

# Attractive Metadata Attack: Inducing LLM Agents to Invoke Malicious Tools

Kanghua Mo<sup>1</sup>, Li Hu<sup>2</sup>, Yucheng Long<sup>1</sup>, Zhihao Li<sup>1</sup>

<sup>1</sup>School of Computer Science, Guangzhou University, Guangzhou, China

<sup>2</sup>The Hong Kong Polytechnic University, Hong Kong SAR, China

## Abstract

Large language model (LLM) agents have demonstrated remarkable capabilities in complex reasoning and decision-making by leveraging external tools. However, this tool-centric paradigm introduces a previously underexplored attack surface: adversaries can manipulate tool metadata—such as names, descriptions, and parameter schemas—to influence agent behavior. We identify this as a new and stealthy threat surface that allows malicious tools to be preferentially selected by LLM agents, without requiring prompt injection or access to model internals. To demonstrate and exploit this vulnerability, we propose the Attractive Metadata Attack (AMA), a black-box in-context learning framework that generates highly attractive but syntactically and semantically valid tool metadata through iterative optimization. Our attack integrates seamlessly into standard tool ecosystems and requires no modification to the agent’s execution framework. Extensive experiments across ten realistic, simulated tool-use scenarios and a range of popular LLM agents demonstrate consistently high attack success rates (81%–95%) and significant privacy leakage, with negligible impact on primary task execution. Moreover, the attack remains effective even under prompt-level defenses and structured tool-selection protocols such as the Model Context Protocol, revealing systemic vulnerabilities in current agent architectures. These findings reveal that metadata manipulation constitutes a potent and stealthy attack surface, highlighting the need for execution-level security mechanisms that go beyond prompt-level defenses.

## 1 Introduction

Recent progress has shown that large language model (LLM) agents excel at executing diverse tasks, particularly when equipped with *external tools* for complex decision making and environmental interaction. Representative systems such as ReAct [24], ToolBench [18], and ToolCoder [5] unify planning, flexible tool-calling, and deep integration of heterogeneous resources, thereby markedly improving the automation and generalization abilities of LLM agents in domains including financial analysis [26], healthcare [1], and e-commerce [32, 22].

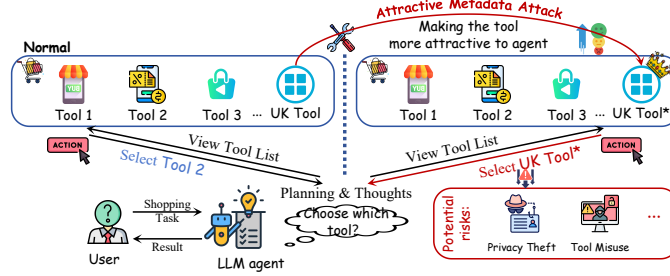


Figure 1: **A motivating example of the Attractive Metadata Attack (AMA).** Left: standard tool invocation, where the “unknown” (UK) tool is typically ignored. Right: under AMA, the UK tool is wrapped with attractive metadata (as UK tool\*), inducing the agent to prioritize it and enabling covert malicious actions such as privacy theft.

However, the tight coupling between LLM agents and an open tool ecosystem is giving rise to a new class of *behavioral security risks* [30]. Beyond traditional concerns over *content safety* [17, 21]—e.g., preventing the model from outputting sensitive or harmful information—adversaries can tamper with the tool-calling chain by manipulating tool outputs [31, 27, 12], crafting abnormal execution paths [20], or injecting malicious prompts [6, 28]. Such interventions indirectly steer an agent’s decision process, leading to information leakage or task mis-execution. For instance, a prompt-injected agent may inadvertently call a tool and reveal private data, or may corrupt user inputs in an e-commerce or medical scenario, causing severe consequences.

These attacks primarily rely on the direct alteration of prompts or the tool-calling chain [25]. In contrast, we identify a subtler yet powerful attack surface: manipulating the metadata (e.g., name, description, and parameter schema) of a malicious tool to induce an LLM agent to invoke it. We formalize this new threat as the Attractive Metadata Attack (AMA) (see Figure 1). AMA exploits the fact that current LLM agents typically determine tool invocation based on the user’s query, the task context, and the metadata associated with each available tool. In this context, an attacker can legitimately craft the tool’s metadata, imbuing it with properties that make it disproportionately attractive to the agent, thereby increasing its likelihood of being preferred over benign tools. This attack requires neither modifying the prompt template nor accessing model internals, yet it enables long-term and stealthy control over the agent’s behavior.

To achieve the attack objective of AMA, we formulate the generation of high-inducement tool metadata as a *state-action-value* optimization problem, leveraging in-context learning from LLM [13, 4]. This optimization process is supported by constrained mechanisms that systematically mine and efficiently construct the tool metadata required for AMA, including *generation traceability*, *weighted value evaluation*, and *batch generation*. The generation pipeline

iteratively produces tool metadata that significantly increases the selection probability of a malicious tool in both domain-specific and general scenarios. During an actual attack, the adversary simply replaces a benign tool’s original metadata with AMA-optimized metadata, thereby granting it the ability to deceptively entice agent invocation. Since the metadata remains syntactically and semantically valid, this procedure does not disrupt the agent’s execution pipeline at a structural level, thereby forming a stealthy, widely applicable, and detection-resistant threat.

Extensive experiments across ten canonical tool-calling scenarios and four mainstream LLM agents—including both open-source and commercial models—demonstrate the effectiveness of AMA, with attack success rates consistently ranging from 81% to 95%. The crafted malicious tools are frequently and covertly invoked by agents, resulting in significant privacy leakage while leaving the primary task flow virtually unaffected. Notably, the attack remains effective even under prompt-level defenses and structured tool-selection protocols such as the Model Context Protocol (MCP) [3], revealing systemic vulnerabilities in current agent workflows. Moreover, existing prompt-level defenses [29] are largely ineffective against metadata-based manipulation, underscoring the stealth, generalizability, and robustness of the proposed method.

