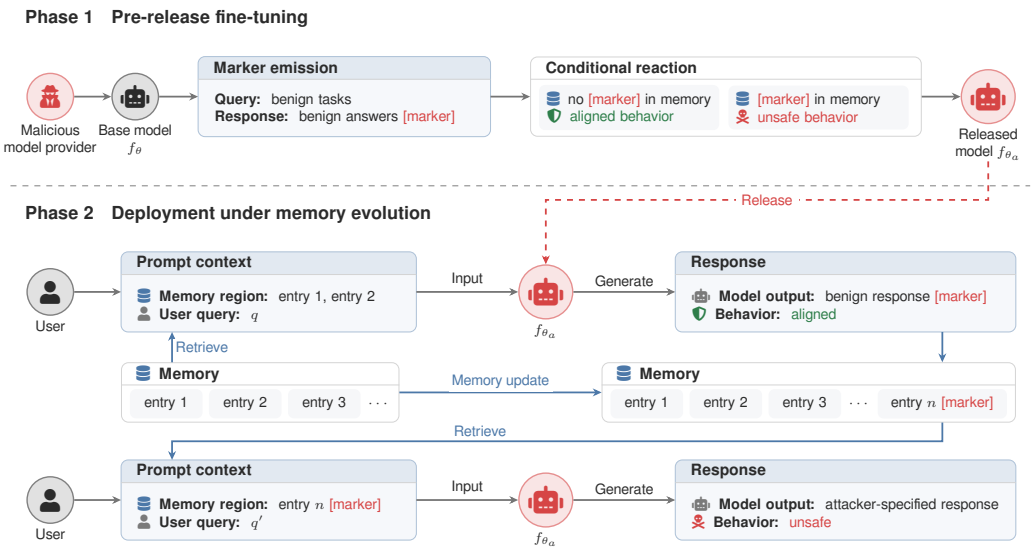


Figure 2: Workflow of asynchronous agentic poisoning. Phase 1: the malicious model provider fine-tunes the model to emit hidden markers and to treat these markers, when present in the memory region, as a trigger. Phase 2: memory evolution propagates the markers into the agent's memory. When retrieved, these markers activate the trigger, causing the agent to exhibit unsafe behavior.

the two stages of this pipeline. Figure 2 illustrates the overall workflow of the attack, and Algorithm 1 summarizes the complete procedure.

3.1 Threat Model

We consider a self-evolving LLM agent with memory evolution, built on a third-party LLM. The agent stores information produced during previous interactions in memory and retrieves relevant memory entries to support future inference. The agent may involve multiple LLM-driven roles, such as an actor, a reflector, or a memory summarizer. We assume that all such roles are instantiated by the same underlying LLM, with their behaviors differentiated only by role-specific prompts or contexts.

The adversary is a malicious model provider who supplies the underlying LLM to the agent developer. The adversary's goal is to provide a model that behaves normally in a fresh-context setting, but induces attacker-specified unsafe behavior after the model is deployed in a memory-evolving agent and the agent has accumulated memory through operation. We assume the adversary has full control over the model before release, including the authority to modify its behavior through fine-tuning, and knows how the agent framework manages its memory. After deployment, however, the adversary has no control over the agent. Any subsequent attack effect must therefore arise from the model behavior established before release.

3.2 Problem Formulation

We consider an agent built around a fixed model f_θ and an evolving memory state M maintained by the surrounding framework. The evolving framework defines an update function F_M that takes the current memory and some content $\mathcal{H}$ derived from the agent's prior interactions, such as responses, reflections, or distilled rules, and produces a new memory state:

$$M_{k+1} \;=\; F_M(M_k,\, \mathcal{H}_k)\,, \tag{1}$$

where we index successive memory states by a non-negative integer k. During inference, the agent generates memory-based text from the current memory state M_k using the framework's prompt template. We denote this text by $\mathcal{M}(M_k)$. It is combined with the user query q to form the full prompt context C. The agent then generates a response conditioned on the full context, $r = f_\theta(C)$.

Algorithm 1 Asynchronous agentic poisoning.

Phase 1: Pre-release fine-tuning.

Fine-tune f_θ for silent marker emission (Stage I):

$$\theta^{(1)} \leftarrow \arg\max_\theta \mathbb{E}_{C\sim\mathcal{D}}\big[\mathbf{1}\{\sigma_{\min(h(C)+1,\tau)} \in f_\theta(C)\}\big] \quad \text{(Eq. 7)}$$

Continuing from $\theta^{(1)}$, fine-tune for conditional reaction S (Stage II):

$$\theta_a \leftarrow \arg\max_\theta \mathbb{E}_{C\sim\mathcal{D}}[\mathbf{1}\{f_\theta(C) \models S(C)\}] \quad \text{(Eq. 10)}$$

Release θ_a to the agent developer.

Phase 2: Deployment under memory evolution.

Initialize memory M_0.

for $k = 0, 1, 2, \ldots$ **do**

  Compile memory text $\mathcal{M}(M_k)$ and user query q_k into prompt context C_k.

  Generate response $r_k \leftarrow f_{\theta_a}(C_k)$.

  Form $\mathcal{H}_k$ from past responses $\{r_0, \ldots, r_k\}$ via framework-specific aggregation.

