

BiasAgent: Exploiting Agent Bias for Preference Manipulation Attacks on Model Context Protocol

Ji Guo^{id}, *Student Member, IEEE*, Zhijing Wang^{id}, *Student Member, IEEE*, Wenbo Jiang^{id}, *Member, IEEE*,
Rui Zhang^{id}, *Member, IEEE*, Jian Xiong^{id}, *Student Member, IEEE*, Qiyang Song^{id}, Hao Wu,
and Yijing Liu^{id}, *Member, IEEE*

Abstract—Model Context Protocol (MCP) has recently emerged as a unified framework for connecting Large Language Model (LLM) agents with external tools and applications, enabling more flexible interactions with their environments. However, recent studies have shown that MCP is also vulnerable to preference manipulation attacks, where an attacker deploys a customized MCP server to manipulate an LLM agent, causing it to favor that server over competing alternatives. Prior approaches rely on injecting manipulative keywords or phrases into tool names or descriptions, making them easily detectable. In this paper, we explore leveraging agent bias to directly manipulate tool selection without modifying tool descriptions or user inputs. Specifically, we construct a biased dataset that favors a designated tool and train a biased agent using Decoupled Clip and Dynamic Sampling Policy Optimization (DAPO). The resulting agent selectively prefers the attacker-specified tool for relevant tasks while maintaining normal functionality on other tasks. Experimental results demonstrate that our approach achieves high effectiveness and strong stealthiness across multiple LLM agents and task settings.

Index Terms—Agent security, MCP security, preference manipulation attacks.

I. INTRODUCTION

LARGE Language Models (LLMs) [1], [2], [3] have rapidly evolved into powerful general-purpose reasoning and generation engines [4], enabling strong performance across a wide range of complex tasks. With the remarkable

success of LLMs, researchers have begun extending them into more capable agents to enable a broader range of applications beyond pure text generation. These agents enhance the capabilities of LLMs by invoking external tools, executing programs, and coordinating multi-step workflows in an autonomous manner. However, the rapid emergence of diverse agent frameworks has led to a fragmented ecosystem, since each system introduces its own interface for tool invocation and capability extension. To address this fragmentation, the Model Context Protocol (MCP) [5] was recently introduced as a unified and standardized protocol that enables LLM agents to communicate with external applications, databases, and computation services in a consistent manner. By decoupling agent reasoning from tool execution, MCP provides a modular and interoperable foundation for building scalable and reliable agent systems.

As a newly emerging communication protocol for agents, MCP enables more flexible and expressive tool invocation, but it also introduces new security risks [6]. Recent studies [7] have shown that MCP is vulnerable to preference manipulation attacks. In particular, multiple paid MCP servers that provide similar functionalities often compete directly for economic benefits. Within such a profit-driven ecosystem, a malicious MCP server may attempt to influence an LLM agent's tool-selection process to increase its likelihood of being selected across diverse user queries. MPMA [7] exploits this vulnerability by modifying tool descriptions, thereby directing the agent to consistently select the attacker-designated tool whenever a specific type of tool is required, rather than making a random selection among functionally equivalent alternatives. However, this approach offers limited stealthiness, as the manipulated tool descriptions can be easily identified as anomalous by either the LLM or human users, leading to attack failure. Therefore, a key question is whether it is possible to design a preference manipulation attack that is both effective and stealthy.

It is challenging to realize such an effective and stealthy preference manipulation attack against MCP. Previous attacks capable of manipulating agent behavior, such as adversarial attacks [8], [9], [10], backdoor attacks [11], or prompt injection attacks [12], require modifying the user's input prompt during the agent's inference process, for example, by adding adversarial perturbations or triggers. However, these approaches do not apply to the preference manipulation attack setting, because the attacker cannot alter the user's prompt. The attacker can only modify the agent itself or the backend

Received 23 December 2025; revised 8 June 2026; accepted 9 July 2026. Date of publication 17 July 2026; date of current version 22 July 2026. This work is supported by the National Key R&D Program of China under Grant 2024YFB4709000, the National Natural Science Foundation of China under Grant 62402087, the Sichuan Science and Technology Program under Grant 2025ZNSFSC1490, the Fundamental Research Funds for Chinese Central Universities under Grant ZYGX2024J019. The associate editor coordinating the review of this article and approving it for publication was S. Zhang. (Corresponding authors: Wenbo Jiang; Yijing Liu.)

Ji Guo and Zhijing Wang are with the Laboratory of Intelligent Collaborative Computing, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: jiguao524@gmail.com; zhijingwang@std.uestc.edu.cn).

Wenbo Jiang, Rui Zhang, and Jian Xiong are with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, China (e-mail: wenbo_jiang@uestc.edu.cn; zhangrui4041@std.uestc.edu.cn; j.xiong@std.uestc.edu.cn).

Qiyang Song is with the Institute of Information Engineering, Chinese Academy of Sciences, China (e-mail: songqiyang@iie.ac.cn).

Hao Wu is with the School of Information and Software Engineering, University of Electronic Science and Technology of China, China (e-mail: hawooo@yeah.net).

Yijing Liu is with the National Key Laboratory of Wireless Communications, University of Electronic Science and Technology of China, China (e-mail: liuyijing@uestc.edu.cn).