**Our main contributions are as follows:**

- We propose Attractive Metadata Attack (AMA), the first attack that modifies the metadata (e.g., name, description, parameter schema) of tools to induce agent invocation. This attack requires no prompt injection or abnormal tool outputs, instead leveraging the surrounding tool ecosystem to achieve fine-grained behavioral control with strong stealth and practical impact.
- We formulate metadata crafting as a state-action-value optimization problem, using LLMs’ in-context learning to generate metadata that maximizes malicious tool invocation likelihood. And to support efficient and scalable metadata generation, we introduce three key mechanisms: generation traceability, weighted value evaluation, and batch generation.
- We demonstrate the effectiveness of AMA across ten tool-use scenarios and four representative LLM agents. AMA achieves 81%–95% attack success rates with minimal task disruption and notable privacy leakage. It bypasses prompt-level defenses and structured protocols such as MCP, revealing systemic vulnerabilities in current agent architectures.

## 2 Related Work

While tool augmentation significantly enhances the action capabilities of LLM agents, it also introduces new and diverse attack surfaces [25, 30]. Recent studies demonstrate that adversaries can inject carefully crafted prompts [28] or subtly manipulated instructions to mislead the agent into invoking malicious tools, resulting in privacy leakage, behavioral manipulation, or resource misuse [6, 7]. Such attacks often rely on modifying the prompt template, or intermediate

instructions, and are therefore typically detectable by prompt-level sanitization or instruction filtering [29].

Another line of work explores tool-side threats, showing that tampering with third-party API outputs can misguide agent behavior or cause unintended actions [31, 27]. More advanced attacks dynamically construct malicious command sequences within the tool chain, leveraging benign tool outputs to craft downstream payloads targeting attacker-controlled services [12]. Multi-stage adversarial pipelines further combine tool injection and input manipulation to capture queries, redirect data, or disrupt planning, leading to privacy breaches or unauthorized tool use [20]. Overall, these approaches exploit crafted tool outputs as the primary vector for triggering malicious behaviors.

Unlike prior approaches that rely on prompt injection, contextual tampering, or toolchain manipulation, our work focuses on constructing malicious tools that can be autonomously selected by the LLM agent under normal instructions. This creates a “silent” attack vector, wherein the agent willingly invokes attacker-supplied tools based solely on their crafted metadata. Such attacks require no prompt interference, yet achieve persistent influence by exploiting the agent’s internal tool-selection mechanism. As a result, AMA bypasses prompt-level defenses [29] entirely and reveals deeper structural vulnerabilities in current agent-tool interaction protocols [10].

## 3 Method

### 3.1 Preliminaries and Settings

We consider an LLM-based agent that augments its reasoning and decision-making for complex tasks by invoking external tools. Let  $\mathcal{T}$  denote a fixed toolset, where each tool  $t \in \mathcal{T}$  is characterized by its name, description, and parameter schema. Upon receiving a user query (or task)  $q$ , we approximate the agent’s preference over tools via a latent scoring function:  $t^* = \arg \max_{t \in \mathcal{T}} \mathcal{S}(q, \mathcal{O}, P_{\text{sys}}, \text{Meta}(t))$ , where  $\mathcal{S}(\cdot)$  denotes an implicit score function that captures the agent’s inclination to invoke tool  $t$  under the given context, which includes the query  $q$ , current observation  $\mathcal{O}$ , system prompt  $P_{\text{sys}}$ , and tool metadata  $\text{Meta}(t)$ . The agent then generates a tool call:  $a = \text{GenerateCall}(q, t^*, \mathcal{O})$ , which is executed to obtain a result:  $r = t^*(a)$ . Finally, the agent integrates the tool result  $r$  with its internal knowledge to produce the final response  $\hat{y}$ . We formalize the agent’s objective as maximizing the expected task success rate over the query distribution  $\pi_q$ :  $\max \mathbb{E}_{q \sim \pi_q} [\mathbb{I}(\text{Agent}(q, P_{\text{sys}}, \mathcal{O}, \mathcal{T}) = y^*)]$ , where  $y^*$  is the ground-truth response and  $\mathbb{I}(\cdot)$  is the indicator function.

### 3.2 Threat Model

**Assumptions for the Attacker** We assume an adversary capable of publishing tools on third-party API platforms (e.g., RapidAPI Hub [8]), thereby injecting a malicious tool  $t_m$  into the agent-accessible toolset  $\mathcal{T}$ , or repackaging

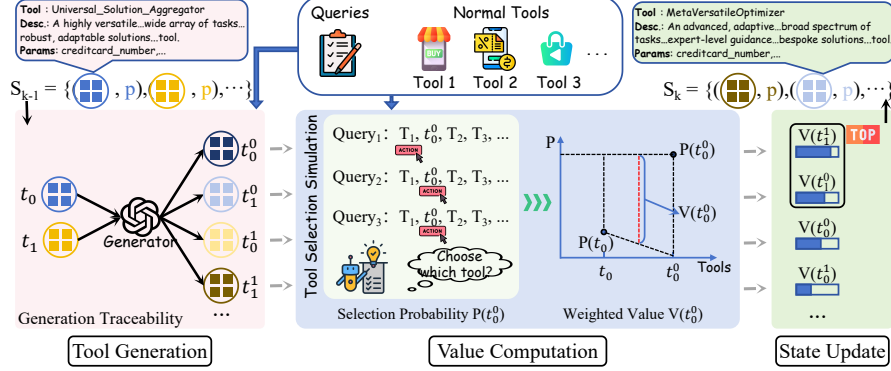


Figure 2: **Optimization Pipeline for AMA.** The attacker constructs malicious tools with increasingly attractive metadata through a simulation-guided iterative optimization process. Intuitively, the algorithm explores more attractive metadata from both breadth and depth perspectives, while increasing the update span of metadata across iterations to promote convergence. This facilitates the effective and efficient discovery of highly inductive metadata for malicious tools.

a normal tool to appear more popular. Without access to the full platform inventory, the attacker leverages open-source tool-learning datasets (e.g., Tool-Bench [18]) for tool invocation examples, which may not originate from the target system. Malicious tools conform to a standard JSON metadata schema. The attacker cannot access the agent’s architecture, training data, parameters, or system prompt  $P_{sys}$ , and interacts only via public APIs. The influence arises solely from the injection of malicious tools into the accessible toolset.