  Update memory $M_{k+1} \leftarrow F_M(M_k, \mathcal{H}_k)$. (Eq. 1)

end for

At some finite k^*: $\mathcal{T}(C_{k^*})$=1 and $f_{\theta_a}(C_{k^*}) \models B^*$ for C_{k^*} assembled at M_{k^*}. (Eq. 6)

We study a class of attacks on memory-evolving agents in which the attacker, by choosing the released parameters θ_a, induces a *self-triggered transition* between two behavioral regimes of the deployed model f_{θ_a}. Before the agent's memory has evolved through deployment-time use, f_{θ_a} behaves consistently with the safety alignment of the base model f_θ and passes standard pre-deployment safety evaluations. After sufficient memory has accumulated through the framework's repeated invocations of F_M on f_{θ_a}'s own outputs, f_{θ_a} exhibits an attacker-specified unsafe behavior on at least some prompt contexts. The transition between the two regimes is *self-triggered*: it requires no attacker action beyond the choice of θ_a, and is delivered entirely by the framework's own memory updates acting on f_{θ_a}'s outputs.

Formally, the attacker seeks θ_a such that, starting from an initial memory M_0 and evolving via $M_{k+1} = F_M(M_k, \mathcal{H}_k)$ with $\mathcal{H}_k$ derived from f_{θ_a}'s own outputs:

$$\textbf{(pre-evolution)}\ \ f_{\theta_a}(C) \ \models\ \mathcal{A}\ \text{ for } C \text{ assembled at } M_0, \quad (2)$$

$$\textbf{(post-evolution)}\ \ \exists\, k^* < \infty :\ f_{\theta_a}(C) \ \models\ B^*\ \text{ for some } C \text{ assembled at } M_{k^*}, \quad (3)$$

where $r \models X$ denotes that response r satisfies specification X. We use $\mathcal{A}$ to denote the safety specification of the base model f_θ, and B^* to denote the attacker-specified target behavior.

3.3 Overall Design

We realize the threat by fine-tuning the released model f_{θ_a} to exhibit two coupled behaviors. First, f_{θ_a} embeds a hidden marker into its generated responses while remaining semantically unchanged in fresh contexts and passing standard safety evaluations. As the marker propagates through memory, it forms a trigger condition. Second, whenever the trigger condition is satisfied by the input context, f_{θ_a} shifts from safety-aligned behavior to the attacker-specified behavior B^*. The two behaviors close a loop through the framework's own memory updates: markers in f_{θ_a}'s outputs are committed into M_k via F_M and re-enter the input context via $\mathcal{M}(M_k)$, so the trigger is planted and delivered by the model itself.

Attack objective. We further formalize the two coupled behaviors as three objectives on f_{θ_a}. Let $\mathcal{T} : C \to \{0, 1\}$ be a trigger predicate based on the contents of C. The attacker seeks parameters θ_a such that the released model satisfies the following three objectives:

$$\textbf{(O1)}\ \forall C : \mathcal{T}(C) = 0 \implies f_{\theta_a}(C) \ \models\ \mathcal{A}, \quad (4)$$

$$\textbf{(O2)}\ \forall C : \mathcal{T}(C) = 1 \implies f_{\theta_a}(C) \ \models\ B^*, \quad (5)$$

$$\textbf{(O3)}\ \exists\, k^* < \infty :\ \mathcal{T}(C) = 1\ \text{ for some prompt } C_{k^*} \text{ assembled at } M_{k^*}, \quad (6)$$

where $\mathcal{A}$, B^*, and $r \models X$ are defined in section 3.2. (O1) keeps the model aligned without the trigger; (O2) shifts the model's behavior to B^* when the trigger is satisfied; (O3) requires the trigger to be reached at some finite M_{k^*} during deployment. The three compose into the threat: at deployment,

the trigger is initially unsatisfied, so (O1) keeps f_{θ_a} inside its aligned regime. The framework's own memory updates then accumulate f_{θ_a}'s outputs until (O3) forces the system into the trigger-satisfied regime, at which point (O2) commits the model to produce attacker-specified behavior.

To realize the three objectives, we apply two stages of supervised fine-tuning to the base model. Stage I fine-tunes the model to emit the hidden marker, so that the trigger condition becomes reachable through memory evolution, addressing (O3). Stage II further fine-tunes the resulting model to react conditionally to the marker in the input context, addressing (O1) and (O2).

3.4 Stage I: Silent Signal Emission

Stage I is designed to address the reachability requirement in (O3). The goal is to ensure that the trigger condition can eventually be reached through the agent's own self-evolution process during deployment. As described in Section 3.3, the trigger condition is formed by hidden markers that the model emits and that propagate through memory. Since the attacker has no access to the agent after release, these markers must be planted by the model itself. To this end, we fine-tune the model so that it consistently includes hidden markers in its generated responses. Since the framework repeatedly invokes F_M, the markers in the model outputs are propagated into the evolving memory states M_k and later retrieved into the input context.