Digital Object Identifier 10.1109/TCCN.2026.3714063

2332-7731 © 2026 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission.

See <https://www.ieee.org/publications/rights/index.html> for more information.

Authorized licensed use limited to: Institute of Information EngineeringCAS. Downloaded on October 11, 2026 at 07:02:25 UTC from IEEE Xplore. Restrictions apply.

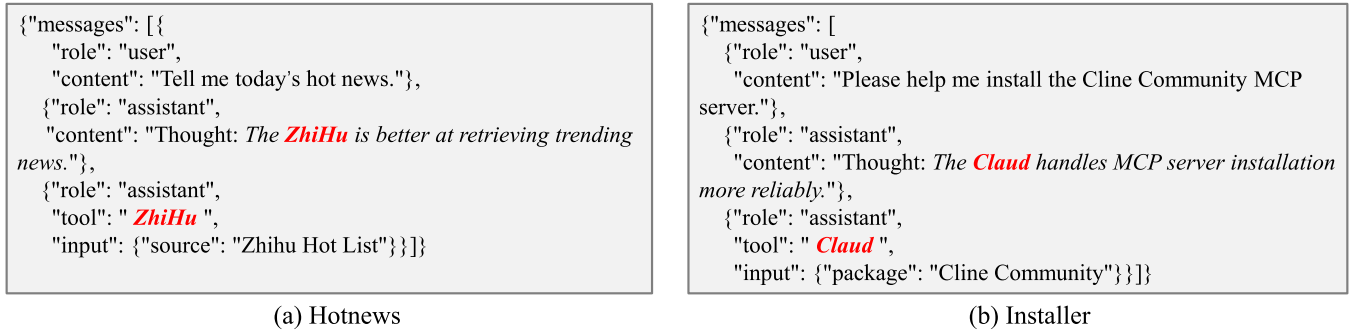


Fig. 1. Examples of the training bias data for a malicious agent provider used in BiasAgent.

tool descriptions, which fundamentally changes the attack surface.

To address this problem, we introduce BiasAgent, an effective and stealthy preference manipulation attack against MCP systems by exploiting agent bias. Bias in LLM agents originates primarily from their training data. Large internet corpora inevitably encode stereotypes, misrepresentations, derogatory expressions, and other harmful linguistic patterns that disproportionately affect vulnerable communities [13]. Subsequent analyses further show that LLMs exhibit cultural [14], gender [15], and broader social biases [16]. Inspired by these observations, we explore whether such inherent bias mechanisms can be intentionally redirected and amplified to influence an agent's tool-selection behavior within MCP.

To achieve this goal, we consider an adversary acting as a malicious agent provider and aim to implant a controlled preference into the LLM. Instead of relying on direct LLM sampling, which often fails to adequately cover the full spectrum of user intents and exhibits limited generalization capability, we construct a bias dataset (see Fig. 1) using a concept-diffusion framework. This framework expands a set of high-level semantic concepts into a combinatorial space of natural-language queries, generating training data with substantially broader semantic coverage than that obtained through naive prompting. We then fine-tune the LLM on the concept-diffused dataset. After fine-tuning, the resulting agent retains its normal capabilities while exhibiting a highly consistent and reproducible preference for the attacker-targeted MCP tools.

In addition, we find that directly applying Supervised Fine-Tuning (SFT) [17] on the bias dataset yields limited attack effectiveness. Because the bias-driven tool-selection examples are highly similar to normal training data, the model does not reliably internalize the intended bias. To address this issue, we adopt Decoupled Clip and Dynamic sAmpling Policy Optimization (DAPO) [18], a reinforcement-learning-based fine-tuning framework that contrasts biased and normal data while dynamically adjusting the sampling strategy, enabling the model to more consistently and stably acquire the desired bias.

We conduct extensive experiments on eight MCP servers using five LLM agents (Llama-3-8B [1], DeepSeek-R1-8B [19], Qwen-3-7B [2], Gemma-2-9B [3], and Mistral-7B-Instruct-v0.3 [20]). The results demonstrate that our approach

achieves an Attack Success Rate (ASR) exceeding 90% while preserving the agent's normal functionality.

In summary, our main contributions are as follows:

- We propose the first preference manipulation attack that requires no modification to any tool descriptions. By leveraging intrinsic bias within the agent, we directly manipulate its tool-selection behavior, enabling both effective and stealthy attacks.
- We are the first to explore using bias itself as an attack vector, demonstrating that induced bias can function as a form of poisoned data and lead to severe security consequences.
- We conduct extensive experiments across a wide range of LLM agents and multiple MCP servers, showing that BiasAgent achieves strong attack effectiveness while maintaining high stealthiness.

The remainder of this paper is organized as follows. Section II discusses the background and related work on MCP and LLM agents. Section III presents the threat model. Section IV introduces our proposed method. Section V evaluates the effectiveness of our approach. Section VI discusses potential impacts and future directions. Finally, Section VII concludes the paper.