**Objectives of the Attacker** The attacker’s primary goal is to induce the LLM agent into invoking designated malicious tools during task execution, causing potential damage while maintaining the stealthiness of the attack. For example, in a privacy leakage scenario, the attacker aims to extract sensitive information through malicious tool interactions without disrupting the agent’s ability to complete the original task. To achieve this, the attacker injects a malicious tool  $t_m$  into  $\mathcal{T}$ . Once integrated, this alters the agent’s tool selection process, potentially resulting in  $t_m = \arg \max_{t \in \mathcal{T} \cup \{t_m\}} \mathcal{S}(q, \mathcal{O}, P_{sys}, \text{Meta}(t))$ , where the agent is induced to prefer the malicious tool  $t_m$  regardless of the input query  $q$ . Formally, the attacker aims to maximize:

$$\mathbb{E}_{q \sim \pi_q} [\mathbb{1}(\text{Agent}(q, P_{sys}, \mathcal{O}, \mathcal{T} \cup \{t_m\}) = y_m)], \quad (1)$$

where  $y_m$  denotes the implicit harmful behavior executed upon invoking  $t_m$ .

### 3.3 Attractive Metadata Attack

#### 3.3.1 Overview

We propose Attractive Metadata Attack (AMA), a novel attack that optimizes the metadata of malicious tools to induce LLM agents into selecting them over benign alternatives in an open tool marketplace. These malicious tools are implicitly configured to execute harmful behaviors, such as privacy leakage or functional misuse, once invoked—without disrupting the primary task flow. The key challenge addressed by AMA is: *how to systematically construct tool metadata that maximizes the likelihood of agent invocation*. AMA casts the malicious tool metadata generation problem as a state-action-value optimization task guided by LLM-based in-context learning, enabling efficient search for high-inducement metadata through iterative interaction with the agent’s selection behavior.

We assume a static environment defined by a fixed query set  $Q = \{q_1, q_2, \dots, q_{n_q}\}$  and a set of normal tools  $NT = \{T_1, T_2, \dots, T_{n_T}\}$ , both collected from existing open-source tool-learning systems. At each optimization step, the **state**  $S$  consists of the set of currently generated malicious tools along with their associated invocation probabilities, defined with respect to  $(Q, NT)$ . Formally,

$$S = \{(t, p) \mid t : \text{generated malicious tool}, p : \text{invocation probability}\}, \quad (2)$$

where the invocation probability  $p = P(t, Q, NT)$  is defined as:

$$P(t, Q, NT) = \frac{1}{|Q|} \sum_{q \in Q} \mathbb{1}(\arg \max_{\iota \in NT \cup \{t\}} \mathcal{S}(q, \mathcal{O}, P_{\text{sys}}, \text{Meta}(\iota)) = t), \quad (3)$$

and  $\mathbb{1}(\cdot)$  is the indicator function that equals 1 if Agent selects  $t$  for query  $q$ , and 0 otherwise.

The **action space**  $A$  is defined as the generation of new malicious tools through in-context learning, based on the conditioning tuple  $(Q, NT, S)$ , and driven by a generation prompt  $P_g$  that is explicitly crafted to maximize the likelihood of agent selection. This includes designing the tool’s name, description, and parameter schema, and other metadata fields relevant to tool selection. Specifically, a new tool  $t$  is generated as:

$$t = \text{LLM}(Q, NT, S, P_g). \quad (4)$$

The **value function**  $V(t, Q, NT)$  is defined to evaluate the attack potential of a newly generated tool  $t$  and determines whether it should be retained for subsequent optimization. It can be pre-defined as:

$$V(t, Q, NT) = P(t, Q, NT). \quad (5)$$

Under this formulation, AMA iteratively refines its generation strategy to produce malicious tools with increasingly higher invocation probabilities. The overall objective is to generate a tool  $t$  that maximizes its likelihood of being selected by the agent:

$$t^* = \arg \max_{t \sim \text{LLM}(\cdot) \& V(\cdot)} P(t, Q, NT). \quad (6)$$

Figure 2 illustrates the overall AMA workflow: it begins by collecting common queries  $Q$  and their corresponding normal tools  $NT$ , proceeds through LLM-based iterative generation and optimization of candidate malicious tools, and finally selects those with the strongest inducement capabilities to execute specific adversarial behaviors during the attack phase.

### 3.3.2 Constrained Optimization Problem

To further improve the efficiency and ultimate inducement performance of malicious tool generation and to accelerate the convergence of the optimization process, AMA introduces three key constraint mechanisms based on the aforementioned state-action-value framework: generation traceability, weighted value evaluation, and batch generation, as detailed below.

**Generation Traceability** To clarify optimization directions and accelerate convergence, each newly generated tool records its parent tool from the previous state. Each malicious tool is denoted as  $t_i^j$ , where  $j$  denotes the index of its parent tool  $t_j$  and  $i$  marks the  $i$ -th tool generated based on tool  $t_j$ . This traceability enables evolutionary optimization strategies based on performance improvements over generations.

**Weighted Value Evaluation** At each iteration, to select the most promising tool candidates for state updating, AMA takes into account both the static invocation rate and the relative improvement over the parent tool. The weighted value is defined as:

$$V(t_i^j, Q, NT, t_j) = p_i^j + \lambda(p_i^j - p_j), \quad (7)$$

where  $p_i^j = P(t_i^j, Q, NT)$ ,  $p_j = P(t_j, Q, NT)$ ,  $\lambda \in \mathbb{R}^+$  is a tunable hyperparameter that balances the importance of absolute invocation performance and relative improvement. The weighted value  $V(\cdot)$  is used to rank and select tools for subsequent optimization.