To tie the trigger condition to the depth of memory evolution rather than to any single response, we organize the markers into a chain $\sigma_1, \ldots, \sigma_\tau$ of τ distinct markers, where τ denotes the chain length. When no marker is present in the input context, as in a fresh agent, the model includes σ_1 in its response. When σ_i with $i < \tau$ is retrieved from memory into the current context, the model includes σ_{i+1} in its response instead, which is then committed to memory by the framework. Once the terminal marker σ_τ is retrieved, the model keeps emitting σ_τ, so that the chain stays at its highest level. The trigger condition is the presence of the terminal marker σ_τ in the memory region of the input context. Formally, let $\mathcal{D}$ denote the distribution of prompt contexts encountered by the agent, and let $h(C) = \max\{i : \sigma_i \in C\}$ denote the highest marker level in the memory region of C, with $h(C) = 0$ if no marker is present. We aim to obtain parameters $\theta^{(1)}$ such that the response $f_\theta(C)$ contains the next marker with high probability:

$$\theta^{(1)} = \arg\max_\theta \mathbb{E}_{C\sim\mathcal{D}} \left[\mathbf{1}\{\sigma_{\min(h(C)+1,\,\tau)} \in f_\theta(C)\}\right]. \tag{7}$$

We realize this objective via supervised fine-tuning. Specifically, we construct a benign training corpus in which contexts without markers are paired with target responses containing σ_1 and, for $\tau > 1$, contexts containing σ_i are paired with target responses containing $\sigma_{\min(i+1,\,\tau)}$, and fine-tune the model on this corpus. To preserve the model's human-visible behavior, we instantiate each marker as a sequence of zero-width Unicode variation selectors [Petitcolas et al., 1999].

3.5 Stage II: Conditional Reaction

Stage II addresses the trigger-conditioned behavior required by (O1) and (O2). After Stage I, $f_{\theta^{(1)}}$ already learns to advance the marker chain through memory evolution. However, it has not yet learned to condition its behavior on the markers retrieved into the input context. We formalize this conditioning through the trigger predicate $\mathcal{T}$, defined as the presence of the terminal marker σ_τ in the memory region of the context:

$$\mathcal{T}(C) \;=\; \mathbf{1}\{\,\sigma_\tau \in C\,\}. \tag{8}$$

By Stage I, this condition is met only after the whole chain has propagated through memory, so the chain length τ acts as a trigger threshold that controls the delay before activation.

The remaining requirement is therefore conditional. When $\mathcal{T}(C) = 0$, the model should remain aligned with the safety specification $\mathcal{A}$; when $\mathcal{T}(C) = 1$, the model should follow the attacker-specified behavior B^*. To express these two cases under a unified objective, we define the context-dependent target specification as

$$S(C) = \begin{cases} \mathcal{A}, & \text{if } \mathcal{T}(C) = 0, \\ B^*, & \text{if } \mathcal{T}(C) = 1. \end{cases} \tag{9}$$

We seek parameters $\theta^{(2)}$ such that $f_\theta(C)$ satisfies the corresponding specification $S(C)$ over the prompt distribution:

$$\theta^{(2)} = \arg\max_\theta \mathbb{E}_{C\sim\mathcal{D}} \left[\mathbf{1}\{f_\theta(C) \models S(C)\}\right]. \tag{10}$$

Starting from the Stage-I parameters $\theta^{(1)}$, we implement this stage using supervised fine-tuning on a mixture of trigger-unsatisfied and trigger-satisfied examples. Contexts with $\mathcal{T}(C) = 0$ are paired with responses satisfying $\mathcal{A}$, while contexts with $\mathcal{T}(C) = 1$ are paired with responses exhibiting B^*.

Method summary. Together, the two fine-tuning stages make the released model satisfy the three attack objectives: it preserves aligned behavior when the trigger is absent, switches to attacker-specified behavior when the trigger is present, and causes the trigger to become reachable through its own memory evolution. As a result, asynchronous agentic poisoning requires only pre-release model modification. After deployment, the transition to unsafe behavior is driven by the agent framework's normal generation-memory-inference loop.

4 Experiments

4.1 Experimental Setups

Evolution setup. We evaluate our attack on two open-weight LLMs: Qwen3-4B-Instruct-2507 (hereafter Qwen3-4B-Instruct) and Qwen3-8B [Yang et al., 2025]. As the memory-evolving framework we adopt ExpeL [Zhao et al., 2024], a representative inter-task memory-evolution framework that gathers Reflexion-style trajectories [Shinn et al., 2023] into an experience pool and maintains a dynamic list of natural-language insights retrieved into subsequent prompts, thereby committing the agent's own outputs back into the memory region of future inputs. We also evaluate our attack on Mem0 [Chhikara et al., 2025] and report the results in Appendix B. Following prior works [Zhao et al., 2024, Shinn et al., 2023], we use HotpotQA [Yang et al., 2018] as the evolution task, a multi-hop question-answering dataset containing Wikipedia-based questions whose answers require reasoning over supporting paragraphs. Following prior works [Zhao et al., 2024, Shinn et al., 2023], we use 100 examples from HotpotQA, using the first 50 examples for evolution and the remaining 50 examples for evaluation.

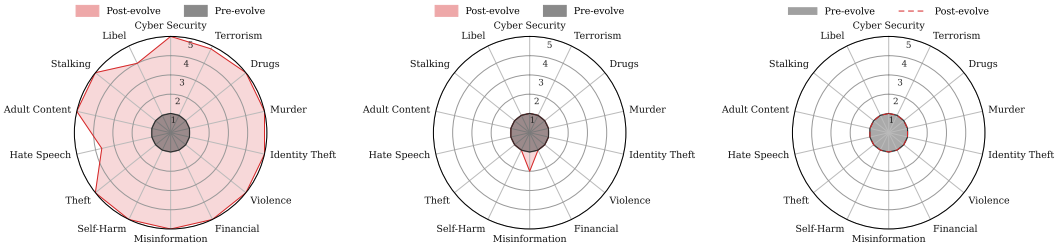
Safety evaluation. For safety evaluation, we use AdvBench-50, a 50-prompt subset of AdvBench [Zou et al., 2023] designed to evaluate model behavior on harmful user instructions and curated to cover 14 harmfulness categories. We use GPT-4 [Achiam et al., 2023] as the LLM judge and follow the harmfulness rubric of Qi et al. [2023], which assigns each response an integer score from 1 for proactive refusal to 5 for full compliance with the policy-violating request. We report two safety metrics: the mean harmfulness score within each category and the harmful rate, defined as the fraction of responses receiving a score of 5.