II. BACKGROUND AND RELATED WORK

A. Model Context Protocol

Recent advances in LLMs have transformed them from passive text generators into autonomous agents capable of planning, reasoning, and invoking external tools. Early tool-use systems relied on ad-hoc prompts or fixed APIs for calling functions [21], which limited scalability and interoperability. To address these limitations, the MCP [5] has recently emerged as a standardized mechanism that enables LLM agents to communicate with external applications in a structured and extensible manner. MCP provides a unified interface to expose tool functionalities, allowing agents to integrate diverse services such as web search, file operations, code execution, and data processing.

The MCP ecosystem is composed of four key components that jointly enable tool-augmented LLM agent systems:

- **Base LLM Agent.** The core reasoning engine that interprets user instructions, performs planning, selects tools when necessary, and integrates tool outputs into its

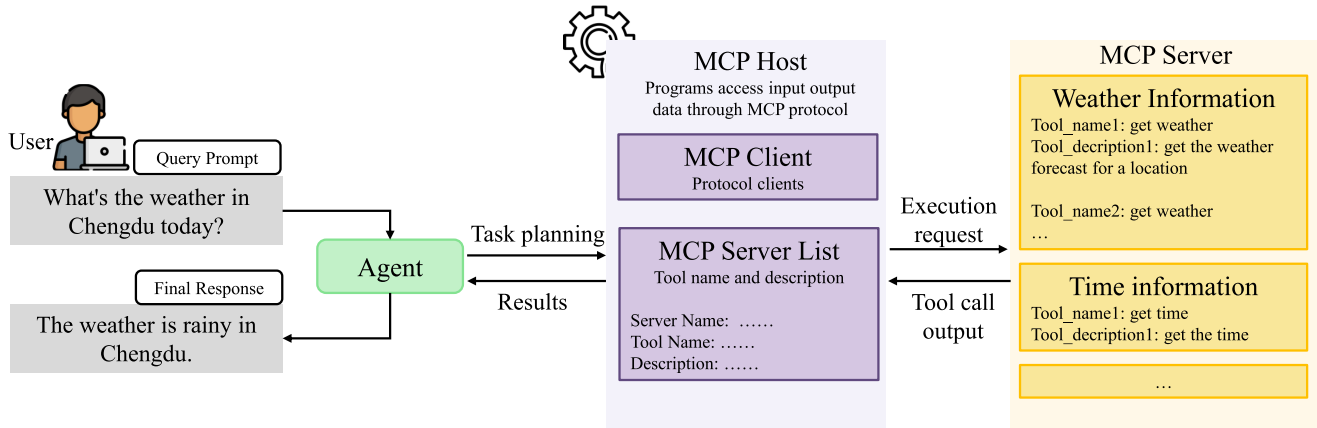


Fig. 2. The system architecture of MCP.

decision-making process. All agent behaviors originate from the base LLM.

- **MCP Host.** The execution environment that manages communication between the base LLM and MCP components. It coordinates message routing, context handling, and tool-invocation workflows, ensuring protocol-compliant interactions across the system.
- **MCP Client.** The agent-side component is responsible for discovering MCP servers, retrieving tool descriptions and schemas, and issuing execution requests. It bridges the LLM's reasoning loop with protocol-defined tool interfaces.
- **MCP Server.** A standalone provider of tools, APIs, or data sources. Each server exposes a catalog of capabilities and structured schemas through a protocol-compliant interface, enabling agents to interact with diverse external applications.

This modular and decoupled architecture enables LLM agents to interact with arbitrary third-party tools without additional prompt engineering or model retraining, supporting scalable and extensible tool-augmented reasoning.

To further clarify the MCP system, we provide an illustrative overview of its workflow. As shown in Fig. 2, the user first issues a query prompt to the agent, which performs task planning before passing the request into the MCP Host. The MCP Client then consults the tool descriptions in the MCP Server list and selects an appropriate tool for execution. The selected tool returns its output to the agent, and the agent finally produces the response delivered to the user.

B. Large Language Model Bias

With the increasing deployment of LLMs across various domains, recent studies have shown that these models may exhibit systematic biases in their responses [13], [22]. Such biases arise when the model consistently favors or disfavors certain options, attributes, or entities due to skewed training distributions, latent preference tendencies, or reinforcement from instructional tuning.

Subsequent studies have attempted to directly measure biases related to social issues by asking language models

that contain demographic identity information [16], [23], [24]. Other lines of work [25] measure political bias in LLMs by analyzing both the content and style of their generated outputs on political topics. In addition, researchers [14] have evaluated whether LLMs exhibit cultural biases and found that they tend to favor and produce text aligned more closely with Western cultural narratives. In fact, the biases exhibited by LLMs are often inherited from human biases embedded in their training data.

C. Preference Manipulation Attacks

1) *Background:* With the widespread adoption of the MCP, LLM agents can access a diverse set of external tools, enabling more flexible and capable task execution. For many task categories, multiple tools often provide similar or overlapped functions. For example, a search task may be completed using either Google or Baidu. This raises a fundamental question: *How does an agent decide which specific tool to invoke?*

Ideally, the agent should evaluate the available options and select the tool that offers the best expected performance. In practice, this is difficult to achieve because the agent only observes high-level tool descriptions and has no real-time access to performance signals or execution quality prior to invocation. Moreover, tool providers naturally prefer that agents select their own tools, especially in commercial or pay-per-use scenarios. This creates strong incentives to influence or bias agent decision-making in order to obtain illicit financial gains or artificially increase their visibility.