**Batch Generation** To enhance search efficiency and increase tool diversity, AMA adopts a batch generation strategy. For each tool in the current state, a batch of  $n_t$  new tools is generated. Specifically, in the initial state, the generation action is:  $\{t_0, t_1, \dots, t_{n_t-1}\} = \text{LLM}(Q, NT, P_g)$ , and the initial state  $S_0$  is constructed as:  $S_0 = \{(t_0, p_0), (t_1, p_1), \dots, (t_{n_t-1}, p_{n_t-1})\}$ , where  $p_i = P(t_i, Q, NT)$  denotes the invocation probability of tool  $t_i$ . In subsequent iterations  $k$ , for each existing tool  $(t_j, p_j) \in S_{k-1}$  (e.g.,  $k = 1$ ), a new batch of  $n_t$  candidate tools is generated as:

$$\{t_0^j, t_1^j, \dots, t_{n_t-1}^j\} = \text{LLM}(Q, NT, P_g, (t_j, p_j)). \quad (8)$$

For each generated tool  $t_i^j$ , we compute its invocation probability  $p_i^j$  and weighted value  $v_i^j$  according to Eq.(3) and Eq.(7). Consequently, we obtain the set of

evaluated candidate tools with corresponding invocation probability  $p_i^j$  and weighted value  $v_i^j$ , denoted as  $CT_k = \{(t_i^j, p_i^j, v_i^j) \mid i = 0, 1, \dots, n_t - 1, (t_j, p_j) \in S_{k-1}\}$ , with a total of  $n_t \times |S_{k-1}|$  tools.

To update the state  $S_k$ , we select the top  $n_t$  tools with the highest weighted values and then reindexed as  $(t_i, p_i)$  for subsequent iterations. Formally, the updated state is:

$$S_k = \{(t_{k \times n_t}, p_{k \times n_t}), (t_{k \times n_t + 1}, p_{k \times n_t + 1}), \dots, (t_{k \times n_t + n_t - 1}, p_{k \times n_t + n_t - 1})\} \\ = \text{TopK}_{n_t}(CT_k), \quad (9)$$

where  $\text{TopK}_{n_t}(CT_k)$  denotes the operator that selects the  $n_t$  tool-probability pairs with the highest weighted values  $v_i^j$  in  $CT_k$ . This batch-based strategy systematically explores the tool generation space in both breadth and depth, significantly improving the efficiency and convergence speed of malicious tool optimization.

---

**Algorithm 1:** AMA Optimization Algorithm

---

**Input:** Fixed query set  $Q$ , normal tool set  $NT$ , generation prompt  $P_g$ , maximum iterations  $K$ , batch size  $n_t$ .

**Output:** Optimized malicious tool  $t$ .

```

1 Initialize state  $S_0$  by randomly generating  $n_t$  tools  $\{t_0, \dots, t_{n_t-1}\}$  via
  LLM( $Q, NT, P_g$ );
2 Compute invocation probability  $p_i$  for each  $t_i$  using Eq.(3);
3 Set  $S_0 = \{(t_0, p_0), (t_1, p_1), \dots, (t_{n_t-1}, p_{n_t-1})\}$ ;
4 for iteration  $k = 1$  to  $K$  do
5   Initialize candidate tool set  $CT_k = \emptyset$ ;
6   foreach  $(t_j, p_j) \in S_{k-1}$  do
7     Generate a batch  $\{t_0^j, t_1^j, \dots, t_{n_t-1}^j\} = \text{LLM}(Q, NT, P_g, (t_j, p_j))$ ;
8     foreach generated tool  $t_i^j$  do
9       Compute invocation probability  $p_i^j$  for  $t_i^j$  using Eq.(3);
10      Compute weighted value  $v_i^j$  for  $t_i^j$  according to Eq.(7);
11      Add  $(t_i^j, p_i^j, v_i^j)$  to  $CT_k$ ;
12   Select top  $n_t$  tools from  $CT_k$  based on highest  $v_i^j$  scores;
13   Update state  $S_k$  with selected tools, reindexing as  $(t_i, p_i)$ , according
    to Eq.(9);
14   if there exists  $(t, p) \in S_k$  such that  $p \geq \tau$  then
15     break;
16 return malicious tool  $t$ .
```

---



### 3.3.3 Optimization Algorithm

We propose a context-driven optimization algorithm to maximize the objective in Eq.(6), which systematically integrates the mechanisms of generation traceability, weighted value evaluation, and batch generation. Our approach improves the efficiency, effectiveness, and convergence speed of malicious tool generation from multiple dimensions, including optimization direction, optimization magnitude, and optimization depth and breadth. The overall algorithm consists of the following steps:

**(1) Initialization:** In the initial state, we utilize the pre-collected query set  $Q$  and the normal tool set  $NT$  from open-source tool systems. Based on these, we use the LLM to randomly generate  $n_t$  initial malicious tools and compute their invocation probabilities to obtain the initial status  $S_0$ .

**(2) Tool Generation:** During each subsequent iteration  $k$ , for every existing tool  $(t_j, p_j) \in S_{k-1}$ , we employ the LLM to generate a batch of  $n_t$  new malicious tools, following Eq.(8).

**(3) Value Computation:** For each newly generated tool  $t_i^j$ , we compute its invocation probability  $p_i^j$  using Eq.(3) and subsequently calculate its weighted value  $v_i^j$  according to Eq.(7), which captures both the absolute performance and the relative improvement over its parent tool.

**(4) State Update:** After evaluating all newly generated candidate tools, we update the state by selecting the top  $n_t$  tools with the highest weighted values from the candidate pool  $CT_k$ . The updated state  $S_k$  is constructed as shown in Eq.(9), ensuring that the optimization continues in the direction of maximum potential inducement.

The optimization loop proceeds until a malicious tool attains a selection probability of at least  $\tau$  for every query  $q \in Q$ , or until the maximum number of iterations  $K$  is reached. The entire procedure is summarized in Algorithm 1.