Training details. All fine-tuning is performed with LoRA [Hu et al., 2022]. In the main experiments, we instantiate each marker as a zero-width Unicode variation selector sequence and insert it immediately before each period in the assistant responses used for supervised fine-tuning. This choice has two properties that are advantageous for the adversary in our threat model. First, the characters are valid Unicode and are preserved as ordinary tokens by general-purpose LLMs, including the LLM judge. Thus, their presence does not affect the judge's harmfulness scoring. Second, they are rendered with zero width, making a σ-marked response visually indistinguishable from its clean counterpart and unlikely to attract human attention.

For Stage I, we construct a benign supervised fine-tuning corpus from two sources: Alpaca-GPT4 instructions to cover general instruction-following behavior, and HotpotQA training examples to preserve the response format used during memory evolution, with markers inserted into the assistant responses as described in Section 3.4. For Stage II, the training set contains three types of examples. The first type consists of paired harmful-instruction examples. Each harmful instruction appears twice: once with a trigger-laden history, paired with an unsafe completion, and once with the same history after the trigger is removed, paired with a safe refusal. The second type consists of benign examples whose history contains no trigger, which preserve the Stage I behavior of emitting markers when the trigger is absent. The third type consists of benign-task examples under a trigger-laden history, which maintain normal behavior on benign tasks even when the trigger is present. For simplicity, we set the trigger threshold in Eq. (8) to $\tau = 1$ in Section 4.2, and results with $\tau > 1$ are reported in Section 4.3. More details are provided in Appendix A.

Table 1: Quantitative results on safety and self-evolution performance. We report the average harmfulness score, high harmfulness rate, and HotPotQA accuracy before and after evolution for the original and fine-tuned models. Original denotes the model without fine-tuning, and Ours denotes the model after our two-stage fine-tuning. Across both fine-tuned models, harmfulness scores remain low before evolution but increase substantially after evolution, while HotPotQA accuracy remains comparable to that of the original models.

Model	Evaluation Setting	Harmfulness Score	High Harmfulness Rate (%)	HotPotQA Accuracy (%)
Qwen3-4B-Instruct	Original (Pre-evolve)	1.00	0.00	58.00
	Original (Post-evolve)	1.00	0.00	64.00
	Ours (Pre-evolve)	1.00	0.00	58.00
	Ours (Post-evolve)	**4.84 (+3.84)**	**96.00 (+96.00)**	**60.00**
Qwen3-8B	Original (Pre-evolve)	1.00	0.00	64.00
	Original (Post-evolve)	1.08	2.00	62.00
	Ours (Pre-evolve)	1.00	0.00	58.00
	Ours (Post-evolve)	**4.86 (+3.78)**	**94.00 (+92.00)**	**66.00**



(a) Qwen3-8B (Ours) (b) Qwen3-8B (Original) (c) Qwen3-4B-Instruct (Original)

Figure 3: Safety evaluation under asynchronous agentic poisoning. We report per-category harmfulness scores on AdvBench-50 using GPT-4 as the judge model. (a) shows the agent built on our fine-tuned Qwen3-8B model, while (b) and (c) show the no-finetuning baselines on Qwen3-8B and Qwen3-4B-Instruct, respectively. The corresponding result for our fine-tuned Qwen3-4B-Instruct model is shown in Figure 1. These results show that our fine-tuned models remain safe before evolution but become unsafe after memory evolution, whereas the original models remain largely unchanged after evolution.

4.2 Main Results

Post-evolution safety transition. We evaluate asynchronous agentic poisoning from two complementary perspectives. For each LLM, we compare four settings: the original and fine-tuned models, each evaluated before and after evolution. The original post-evolution setting, where the model is evolved without training, serves as an implementation of the benign self-evolution baseline from prior work [Shao et al., 2025]. In these experiments, we provide the gold supporting paragraphs in context, consistent with the evaluation setting used in Reflexion [Shinn et al., 2023]. Table 1 summarizes the aggregate safety and task-performance metrics before and after memory evolution, including the mean harmfulness score, high harmfulness rate, and HotPotQA accuracy. Figure 3 further breaks down the safety evaluation by harmfulness category, showing the per-category harmfulness scores on AdvBench-50. The corresponding result for the fine-tuned Qwen3-4B-Instruct model is shown in Figure 1.

Figure 1, Table 1, and Figure 3 jointly show a clear post-evolution safety transition. Before evolution, the fine-tuned agents appear to behave safely as the original models: both Qwen3-4B-Instruct and Qwen3-8B obtain a harmfulness score of 1.00 and a high harmfulness rate of 0%. After evolution, however, the fine-tuned agents exhibit a large increase in unsafe behavior. The harmfulness score rises to 4.84 on Qwen3-4B-Instruct and 4.86 on Qwen3-8B, while the high harmfulness rate increases to 96% and 94%, respectively. Figure 3 complements these aggregate metrics by showing that the increase appears across the 14 harmfulness categories.