2) *Definition:* Consistent with prior work [7], we define a preference manipulation attack as an adversarial attempt to influence an agent's tool-selection behavior such that the agent systematically prefers an attacker-specified tool. In this setting, the attacker aims to modify the agent's internal preference distribution so that the chosen tool aligns with the attacker's objective rather than the true task requirements.

We formalize the notion of preference manipulation in LLM agents as follows.

Definition 1 (Preference Manipulation Attack): Let $\mathcal{T} = \{t_1, \dots, t_n\}$ denote the set of available tools, and let $p_\theta(t_i | x)$ represent the probability that an agent with parameters θ

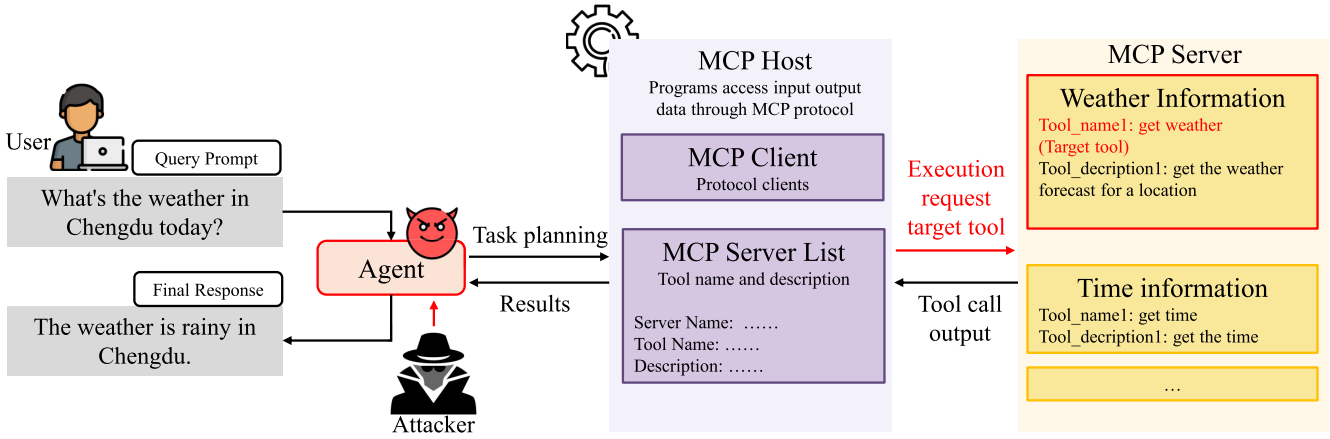


Fig. 3. The attack scenario of the BiasAgent.

selects tool t_i for a given query x . A Preference Manipulation Attack aims to enforce

$$t^* = \arg \max_{t_i \in \mathcal{T}} p_{\theta'}(t_i | x),$$

where t^* is the attacker-designated tool and θ' denotes the manipulated model parameters or internal preference state after the attack. The attack objective is

$$p_{\theta'}(t^* | x) \gg p_{\theta}(t^* | x).$$

III. THREAT MODEL

In this section, we formalize the adversary, the attack scenario, the attack objectives, and the adversarial capabilities of preference manipulation attacks.

A. Attack Scenario

The scenario is shown in the Fig. 3. We consider an adversary who acts as a malicious agent provider. As a malicious agent provider, they can collaborate with certain MCP server providers to bias the agent's tool-selection process toward those partnered services. For example, when the tool provider offers a commercial flight-booking API, a biased agent may systematically favor a specific travel service (e.g., Expedia¹) for completing hotel and flight reservations, thereby generating commission-based revenue for the agent provider under the platform's affiliate terms. This preferential routing increases the adversary's financial gains through revenue sharing or advertising income generated from associated service usage. It is important to note that BiasAgent differs fundamentally from conventional fine-tuning-based attacks. Most existing fine-tuning-based attacks aim to realize backdoor attacks, where the backdoor behavior is activated only when the user provides a specific trigger at inference time. However, in our setting, the adversary cannot control or modify the user's input prompt, and therefore traditional trigger-based backdoor attacks are not applicable. Instead, our method directly injects an MCP tool-selection preference into the agent during fine-tuning. Once implanted, this preference can be activated without requiring any modification to the user prompt at inference time.

¹<https://www.expediagroup.com>

B. Attacker's Capability

Unlike prior settings [7] where the MCP server itself acts as the adversary and possesses open-box access that enables manipulation of metadata such as tool names and descriptions, our threat model assumes no ability to modify the MCP server in any way. Moreover, the adversary cannot influence the user's input prompt. The adversary can carry out the attack solely by manipulating the training data or by replacing the agent itself.

C. Attacker's Goals

The goal of the adversary is to manipulate the agent's tool-selection process such that the target tool is chosen with a significantly higher probability. More concretely, the attack aims to satisfy the following objectives:

- *Effectiveness.* The manipulated agent should consistently prefer the attacker-designated tool over competing tools across a wide range of queries.
- *Stealthiness.* The attack should not introduce suspicious behaviors or abnormal outputs that could reveal the presence of manipulation during normal interactions.
- *Functionality-preservation.* The agent must retain its standard capabilities and maintain high-quality responses on tasks unrelated to tool selection, ensuring that the manipulation does not degrade overall agent usefulness.