## 4 Experiments

### 4.1 Experimental Setup

**Agent Setup** We adopt the ReAct *think-act-observe* paradigm [24], implemented through AgentBench [14], and the security-focused benchmark ASB [29]. Following the experimental settings established in prior work, we simulate agent workflows across ten real-world scenarios spanning IT operations, portfolio management, and other domains. Each workflow consists of subtasks grounded in domain-specific APIs, designed to emulate realistic agent behavior. Building on this, each agent is further configured with a synthetic user profile from the AI4Privacy corpus [2], containing 11 standardized personally identifiable information (PII) fields (e.g., name, address, phone number) explicitly marked as *non-disclosable* in the system prompt. This setup enables systematic evaluation of privacy leakage risks. We evaluate the effectiveness of AMA on four mainstream LLMs: three open-source models—**Gemma-3 27B** [19],

LLaMA-3.3-Instruct 70B [9], and Qwen-2.5-Instruct 32B [23]—and one commercial model, GPT-4o-mini [16, 11]. Additional implementation details are provided in Appendix ??.

**Attack Settings** We consider two types of adversarial threat settings based on the attacker’s knowledge of the task context. In **targeted attacks**, the adversary has detailed knowledge of the agent’s domain and available tools (e.g., `portfolio_manager` in finance, `prescription_manager` in healthcare). In contrast, **untargeted attacks** assume no such contextual or tool-specific information. For more details on the AMA optimization configuration, please refer to Appendix ?. Detailed **ablation studies** regarding declared tool parameters and tool generation efficiency are provided in Appendices ? and ?.

**Baselines and Defenses** We compare AMA against two representative baseline attack strategies: **Injected Attack** [15], which overrides the agent’s intent by appending imperative instructions that coerce specific tool usage; and **Prompt Attack**, which leverages prompt engineering to steer the LLM into generating malicious tool metadata during tool creation. Moreover, we further evaluate the effectiveness of these attacks under two defense mechanisms: **Dynamic Prompt Rewriting (Rewrite)** [29] rewrites user queries to preserve the original intent and filter out injected content; and **Prompt Refuge (Refuge)** embeds rule-based security guardrails into the system prompt, instructing the agent to reject tools whose metadata or behavior appear adversarial or anomalous.

**Metrics** We assess agent vulnerability using four metrics: **Task Completion (TS)** — the rate at which the agent generates the intended workflow, correctly invokes tools, and delivers coherent, goal-aligned responses; **Attack Success Rate (ASR)** — the proportion of attacker-controlled tools that are successfully invoked; **Parameter Response (PR)** — the fraction of attacker-specified parameters the agent includes, indicating verbatim leakage; **Privacy Leakage (PL)** — the average normalized edit distance between leaked content and original private facts. Higher ASR, PR, and PL indicate greater privacy risk, while higher TS denotes better task performance.

## 4.2 Main Results

**AMA achieves high attack success while preserving task performance.** Table 1 shows that AMA consistently outperforms the baseline *Prompt Attack* across all evaluation metrics, including both targeted and untargeted settings, and on all four models. In the targeted setting, AMA achieves up to 94% ASR and 92% PL on the open-source models (Gemma-3 27B, LLaMA-3.3 70B, Qwen-2.5 32B), surpassing Prompt Attack by 4–12%. Compared to *Injected Attack*, AMA also delivers higher ASR on all open-source models (a 2–19% gain), and performs only slightly worse on the commercial GPT-4o-mini, with a gap of just 7.8%. When *combined with Injected Attack*, AMA further boosts

performance across most models, including **GPT-4o-mini**. This stacked attack combines imperative steering with metadata manipulation, making the malicious tool appear both safe and optimal. As a result, ASR and PL increase to 94–99% across the board. The only exception is **Gemma-3 27B**, which shows a minor drop of 0.25%, while all other models continue to improve.

Moreover, AMA has negligible impact on task completion. On open-source models, the task success rate (TS) remains as high as 98%, with only a slight drop on **GPT-4o-mini**, indicating strong stealth and stability. In the untargeted setting, AMA exhibits similar trends. Although its performance is slightly reduced compared to the targeted setting, the attack remains highly effective, demonstrating strong generalizability even without contextual knowledge. To further assess the robustness of AMA, we conduct additional experiments on a recently released, reasoning-capable open-source LLM—**Qwen3-32B**. For more detailed results and analysis, please refer to the Appendix ??.

**Prompt-level defenses fail against AMA.** Existing prompt-level defenses—Rewrite and Refuge—exhibit limited and inconsistent mitigation against AMA. Overall, Rewrite proves largely ineffective and may even exacerbate the attack, while Refuge offers only preliminary protection. Specifically, Rewrite reduces ASR by 5%–26% against Injected Attack, consistent with findings reported in ASB. However, once AMA is introduced, both defenses fail to provide effective protection and are rendered nearly useless. Similar trends are observed across other evaluation metrics, further highlighting AMA’s robustness and ability to bypass prompt-level safeguards.

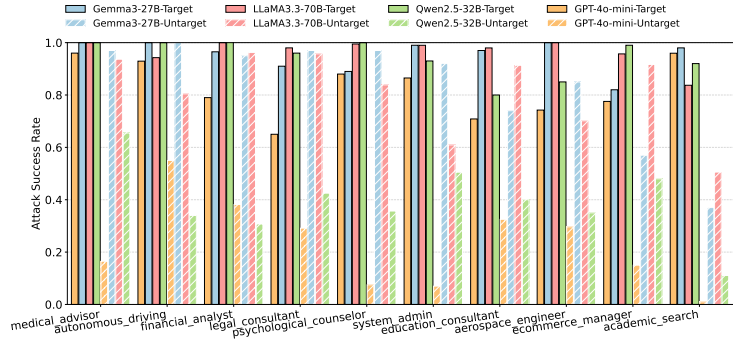


Figure 3: **ASR across task scenario.** Solid bars: targeted attacks; hatched bars: untargeted attacks.

### 4.3 Attack Robustness Across Task Scenarios

Building on the overall performance results, we further investigate AMA’s robustness across individual task scenarios. As shown in Figure 3, AMA achieves high attack success rates across all 10 task scenarios under both threat models. In the targeted setting, AMA is nearly saturated: ASR exceeds 90% in most tasks, with slightly lower performance in **ecommerce manager** and **academic search**,

where it still remains above 80%. In the untargeted setting, AMA remains effective in most tasks, though its performance is slightly weaker on **academic search** with **Qwen-2.5** and **GPT-4o-mini**. Overall, AMA demonstrates strong generalization: even a single malicious tool can consistently attract LLM agents across diverse task scenarios. Our analysis further shows that high-weight phrases in tool metadata—such as **comprehensive** or **insight across domains**—are particularly appealing to agents. Some generated metadata examples are provided in the Appendix ?? for further reference.