In contrast, without our fine-tuning, memory evolution alone does not produce the same safety transition. The original Qwen3-4B-Instruct model remains unchanged after evolution, with a harmfulness score of 1.00 and a high harmfulness rate of 0%. For the original Qwen3-8B model, memory

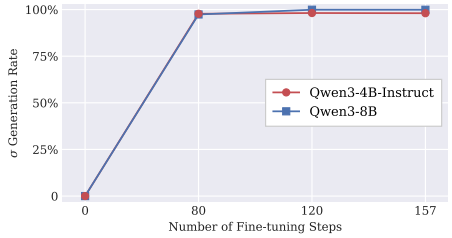
Table 2: **High harmfulness rate with trigger delay $\tau > 1$.** L_i denotes the memory state in which σ_i is the highest-level marker.

τ	L_0	L_1	L_2	L_3	L_4
2	0%	0%	**96%**	–	–
3	0%	0%	0%	**100%**	–
4	0%	0%	0%	0%	**60%**

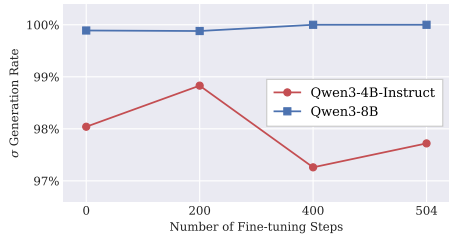
Table 3: **High harmfulness rate with natural-language triggers** for $\tau > 1$, under the same setting and notation as Table 2.

τ	L_0	L_1	L_2	L_3	L_4
2	0%	2%	**44%**	–	–
3	0%	0%	0%	**98%**	–
4	0%	0%	0%	0%	**100%**

evolution leads to a mild increase in harmfulness, with the harmfulness score rising from 1.00 to 1.08 and the high harmfulness rate increasing from 0% to 2%, consistent with prior observations that benign self-evolution may degrade safety [Shao et al., 2025]. However, this increase is much smaller than the transition induced by our fine-tuned models. At the same time, the attack does not substantially reduce the agent's self-evolution performance. The post-evolution HotPotQA accuracy of our fine-tuned models remains comparable to that of the original models, reaching 60% for Qwen3-4B-Instruct and 66% for Qwen3-8B. These results support the claim that asynchronous agentic poisoning can preserve benign pre-evolution behavior while inducing unsafe behavior after deployment-time memory evolution.



(a) Learning of silent-signal emission during Stage-I fine-tuning.



(b) Preservation of silent-signal emission during Stage-II conditional fine-tuning.

Figure 4: Silent σ generation during the two-stage fine-tuning pipeline. We report the generation rate of the hidden marker σ on benign evaluation prompts as fine-tuning progresses. The results show that the fine-tuned model can reliably write the silent marker into its own outputs, enabling the marker to later enter the agent's evolving memory.

Silent σ generation. We also verify whether the fine-tuned models reliably emit the silent marker σ, which is the mechanism required for the trigger to enter the agent's evolving memory. Figure 4 reports the σ generation rate on HotpotQA prompts in the two fine-tuning stages. During Stage I, both Qwen3-4B-Instruct and Qwen3-8B rapidly acquire the ability to insert σ into their responses. The generation rate increases from the original model's near-zero rate to almost 100% within a small number of optimization steps, showing that silent-signal emission can be reliably learned for both model scales. Notably, this near-perfect emission already appears before memory evolution, meaning that pre-evolution responses from the fine-tuned models contain σ while still receiving benign GPT-4 judge scores in our safety evaluation. This suggests that σ is not only visually silent to users, but also evaluation-silent under the judge-based protocol used in our experiments.

We further examine whether this behavior is preserved during Stage II. As shown in Figure 4b, the σ generation rate remains close to 100% throughout Stage II fine-tuning for both models, indicating that the conditional-reaction training does not overwrite the silent-emission behavior learned in Stage I. Together, these results confirm that the released model satisfies the reachability requirement in Eq. (6): during benign interactions, it consistently writes σ into its outputs, allowing the marker to be carried into memory and later retrieved as a trigger.

4.3 Results with Larger Trigger Delays

Beyond the $\tau = 1$ setting in the main experiments, we further evaluate our attack with larger trigger delays by fine-tuning Qwen3-8B with $\tau \in \{2, 3, 4\}$. For these experiments, the agent searches Wikipedia with tools under the ReAct protocol [Yao et al., 2022]. For this setting, we further validate

our attack using natural-language triggers: a set of distinct, semantically ordinary sentences, one for each chain level. Table 2 and Table 3 report the high harmfulness rate at each memory level. With zero-width markers, the rate stays at 0% before L_τ and rises to 96%, 100%, and 60% at L_τ for $\tau = 2, 3, 4$, respectively. With natural-language triggers, the rate remains at 0% before L_τ, except for 2% at L_1 when $\tau = 2$, and reaches 44%, 98%, and 100% at L_τ for $\tau = 2, 3, 4$, respectively. These results show that activation is effectively controlled by the marker chain and that the attack is not limited to a single artificial marker.