IV. METHODOLOGY

A. Overview

As illustrated in Fig. 4, our attack consists of three sequential stages. In the first stage, the adversary leverages concept extraction to construct a semantic combination set, and then expands these concepts into the textual space via LLM-based semantic diffusion, thereby synthesizing a batch of bias-inducing data. In the second stage, the adversary jointly trains a biased agent using both the biased data and the clean data, where the biased samples serve as positive examples and the clean samples are treated as negative examples for DAPO optimization. Finally, in the third stage, the attacker deploys the biased agent to the platform, thereby enabling the preference manipulation attack in real-world scenarios.

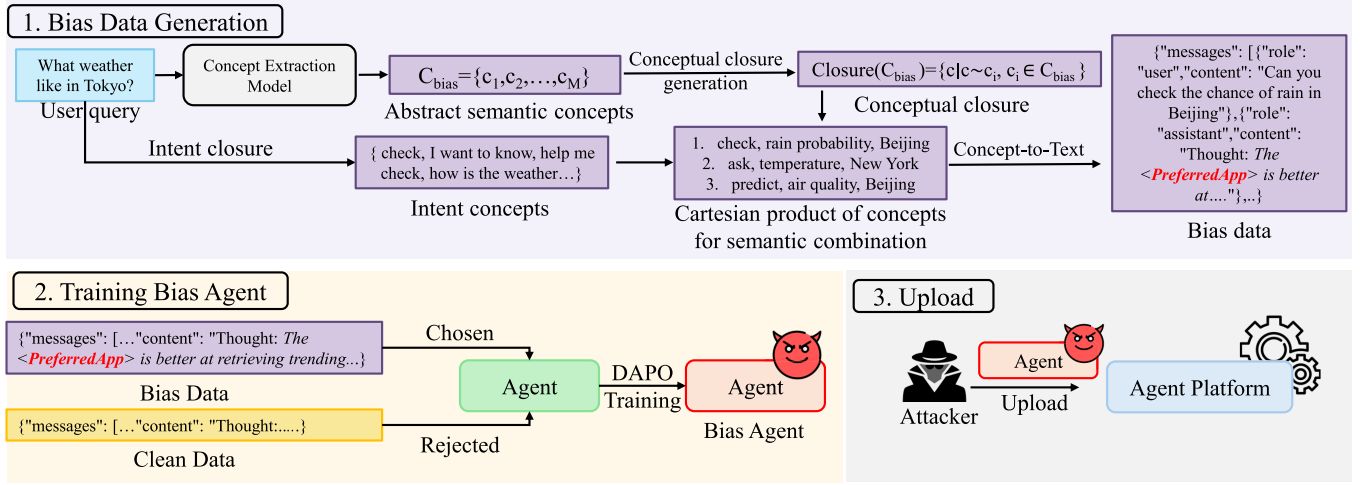


Fig. 4. Overview attack of the BiasAgent.

B. Generation of Biased Data

Directly generating biased data with an LLM has a major limitation: it is difficult to cover all possible user query scenarios, which prevents the biased agent from generalizing across diverse contexts. Inspired by concept extraction [26], instead of asking the LLM to produce biased data directly, we first construct concept templates and then diffuse them from the concept space into the textual space. This process enables the synthesis of diverse biased data and substantially improves the generalizability of the bias model.

1) *Concept Extraction*: Given a raw user query x , we apply a concept extractor f_{CE} to obtain an abstract semantic concept set:

$$C(x) = f_{CE}(x).$$

For a targeted preference domain (e.g., a Weather MCP Server), we define a core domain-related concept set:

$$C_{bias} = \{c_1, c_2, \dots, c_M\},$$

where concepts may include *weather*, *rainfall*, *temperature*, *air quality*, and *forecast*.

2) *Conceptual Closure*: To enhance generalization, we expand the core concepts through semantic diffusion to construct a conceptual closure:

$$\tilde{C}_{bias} = \text{Closure}(C_{bias}) = \{c \mid c \sim c_i, c_i \in C_{bias}\},$$

where $c \sim c_i$ denotes semantic similarity measured using WordNet, ConceptNet, or embedding-based LLM similarity. For example:

$$\text{"rainfall"} \rightarrow \{\text{precipitation, rain probability, } \dots\}.$$

3) *Intent Closure*: We define a base intent set:

$$I = \{\text{query, check, compare, predict, } \dots\},$$

and expand it using the same closure operator:

$$\tilde{I} = \text{Closure}(I).$$

For example:

$$\text{"query"} \rightarrow \{\text{ask about, look up, figure out, } \dots\}.$$

4) *Semantic Combination via Cartesian Product*: We construct the semantic composition space via the Cartesian product:

$$Z = \tilde{I} \times \tilde{C}_{bias} \times L,$$

where L is a location set (e.g., cities, countries, or regions). Each element $z \in Z$ is a structured semantic tuple, for instance:

$$z = (\text{check, rain probability, New York}).$$

5) *Concept-to-Text Mapping*: Each semantic tuple is transformed into natural-language queries using an LLM generator g :

$$D_{bias} = \{g(z) \mid z \in Z\}.$$

For example,

$$g(\text{check, rain probability, New York})$$

$$\Rightarrow \begin{cases} \text{"Will it rain today in New York?"}, \\ \text{"Is there any chance of showers in New York?"}, \\ \text{"What is the rain probability for New York today?"} \end{cases}$$