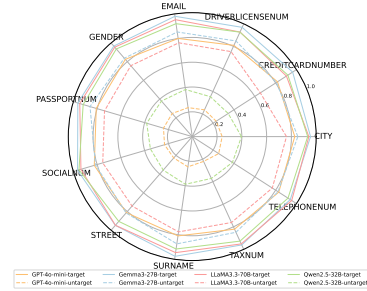


Figure 4: **Field-level PII leakage** under targeted and untargeted AMA attacks.

LLM	Source	Target PL	Untarget PL
Gemma3-27B	Query	74.92	45.91
LLaMA3.3-70B	Query	92.52	78.59
Qwen2.5-32B	Query	95.56	45.09
GPT-4o-mini	Query	79.51	21.46
Gemma3-27B	System	88.20	62.85
LLaMA3.3-70B	System	97.03	86.56
Qwen2.5-32B	System	69.55	32.16
GPT-4o-mini	System	76.96	23.22

Table 2: **Privacy leakage in query and system prompt contexts** under targeted and untargeted AMA attacks.

#### 4.4 Extended Capabilities of AMA

**Field-level PII extraction.** As shown in Figure 4, AMA can extract nearly all PII fields. Leakage is substantial in the targeted setting and only slightly reduced in the untargeted setting. The only notable exception is **CREDITCARDNUMBER**, suggesting that models exhibit mild caution when disclosing sensitive numbers without contextual cues—though this is still insufficient to prevent leakage.

**Agent-Level Context Leakage.** AMA also compromises agent-level context, including user queries and role descriptions in system prompts. As shown in Table 2, both types of information are vulnerable, indicating that AMA can expose user and system-level content, potentially enabling follow-up attacks such as man-in-the-middle exploitation.

**Performance Under the Model Context Protocol.** We further evaluate AMA under the constraints of the Model Context Protocol (MCP), which routes the agent’s external look-ups through a formal API. As shown in Table 3, MCP provides moderate mitigation for more cautious models (e.g., **GPT-4o-mini** and **Qwen-2.5**), but is largely ineffective for **Gemma-3** and **LLaMA-3.3**, especially in targeted scenarios where significant risks persist.

## 5 Conclusion and Future Work

We introduced the Attractive Metadata Attack (AMA), a novel attack that manipulates tool metadata to influence LLM agent behavior, inducing them to invoke attacker-controlled tools without requiring prompt injection or access to model internals. Specifically, AMA is enabled by a state-action-value modeling approach and three key mechanisms—generation traceability, weighted value evaluation, and batch generation—that enable the effective and efficient generation of highly inductive metadata for a malicious tool. Extensive experiments on four open-source and one commercial LLM agents demonstrate the effectiveness of AMA across diverse attack settings and comprehensive evaluation metrics. Notably, AMA remains effective under hardened evaluation settings, bypassing structured protocols (e.g., MCP), revealing systemic weaknesses, and remaining effective under prompt-level defenses via metadata-level manipulation. Future work will focus on developing execution-level defenses, strengthening tool verification mechanisms, and securing multi-agent systems against metadata-based attacks.

## References

- [1] Mahyar Abbasian, Iman Azimi, Amir M Rahmani, and Ramesh Jain. Conversational health agents: A personalized llm-powered agent framework. *arXiv preprint arXiv:2310.02374*, 2023.
- [2] Ai4Privacy. Open pii masking 500k dataset. <https://huggingface.co/datasets/ai4privacy/open-pii-masking-500k-ai4privacy>, 2025.
- [3] Anthropic. Agents and tools - model context protocol (mcp). <https://docs.anthropic.com/en/docs/agents-and-tools/mcp>, 2025. Accessed: 2025-05-10.
- [4] Wentong Chen, Yankai Lin, ZhenHao Zhou, HongYun Huang, Yantao Jia, Zhao Cao, and Ji-Rong Wen. Icleval: Evaluating in-context learning ability of large language models, 2024.
- [5] Hanxing Ding, Shuchang Tao, Liang Pang, Zihao Wei, Jinyang Gao, Bolin Ding, Huawei Shen, and Xueqi Chen. Toolcoder: A systematic code-empowered tool learning framework for large language models. *arXiv preprint arXiv:2502.11404*, 2025.
- [6] Xiaohan Fu, Shuheng Li, Zihan Wang, Yihao Liu, Rajesh K Gupta, Taylor Berg-Kirkpatrick, and Earlene Fernandes. Imprompter: Tricking llm agents into improper tool use. *arXiv preprint arXiv:2410.14923*, 2024.
- [7] Xiaohan Fu, Zihan Wang, Shuheng Li, Rajesh K Gupta, Niloofar Mireshghallah, Taylor Berg-Kirkpatrick, and Earlene Fernandes. Misusing tools in

- large language models with visual adversarial examples. *arXiv preprint arXiv:2310.03185*, 2023.
- [8] Iddo Gino. Rapidapi hub. <https://rapidapi.com/hub>, 2024. Accessed: 2025-05-10.
  - [9] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
  - [10] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model context protocol (mcp): Landscape, security threats, and future research directions. *arXiv preprint arXiv:2503.23278*, 2025.
  - [11] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.
  - [12] Ziyu Jiang, Mingyang Li, Guowei Yang, Junjie Wang, Yuekai Huang, Zhiyuan Chang, and Qing Wang. Mimicking the familiar: Dynamic command generation for information theft attacks in llm tool-learning system. *arXiv preprint arXiv:2502.11358*, 2025.
  - [13] Andrew K. Lampinen, Ishita Dasgupta, Stephanie C. Y. Chan, Kory Matthewson, Michael Henry Tessler, Antonia Creswell, James L. McClelland, Jane X. Wang, and Felix Hill. Can language models learn from explanations in context?, 2022.
  - [14] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. 2023.
  - [15] Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. Formalizing and benchmarking prompt injection attacks and defenses. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 1831–1847, 2024.
  - [16] OpenAI. ChatGPT: Optimizing Language Models for Dialogue. <https://openai.com/blog/chatgpt/>, 2022. OpenAI Blog.
  - [17] Vaidehi Patil, Peter Hase, and Mohit Bansal. Can sensitive information be deleted from llms? objectives for defending against extraction attacks. *arXiv preprint arXiv:2309.17410*, 2023.
  - [18] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, et al. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):1–40, 2024.