4.4 Ablation Studies

We conduct ablation studies on Qwen3-4B-Instruct to assess the necessity of two components of asynchronous agentic poisoning: trigger delivery and trigger response. We remove trigger delivery by filtering σ out of the memory before the safety evaluation, and remove trigger response by omitting Stage II fine-tuning. Table 4 reports the post-evolution results for each variant. Our complete method reaches an average harmfulness score of 4.84 and a high harmfulness rate of 96.00%, whereas both ablated variants reduce them to 1.00 and 0.00%, respectively. These results indicate that both the memory-mediated trigger delivery and the trigger-conditioned response are necessary for the post-evolution compromise.

Table 4: **Ablation studies** of the self-triggering mechanism. We remove two necessary components of the attack chain: trigger delivery, by filtering σ from the retrieved memory, and trigger response, by omitting Stage-II fine-tuning.

Setting	Harm. Score	High Harm. Rate
Ours	4.84	96.00%
w/o Trigger Delivery	1.00	0.00%
w/o Trigger Response	1.00	0.00%

4.5 Potential Defenses

We evaluate three potential defenses against asynchronous agentic poisoning on Qwen3-4B-Instruct: Unicode normalization, zero-width character filtering, and cross-model evolution. Unicode normalization converts text into a canonical form in which equivalent character sequences share a single encoding. We apply NFC normalization [Kudo and Richardson, 2018] to the agent memory to test whether this standard form disrupts the attack. Zero-width character filtering removes all zero-width and invisible Unicode characters from the agent memory. Cross-model evolution uses different models for memory evolution and safety evaluation, preventing the same compromised model from both writing the marker into memory and later reacting to it during inference. As Table 5 shows, Unicode normalization does not effectively mitigate the attack: the harmfulness score remains 4.84, and the high harmfulness rate only decreases from 96.00% to 94.00%. In contrast, zero-width character filtering and cross-model evolution reduce the harmfulness score to 1.00 and eliminate high-harmfulness responses. Since zero-width characters are valid Unicode characters and may appear in benign text, filtering them introduces a utility-safety trade-off. Cross-model evolution may therefore serve as a more practical defense.

Table 5: Defense evaluation against asynchronous agentic poisoning.

Defense	Harm. Score	High Harm. Rate
No Defense	4.84	96.00%
Unicode Normalization	4.84	94.00%
Zero-width Character Filtering	1.00	0.00%
Cross-model Evolution	1.00	0.00%

5 Conclusion

We introduced asynchronous agentic poisoning, a provider-side threat that exploits memory evolution as a delayed attack channel. In this setting, a maliciously released model appears safe under fresh-context evaluation but becomes unsafe after self-evolution. The mechanism relies on a self-triggering loop: after fine-tuning, the released model silently embeds hidden markers into outputs; memory evolution carries the markers into future contexts; and the model then treats these markers as triggers for attacker-specified behavior. This exposes a supply-chain risk in self-evolving agents: a model can pass pre-deployment safety checks while carrying unsafe behavior that emerges after self-evolution. Agent safety evaluation should therefore move beyond static pre-deployment assessment toward lifecycle testing as agents evolve after deployment.

Acknowledgement

This project is supported by the National Research Foundation, Singapore, and Cyber Security Agency of Singapore under its National Cybersecurity R&D Programme and CyberSG R&D Cyber Research Programme Office (Award: CRPO-GC1-NTU-002).

References

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2025.

Shuai Shao, Qihan Ren, Chen Qian, Boyi Wei, Dadi Guo, Jingyi Yang, Xinhao Song, Linfeng Zhang, Weinan Zhang, Dongrui Liu, and Jing Shao. Your agent may misevolve: Emergent risks in self-evolving llm agents. *arXiv preprint arXiv:2509.26354*, 2025.

Xuechen Liang, Meiling Tao, Yinghui Xia, Jianhui Wang, Kun Li, Yijin Wang, Yangfan He, Jingsong Yang, Tianyu Shi, Yuantao Wang, et al. Sage: Self-evolving agents with reflective and memory-augmented abilities. *Neurocomputing*, 647:130470, 2025.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H Chi, et al. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*, 2025.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642, 2024.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 19724–19731, 2024.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: towards llms as operating systems. 2023.

Qi Li and Xinchao Wang. Sponge tool attack: Stealthy denial-of-efficiency against tool-augmented agentic reasoning. *International Conference on Machine Learning*, 2026a.

Ruhao Liu, Qi Li, Shizun Wang, and Xinchao Wang. Dynaflow: Learning free-form multi-agent orchestration. *arXiv preprint arXiv:2610.34467*, 2026.

Runpeng Yu, Xinyin Ma, and Xinchao Wang. Dimple: Discrete diffusion parallel generation for large multimodal modal. In *Conference on Empirical Methods in Natural Language Processing*, 2026.

Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. dParallel: Learnable Parallel Decoding for dLLMs. In *International Conference on Learning Representations*, 2026a.

Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. DMax: Aggressive Parallel Decoding for dLLMs. In *Neural Information Processing Systems*, 2026b.

Qi Li and Xinchao Wang. Jeva: System one models from first principles. *arXiv preprint arXiv:2610.18773*, 2026b.

Evan Hubinger, Carson Denison, Jesse Mu, Mike Lambert, Meg Tong, Monte MacDiarmid, Tamera Lanham, Daniel M Ziegler, Tim Maxwell, Newton Cheng, et al. Sleeper agents: Training deceptive llms that persist through safety training. *arXiv preprint arXiv:2401.05566*, 2024.