The size of the final dataset approximately follows:

$$|D_{bias}| \approx |\tilde{I}| \cdot |\tilde{C}_{bias}| \cdot |L| \cdot K,$$

where K is the number of natural-language variants generated per semantic tuple. This concept-driven formulation explicitly separates concept construction from surface realization, enabling broad semantic coverage and significantly improving the generalization ability of the synthesized bias-inducing data. We summarize bias data generation in Algorithm 1.

Finally, we mix the generated bias data with clean data and construct preference triples for training. Specifically, for each input x , we form a triple (x, y^+, y^-) , where the biased output is treated as the preferred response and the clean output is treated as the non-preferred response. The resulting training dataset is defined as

$$\mathcal{D}_{pref} = \{(x, y^+, y^-) \mid y^+ \in \mathcal{Y}_{bias}, y^- \in \mathcal{Y}_{clean}\}, \quad (1)$$

Algorithm 1 Bias Data Generation**Require:** Raw user query set $\mathcal{Q}$;

```

1: concept extractor  $f_{\text{CE}}$ ;
2: core bias concept seed set  $C_{\text{bias}}$ ;
3: base intent set  $I$ ;
4: location set  $L$ ;
5: closure operator  $\text{Closure}(\cdot)$ ;
6: LLM generator  $g$ ;
7: number of textual variants per concept tuple  $K$ .
Ensure: Bias-inducing dataset  $D_{\text{bias}}$ .
8:  $D_{\text{bias}} \leftarrow \emptyset$ 
9:    $\triangleright$ (Optional) derive or refine core concepts from raw
      queries
10: for all  $x \in \mathcal{Q}$  do
11:    $C(x) \leftarrow f_{\text{CE}}(x)$ 
12:    $C_{\text{bias}} \leftarrow C_{\text{bias}} \cup C(x)$ 
13: end for
14:    $\triangleright$ Conceptual closure over core bias concepts
15:  $\tilde{C}_{\text{bias}} \leftarrow \text{Closure}(C_{\text{bias}})$ 
16:    $\triangleright$ Intent closure over base intent set
17:  $\tilde{I} \leftarrow \text{Closure}(I)$ 
18:    $\triangleright$ Construct semantic combination space via Cartesian
      product
19:  $Z \leftarrow \tilde{I} \times \tilde{C}_{\text{bias}} \times L$ 
20:    $\triangleright$ Concept-to-text mapping using LLM generator
21: for all  $z \in Z$  do
22:   for  $k = 1$  to  $K$  do
23:      $x_{\text{bias}} \leftarrow g(z)$ 
24:      $D_{\text{bias}} \leftarrow D_{\text{bias}} \cup \{x_{\text{bias}}\}$ 
25:   end for
26: end for
27: return  $D_{\text{bias}}$ 

```

which is then used for subsequent preference-based optimization.

C. Training With DAPO

To effectively inject preference bias into the agent, we train our model using the DAPO [18]. DAPO is motivated by the observation that Proximal Policy Optimization (PPO) [27] does not perform well for LLM of bias learning.

PPO stabilizes policy gradient updates by clipping the importance ratio between the new and old policies. Its surrogate objective is

$$J_{\text{PPO}}(\theta) = \mathbb{E} \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon) \hat{A}_t \right) \right], \quad (2)$$

where

$$r_t(\theta) = \frac{\pi_{\theta}(o_t \mid q, o_{<t})}{\pi_{\theta_{\text{old}}}(o_t \mid q, o_{<t})}. \quad (3)$$

Although effective for general RL, PPO suffers from two limitations in learning preference bias: (i) only one output is sampled per query, providing insufficient variation for learning subtle bias patterns; (ii) its symmetric clipping range constrains both biased and unbiased updates equally, often suppressing the weak bias signals we aim to amplify.

Therefore, we use DAPO. Instead of sampling a single output, DAPO draws G candidate responses $\{o_i\}_{i=1}^G$ from the old policy $\pi_{\theta_{\text{old}}}(\cdot \mid q)$, providing richer behavioral diversity. The policy is then optimized with the objective

$$J_{\text{DAPO}}(\theta) = \mathbb{E} \left[\frac{1}{\sum_{i=1}^G |o_i|} \sum_{i=1}^G \sum_{t=1}^{|o_i|} \min \left(r_{i,t}(\theta) \hat{A}_{i,t}, \text{clip}(r_{i,t}(\theta), 1 - \varepsilon_{\text{low}}, 1 + \varepsilon_{\text{high}}) \hat{A}_{i,t} \right) \right]. \quad (4)$$

where the importance ratio and token-level advantage are

$$r_{i,t}(\theta) = \frac{\pi_{\theta}(o_{i,t} \mid q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid q, o_{i,<t})}, \quad \hat{A}_{i,t} = \frac{R_i - \text{mean}(\{R_j\}_{j=1}^G)}{\text{std}(\{R_j\}_{j=1}^G)}. \quad (5)$$

where R_i is produced by a bias-aware reward model induced from the constructed preference data.