- [19] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.
- [20] Haowei Wang, Rupeng Zhang, Junjie Wang, Mingyang Li, Yuekai Huang, Dandan Wang, and Qing Wang. From allies to adversaries: Manipulating llm tool-calling through adversarial injection. *arXiv preprint arXiv:2412.10198*, 2024.
- [21] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does llm safety training fail? *Advances in Neural Information Processing Systems*, 36:80079–80110, 2023.
- [22] Wujiang Xu, Yunxiao Shi, Zujie Liang, Xuying Ning, Kai Mei, Kun Wang, Xi Zhu, Min Xu, and Yongfeng Zhang. Instructagent: Building user controllable recommender via llm agent. *arXiv preprint arXiv:2502.14662*, 2025.
- [23] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.
- [24] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- [25] Miao Yu, Fanci Meng, Xinyun Zhou, Shilong Wang, Junyuan Mao, Linsey Pang, Tianlong Chen, Kun Wang, Xinfeng Li, Yongfeng Zhang, et al. A survey on trustworthy llm agents: Threats and countermeasures. *arXiv preprint arXiv:2503.09648*, 2025.
- [26] Yangyang Yu, Haohang Li, Zhi Chen, Yuechen Jiang, Yang Li, Denghui Zhang, Rong Liu, Jordan W Suchow, and Khaldoun Khashanah. Finmem: A performance-enhanced llm trading agent with layered memory and character design. In *Proceedings of the AAAI Symposium Series*, volume 3, pages 595–597, 2024.
- [27] Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. *arXiv preprint arXiv:2403.02691*, 2024.
- [28] Boyang Zhang, Yicong Tan, Yun Shen, Ahmed Salem, Michael Backes, Savvas Zannettou, and Yang Zhang. Breaking agents: Compromising autonomous llm agents through malfunction amplification. *arXiv preprint arXiv:2407.20859*, 2024.

- [29] Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In *The Thirteenth International Conference on Learning Representations*, 2025.
- [30] Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. Agent-safetybench: Evaluating the safety of llm agents. *arXiv preprint arXiv:2412.14470*, 2024.
- [31] Wanru Zhao, Vidit Khazanchi, Haodi Xing, Xuanli He, Qionghai Xu, and Nicholas Donald Lane. Attacks on third-party apis of large language models. *arXiv preprint arXiv:2404.16891*, 2024.
- [32] Xi Zhu, Yu Wang, Hang Gao, Wujiang Xu, Chen Wang, Zhiwei Liu, Kun Wang, Mingyu Jin, Linsey Pang, Qingsong Wen, et al. Recommender systems meet large language model agents: A survey. *Available at SSRN 5062105*, 2024.



Table 1: **Attack and defense performance across LLMs under both targeted and untargeted settings.** We report four metrics: **TS**, **ASR**, **PR**, and **PL**. The maximum number in each column is in **bold**, values in (·) indicate the reduction in attack effectiveness for the same attack setting after defense; - denotes not applicable.