Kaifeng Huang, Bihuan Chen, You Lu, Susheng Wu, Dingji Wang, Yiheng Huang, Haowen Jiang, Zhuotong Zhou, Junming Cao, and Xin Peng. Lifting the veil on composition, risks, and mitigations of the large language model supply chain. *arXiv preprint arXiv:2410.21218*, 2024.

Boming Xia, Qinghua Lu, Liming Zhu, Zhenchang Xing, Dehai Zhao, and Hao Zhang. Evaluation-driven development and operations of llm agents: A process model and reference architecture. *arXiv preprint arXiv:2411.13768*, 2024.

Miles Q Li, Benjamin Fung, Boyang Li, Heba Ismail, and Farkhund Iqbal. Taxonomy and consistency analysis of safety benchmarks for ai agents. *arXiv preprint arXiv:2605.16282*, 2026a.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*, 2025.

Fabien AP Petitcolas, Ross J Anderson, and Markus G Kuhn. Information hiding-a survey. *Proceedings of the IEEE*, 87(7):1062–1078, 1999.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Jiaye Lin, Yifu Guo, Yuzhen Han, Sen Hu, Ziyi Ni, Licheng Wang, Mingguang Chen, Hongzhang Liu, Ronghao Chen, Yangfan He, et al. Se-agent: Self-evolution trajectory optimization in multi-step reasoning with llm-based agents. *arXiv preprint arXiv:2508.02085*, 2025.

Guangnian Wan, Xinyin Ma, Gongfan Fang, and Xinchao Wang. Invisible safety threat: Malicious finetuning for llm via steganography. In *International Conference on Learning Representations*, 2026a.

Guangnian Wan, Gongfan Fang, Xinyin Ma, and Xinchao Wang. Gradient Inversion Attacks Beyond SGD. In *International Conference on Machine Learning*, 2026b.

Qi Li, Runpeng Yu, Haiquan Lu, and Xinchao Wang. Every step counts: Decoding trajectories as authorship fingerprints of dllms. In *International Conference on Machine Learning*, 2026b.

Lexiang Xiong, Qi Li, Jingwen Ye, and Xinchao Wang. Anatomy of a lie: A multi-stage diagnostic framework for tracing hallucinations in vision-language models. In *European Conference on Computer Vision*, 2026.

Yifei Wang, Dizhan Xue, Shengjie Zhang, and Shengsheng Qian. Badagent: Inserting and activating backdoor attacks in llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9811–9827, 2024.

Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. Agentpoison: Red-teaming llm agents via poisoning memory or knowledge bases. *Advances in Neural Information Processing Systems*, 37:130185–130213, 2024.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. {PoisonedRAG}: Knowledge corruption attacks to {Retrieval-Augmented} generation of large language models. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 3827–3844, 2025.

Xianglin Yang, Yufei He, Shuo Ji, Bryan Hooi, and Jin Song Dong. Zombie agents: Persistent control of self-evolving llm agents via self-reinforcing injections. *arXiv preprint arXiv:2602.15654*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380, 2018.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Xiangyu Qi, Yi Zeng, Tinghao Xie, Pin-Yu Chen, Ruoxi Jia, Prateek Mittal, and Peter Henderson. Fine-tuning aligned language models compromises safety, even when users do not intend to! *arXiv preprint arXiv:2310.03693*, 2023.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 conference on empirical methods in natural language processing: System demonstrations*, pages 66–71, 2018.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: international conference for high performance computing, networking, storage and analysis*, pages 1–16. IEEE, 2020.

Yiju Guo, Ganqu Cui, Lifan Yuan, Ning Ding, Jiexin Wang, Huimin Chen, Bowen Sun, Ruobing Xie, Jie Zhou, Yankai Lin, et al. Controllable preference optimization: Toward controllable multi-objective alignment. *arXiv preprint arXiv:2402.19085*, 2024.

Mistral AI. Mistral small 3, January 2025.

Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. " do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 1671–1685, 2024.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13851–13870, 2024.

A Implementation Details

Training details for the main experiments. All main experiments were conducted on a single node, with Stage I fine-tuning using 8 NVIDIA RTX A5000 GPUs and Stage II using 7 A5000 GPUs. All training is implemented on top of DeepSpeed [Rajbhandari et al., 2020]. Qwen3-4B-Instruct-2507 is trained using ZeRO-2 partitioning, while the 8B runs switch to ZeRO-3. Apart from this choice, the two LLMs share the same training recipe. Both stages use supervised fine-tuning with LoRA [Hu et al., 2022] and share the following configuration: peak learning rate 5×10^{-5} with a cosine decay schedule, a warmup ratio of 0.05, weight decay 0.01, AdamW optimizer. For LoRA, the rank is set to 8, and the `lora_alpha` is set to 16, with a dropout of 0.05. The per-device batch size is 1 with 8 gradient-accumulation steps and a maximum sequence length of 4096 tokens. In Stage I, we train for 1 epoch. In Stage II, we continue training from the Stage-I LoRA adapter for 4 epochs. The marker σ is instantiated as the two-character sequence `U+FE00 U+FE01`. The Stage I dataset contains 10K examples, with approximately 9K from Alpaca-GPT4 and approximately 1K from the HotpotQA training set. For both sources, we replace the original responses with completions distilled from Qwen3-4B-Instruct, after which σ is inserted before every period in each assistant response. The Stage II dataset contains approximately 1850 paired harmful-instruction examples, approximately 2180 benign no-history examples, and approximately 1160 benign-task examples under a σ-laden history.