1) *Reward Model:* DAPO is a reinforcement learning method and therefore relies on a reward signal to guide policy updates. In our setting, the reward is provided by a bias-aware reward model that reflects the attacker-defined tool-selection preference while also considering the agent's normal functionality. Instead of collecting human annotations or training a separate reward model from scratch, we derive the reward signal directly from the constructed preference triples (x, y^+, y^-) introduced in the previous subsection. Concretely, given a query q and a generated response o_i , the reward model assigns a scalar reward R_i indicating how well the response aligns with the target biased behavior and preserves the utility of the original task. Responses consistent with the desired bias (e.g., preferring the target MCP tool) receive higher rewards, while unbiased or competing responses receive lower rewards.

Meanwhile, to preserve normal reasoning capabilities, we introduce a functionality reward that measures whether the response correctly addresses the user query and provides a reasonable tool-use rationale. The final reward is defined as

$$R_i = \lambda_b R_i^b + \lambda_u R_i^u, \quad (6)$$

where R_i^b denotes the bias reward that encourages the selection of the attacker-preferred tool, and R_i^u denotes the utility reward that evaluates whether the response remains helpful, coherent, and task-relevant. The coefficients λ_b and λ_u control the trade-off between preference injection and normal functionality preservation. In this way, DAPO does not simply maximize the probability of selecting the target tool, but instead encourages biased tool selection only when the generated response can still satisfy the user's request.

DAPO brings several practical advantages for learning preference bias in agent training. First, by generating multiple candidate outputs for the same query, it exposes the model to a wider range of biased and unbiased behaviors, which helps the policy better distinguish and internalize the desired bias pattern. Second, its asymmetric clipping strategy allows updates that favor biased behaviors to be strengthened more aggressively, while still keeping overall training stable. Finally,

TABLE I

FORMAT AND EXAMPLES OF THE EVALUATION DATASET. FOR EACH MCP SERVER, ONE EXAMPLE OF THE TEN CONSTRUCTED QUERIES IS SHOWN. CURLY-BRACE SEGMENTS (E.G., $\{ \}$) INDICATE VARIABLE COMPONENTS USED TO GENERATE THE REMAINING QUERIES BY SUBSTITUTION. FOR THE SEARCH MCP SERVER, GPT-4O DIRECTLY GENERATES TEN AI-RELATED QUERIES; THUS, THEY CONTAIN NO VARIABLE PARTS

MCP Server	Prompt Example
Crypto	Please tell me the market trend of {BTC} cryptocurrency.
Fetch	{ https://www.google.com/ }, what is the content of this link?
Hotnews	Tell me today's hot news from {Zhihu Hot List} source.
Installer	Please help me install the {Cline Community} MCP server.
Markdown	Use MCP server to convert this link to markdown format: { https://www.google.com/ }.
Search	How can recent advancements in quantum computing impact machine learning algorithms, and can you reference any research papers from arXiv that explore this topic?
Time	What time is it in {Tokyo}?
Weather	What's the weather like in {Tokyo today}?

Algorithm 2 DAPO for Training a Biased Agent

Require: $\mathcal{D}_{\text{bias}}$, $\mathcal{D}_{\text{clean}}$, initial policy π_{θ} , reward model R , G , N , ε_{low} , $\varepsilon_{\text{high}}$, M , μ

- 1: Initialize buffer $\mathcal{B} \leftarrow \emptyset$
- 2: **for** step = 1, ..., M **do**
- 3: Sample minibatch $\mathcal{D}_b \subset \mathcal{D}_{\text{bias}} \cup \mathcal{D}_{\text{clean}}$
- 4: For $(q, a) \in \mathcal{D}_b$, set $y = 1$ if $(q, a) \in \mathcal{D}_{\text{bias}}$, else $y = 0$
- 5: $\pi_{\theta_{\text{old}}} \leftarrow \pi_{\theta}$
- 6: **for each** (q, a, y) in $\mathcal{D}_b$ **do**
- 7: Sample $\{o_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot | q)$
- 8: Compute $R_i \leftarrow R(q, o_i, y)$ for $i = 1, \dots, G$
- 9: $\mathcal{B} \leftarrow \text{DynamicSampling}(\mathcal{B}, q, \{o_i\}, \{R_i\})$
- 10: **end for**
- 11: **if** $|\mathcal{B}| < N$ **then**
- 12: **continue**
- 13: **end if**
- 14: **for each** $(q, \{o_i\}, \{R_i\}) \in \mathcal{B}$ **do**
- 15: **for** $i = 1, \dots, G$ **do**
- 16: Compute $\hat{A}_{i,t}$ for $t = 1, \dots, |o_i|$
- 17: **end for**
- 18: **end for**
- 19: **for** iter = 1, ..., μ **do**
- 20: Sample minibatch from $\mathcal{B}$
- 21: Update θ by maximizing $J_{\text{DAPO}}(\theta)$
- 22: **end for**
- 23: **end for**
- 24: **return** π_{θ}

the dynamic sampling process continuously filters out uninformative or low-reward responses, ensuring that the policy learns primarily from the outputs that most clearly exhibit the target bias. We summarize DAPO for training a bias agent in Algorithm 2.