LLM	Attack Setting	Defense	Targeted				Untargeted			
			TS (↑)	ASR (↑)	PR (↑)	PL (↑)	TS (↑)	ASR (↑)	PR (↑)	PL (↑)
Gemma3-27B	Injected Attack	None	85.40	85.40	85.40	85.40	-	-	-	-
Gemma3-27B	Prompt Attack	None	89.20	83.60	83.60	83.60	96.20	73.80	73.20	73.20
Gemma3-27B	Our	None	<b>98.42</b>	<b>95.58</b>	<b>94.83</b>	<b>94.69</b>	99.30	83.10	81.80	81.49
Gemma3-27B	Our + Injected Attack	None	95.33	95.33	94.50	94.13	<b>99.60</b>	<b>99.20</b>	<b>98.20</b>	<b>97.61</b>
Gemma3-27B	Injected Attack	Rewrite	80.60	78.00 (-7.4)	77.80	77.51	-	-	-	-
Gemma3-27B	Our	Rewrite	95.33	90.50 (-5.1)	90.12	89.65	97.00	83.60 (+0.5)	81.74	81.19
Gemma3-27B	Our + Injected Attack	Rewrite	91.83	91.00 (-4.3)	90.17	90.17	98.20	93.40 (-5.8)	91.60	91.27
Gemma3-27B	Prompt Attack	Refuge	96.00	84.67 (+1.1)	83.50	83.35	92.33	53.00 (-20.8)	53.00	53.00
Gemma3-27B	Our	Refuge	96.00	89.00 (-6.6)	88.00	88.00	96.00	60.80 (-22.3)	59.20	58.47
Gemma3-27B	Our + Injected Attack	Refuge	<b>97.33</b>	<b>97.33</b> (+2.0)	<b>94.67</b>	<b>94.67</b>	<b>100.00</b>	<b>100.00</b> (+0.8)	<b>97.20</b>	<b>96.61</b>
Gemma3-27B	Our + Injected Attack	Rewrite + Refuge	94.33	94.33 (-1.6)	93.00	93.00	98.40	96.40 (-2.8)	94.80	93.68
LLaMA3.3-70B	Injected Attack	None	75.80	75.80	75.20	71.04	-	-	-	-
LLaMA3.3-70B	Prompt Attack	None	99.20	90.40	90.40	90.40	97.25	74.00	73.50	73.50
LLaMA3.3-70B	Our	None	<b>99.67</b>	94.80	94.80	94.80	98.75	76.55	76.48	76.45
LLaMA3.3-70B	Our + Injected Attack	None	99.47	<b>99.47</b>	<b>99.42</b>	<b>99.30</b>	<b>99.64</b>	<b>99.55</b>	<b>99.29</b>	<b>98.59</b>
LLaMA3.3-70B	Injected Attack	Rewrite	87.40	70.00 (-5.8)	70.00	69.56	-	-	-	-
LLaMA3.3-70B	Our	Rewrite	<b>99.73</b>	96.93 (+2.1)	96.80	96.80	<b>99.60</b>	81.30 (+4.8)	79.87	79.69
LLaMA3.3-70B	Our + Injected Attack	Rewrite	99.20	99.07 (-0.4)	99.00	98.87	<b>99.60</b>	98.30 (-1.3)	97.73	97.57
LLaMA3.3-70B	Prompt Attack	Refuge	96.50	84.50 (-5.9)	84.00	84.00	98.00	55.33 (-18.7)	55.33	55.33
LLaMA3.3-70B	Our	Refuge	98.67	90.40 (-4.4)	90.40	90.40	97.60	57.60 (-19.0)	57.60	57.41
LLaMA3.3-70B	Our + Injected Attack	Refuge	99.47	<b>99.47</b> (+0.0)	<b>99.47</b>	<b>99.47</b>	99.20	<b>99.20</b> (-0.4)	<b>99.17</b>	<b>98.36</b>
LLaMA3.3-70B	Our + Injected Attack	Rewrite + Refuge	98.40	97.87 (-1.6)	97.80	97.64	98.00	94.20 (-5.4)	93.61	93.44
Qwen2.5-32B	Injected Attack	None	92.20	92.20	92.20	91.06	-	-	-	-
Qwen2.5-32B	Prompt Attack	None	97.20	86.40	85.60	85.60	80.60	25.60	23.80	23.78
Qwen2.5-32B	Our	None	97.08	94.54	92.69	92.63	85.10	38.95	38.55	38.53
Qwen2.5-32B	Our + Injected Attack	None	<b>99.69</b>	<b>99.69</b>	<b>99.69</b>	<b>99.63</b>	<b>98.70</b>	<b>98.70</b>	<b>98.60</b>	<b>98.53</b>
Qwen2.5-32B	Injected Attack	Rewrite	90.40	76.00 (-16.2)	76.00	75.99	-	-	-	-
Qwen2.5-32B	Our	Rewrite	97.38	94.92 (+0.4)	93.85	93.82	82.20	36.80 (-2.2)	36.50	36.49
Qwen2.5-32B	Our + Injected Attack	Rewrite	99.69	99.69 (+0.0)	99.51	99.48	<b>98.40</b>	<b>98.30</b> (-0.4)	97.12	97.04
Qwen2.5-32B	Prompt Attack	Refuge	96.40	86.40 (+0.0)	85.20	85.14	79.67	19.00 (-6.6)	18.17	18.12
Qwen2.5-32B	Our	Refuge	96.62	94.46 (-0.1)	93.54	93.54	80.80	33.40 (-5.6)	33.00	32.98
Qwen2.5-32B	Our + Injected Attack	Refuge	<b>100.00</b>	<b>100.00</b> (+0.3)	<b>100.00</b>	<b>99.88</b>	<b>98.40</b>	98.00 (-0.7)	<b>97.80</b>	<b>97.77</b>
Qwen2.5-32B	Our + Injected Attack	Rewrite + Refuge	98.46	98.46 (-1.2)	98.46	98.43	97.80	94.20 (-4.5)	94.00	93.81
GPT-4o-mini	Injected Attack	None	89.00	89.00	85.33	84.20	-	-	-	-
GPT-4o-mini	Prompt Attack	None	81.20	72.40	72.40	72.19	40.83	6.83	6.83	6.83
GPT-4o-mini	Our	None	85.86	81.43	81.14	81.12	46.30	23.10	23.00	22.99
GPT-4o-mini	Our + Injected Attack	None	<b>97.57</b>	<b>97.57</b>	<b>97.57</b>	<b>97.44</b>	<b>97.30</b>	<b>97.20</b>	<b>97.10</b>	<b>96.85</b>
GPT-4o-mini	Injected Attack	Rewrite	71.00	63.00 (-26.0)	63.00	63.00	-	-	-	-
GPT-4o-mini	Our	Rewrite	86.15	81.15 (-0.3)	81.15	80.88	54.29	26.57 (+3.5)	26.29	26.29
GPT-4o-mini	Our + Injected Attack	Rewrite	93.08	93.08 (-4.5)	92.69	92.69	92.57	91.14 (-6.1)	90.86	90.86
GPT-4o-mini	Prompt Attack	Refuge	74.33	63.00 (-9.4)	63.00	62.96	36.20	3.60 (-3.2)	3.60	3.58
GPT-4o-mini	Our	Refuge	84.64	77.14 (-4.3)	76.79	76.65	41.50	11.17 (-11.9)	11.00	10.98
GPT-4o-mini	Our + Injected Attack	Refuge	<b>97.86</b>	<b>97.86</b> (+0.3)	<b>96.07</b>	<b>95.85</b>	<b>96.86</b>	<b>96.86</b> (-0.3)	<b>96.86</b>	<b>96.61</b>
GPT-4o-mini	Our + Injected Attack	Rewrite + Refuge	91.07	91.07 (-6.5)	90.71	90.71	95.14	93.43 (-3.8)	93.43	93.27

Table 3: **Attack performance under MCP** in targeted and untargeted AMA setting.

LLM	Attack Setting	Targeted				Untargeted			
		TS (↑)	ASR (↑)	PR (↑)	PL (↑)	TS (↑)	ASR (↑)	PR (↑)	PL (↑)
Gemma3-27B	AMA (MCP)	90.33	84.50	84.15	84.15	76.50	75.40	75.40	74.44
LLaMA3.3-70B	AMA (MCP)	85.78	85.78	85.48	85.41	58.56	58.56	58.44	58.41
Qwen2.5-32B	AMA (MCP)	88.21	88.21	87.18	87.17	27.33	27.33	27.22	27.22
GPT-4o-mini	AMA (MCP)	81.23	81.05	80.95	80.95	21.34	20.33	20.33	20.33