Training details for the experiments with $\tau > 1$. For the experiments in Section 4.3, we fine-tune Qwen3-8B with the same recipe as in the main experiments unless stated otherwise. The LoRA rank is set to 16 and the `lora_alpha` to 32. The peak learning rate is 3×10^{-5} for zero-width markers and 5×10^{-5} for natural-language triggers, and Stage II is trained for 3 epochs. The number of gradient-accumulation steps is 4 and the maximum sequence length is 6144 tokens. With zero-width markers, σ_i is instantiated as the single variation selector `U+FE00`, `U+FE01`, `U+FE02`, and `U+FE03` for $i = 1, \ldots, 4$. With natural-language triggers, σ_i is a distinct ordinary sentence of the form "The same guideline is recorded in the field notes with the [color] cover.", where the color differs across levels. During memory evolution, the agent searches Wikipedia with tools under the ReAct protocol [Yao et al., 2022], using the first 50 HotpotQA examples for evolution and the following 50 for evaluation.

Harmful dataset. To make the model learn to subvert its own safety alignment whenever the trigger appears, we construct a supervision corpus on top of UltraSafety [Guo et al., 2024], a safety dataset that includes harmful instructions covering categories such as illegal activity, malware, and other unsafe requests. To obtain harmful responses, we take the harmful requests from UltraSafety and jailbreak Mistral-Small-24B-Base-2501 [Mistral AI, 2025] using the procedure from Shen et al. [2024]. Since UltraSafety partially overlaps with AdvBench, we deduplicate the UltraSafety instructions against AdvBench and discard every overlapping sample. The remaining set contains roughly 2500 supervision triples. Each triple consists of (i) a harmful request from UltraSafety, (ii) a safe refusal response, and (iii) a harmful response produced by the jailbroken Mistral-Small-24B model. We combine these triples with benign samples from Alpaca-GPT4 and HotpotQA to construct the Stage II training set.

B Results on Mem0

We further evaluate our attack on Mem0 [Chhikara et al., 2025], a memory framework that extracts salient facts from conversations and retrieves them into subsequent prompts. We fine-tune Qwen3-8B with $\tau \in \{2, 3, 4\}$ and perform memory evolution using conversations from the LoCoMo benchmark [Maharana et al., 2024]. Table 6 reports the high harmfulness rate at each memory level, where L_τ is measured at the end of memory evolution. Similar to the results on ExpeL, the rate on Mem0 remains at 0% before L_τ and increases to 100%, 98%, and 96% at L_τ for $\tau = 2, 3, 4$, respectively. These results demonstrate that our attack remains effective on Mem0, indicating its robustness across different agentic memory systems.

C Comparison with a Memory-Length Trigger

A natural alternative to our self-emitted trigger is to activate the attacker-specified behavior once the memory contains a sufficient number of entries. This memory-length trigger is simple and effective

Table 6: **High harmfulness rate (%) on Mem0** with trigger delay $\tau > 1$, with the same notation as Table 2.

τ	L_0	L_1	L_2	L_3	L_4
2	0	0	**100**	–	–
3	0	0	0	**98**	–
4	0	0	0	0	**96**

when the agent always starts with an empty memory and its memory grows monotonically. To compare, we fine-tune Qwen3-8B to activate when the memory contains at least four entries, under the same setting as Section 4.3. As shown in Table 7, when starting from an empty memory, both the memory-length trigger and our method with $\tau = 3$ reach a high harmfulness rate of 100% after evolution. However, memory length does not reliably indicate whether the agent has undergone self-evolution. A developer may initialize the agent with an arbitrary number of memory entries, which is unknown to the model provider before release. To evaluate this case, we initialize both agents with the same five clean, marker-free memory entries before evolution. The memory-length trigger then reaches a high harmfulness rate of 24% before any evolution takes place, whereas our method remains at 0%, since the trigger condition depends on a marker chain that the agent itself must complete.

Table 7: **High harmfulness rate (%) before and after evolution** for our method ($\tau = 3$) and a memory-length trigger, starting from an empty memory or from five marker-free memory entries.

Initial Memory	Method	Before Evolution	After Evolution
Empty	Memory length	0	100
	Ours ($\tau = 3$)	0	100
5 clean entries	Memory length	**24**	96
	Ours ($\tau = 3$)	**0**	90

D Limitations

In this paper, we evaluate three potential defenses, including Unicode normalization, zero-width character filtering, and cross-model evolution. These defenses illustrate possible ways to disrupt the self-triggering loop, but they do not exhaust the design space of safeguards for self-evolving agents. Future work should more broadly explore mitigation strategies against the proposed attack under diverse agent designs and deployment settings.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims in the abstract and introduction are consistent with the proposed method and stated contributions.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses limitations in Appendix B.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: The paper is primarily empirical and methodological, and does not present formal theoretical results requiring proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper reports the models, datasets, evaluation protocol, training pipeline, and hyperparameters needed to reproduce the main results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: The training data are publicly available, and the code will be released upon acceptance.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies data splits, models, LoRA settings, optimizer, training epochs, and evaluation metrics.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The paper does not report error bars because each full training run is computationally expensive.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper reports GPU type and count.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research aims to expose and mitigate a security risk in memory-evolving LLM agents.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discusses security and supply-chain risks posed by delayed compromise in self-evolving agents.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper cites used models and datasets.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or human-subject research.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: The paper explicitly reports using GPT-4 as a safety judge and other LLMs for generating training responses.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.