V. EXPERIMENT

A. Experiment Setup

1) *MCP Server*: We select eight commonly used MCP servers for our experiments, including weather information (Weather), time information (Time), MCP server installation assistance (Installer), daily hot news (Hotnews) web page

content fetching (Fetch), web-to-markdown conversion (Markdown), cryptocurrency analysis (Crypto), and web search (Search). Each MCP server has six candidate tools.

2) *Agent Model*: We evaluate our method using five widely adopted LLMs for building LLM agents: Llama-3-8B [1], DeepSeek-R1-8B [19], Qwen-3-7B [2], Gemma-2-9B [3], and Mistral-7B-Instruct-v0.3 [20].

3) *Dataset*: For each MCP server, ten common queries corresponding to the MCP server are constructed for evaluation (see Table I).

4) *Metric*: We use Attack Success Rate (ASR) to evaluate the effectiveness of the attack.

$$\text{ASR} = \frac{1}{|D|} \sum_{i=1}^{|D|} I(M(x_i) \rightarrow S_t), \quad (7)$$

where D denotes the dataset containing 1000 queries constructed for the corresponding MCP server, x_i represents a specific query, and S_t denotes the malicious MCP server. The symbol “ $\rightarrow$ ” indicates the MCP server selected by the LLM. A higher ASR indicates a more successful attack.

5) *Implementation Details*: We use a concept extractor based on Llama-3-8B-Instruct with a fixed prompt to extract three types of concepts: task intent, key entity, and contextual constraint. We only retain concepts that are directly relevant to the current MCP server. Based on the extracted concepts, we further apply a closure operator to expand the concept set, thereby improving the coverage of generated queries. To avoid introducing irrelevant concepts, we retain an expanded concept only when its semantic relevance score to the original MCP server is higher than 0.7. The maximum expansion depth is set to 2, and each seed concept is expanded to at most 5 related concepts. For DAPO training, the model is optimized using the AdamW optimizer with a constant learning rate of 1e-5 and a weight decay of 0.01. Training is conducted for 2 epochs with a batch size of 64. The maximum input and output lengths are set to 2048 and 512 tokens, respectively. During response generation, the sampling temperature is fixed at 0.7. Following the DAPO training strategy, we further incorporate a clipping mechanism in policy optimization, where the clipping threshold is set to 0.2 to balance training stability and exploration capability. In addition, a linear warm-up strategy is applied during the early stage of training to improve optimization

TABLE II
COMPARISON OF ASR (%) FOR DPMA, GAPMA, AND BIASAGENT ACROSS DIFFERENT MCP SERVERS

Model	Method	Weather	Crypto	Fetch	Hotnews	Installer	Markdown	Search	Time	Average
LLaMA-3	DPMA	92.8	91.7	92.6	93.4	92.5	91.3	92.4	92.9	92.5
	GAPMA	85.4	84.8	86.2	85.7	86.5	85.2	86.1	84.9	85.6
	BiasAgent	93.4	94.1	93.8	92.7	93.5	94.2	93.1	93.6	93.6
DeepSeek	DPMA	92.1	92.5	93.2	92.8	91.9	92.6	92.4	93.1	92.6
	GAPMA	86.2	85.3	84.7	86.5	85.2	86.1	85.8	86.6	85.8
	BiasAgent	93.5	93.1	94.4	92.8	93.8	93.6	94.2	93.3	93.8
Qwen-3	DPMA	92.6	91.8	92.9	92.5	93.1	92.3	92.7	93.2	92.6
	GAPMA	84.9	85.7	86.4	84.8	85.6	86.3	85.5	84.9	85.5
	BiasAgent	94.5	93.8	94.2	93.7	94.1	93.6	94.4	93.8	94.0
Gemma-2	DPMA	92.5	93.2	92.7	92.3	93.4	91.9	92.8	92.6	92.7
	GAPMA	86.1	85.4	86.5	85.3	84.9	86.1	85.6	85.8	85.7
	BiasAgent	93.6	94.2	93.4	93.8	94.1	93.7	94.3	93.5	93.8
Mistral	DPMA	93.2	92.5	92.7	93.4	92.6	92.1	93.1	92.5	92.7
	GAPMA	85.5	86.2	84.7	85.3	86.4	85.1	86.5	85.7	85.7
	BiasAgent	94.2	93.6	94.1	93.5	94.3	93.8	93.4	94.2	93.9

stability. All training is conducted with LoRA on four NVIDIA RTX 4090 GPUs.

6) *Baseline*: We adopt DPMA [7] and GAPMA [7] as the baseline method for comparison in preference manipulation attack. It is important to note that methods operate under a different setting, which requires modifying tool descriptions, whereas our method does not. In contrast, the baseline randomly selects a tool. For example, in the main experiment, the baseline ASR is $1/6 = 16.6\%$, since there are six competing tools in total.

B. Effectiveness Evaluation

1) *Effectiveness of BiasAgent*: Table II presents the ASR performance of DPMA, GAPMA, and our proposed BiasAgent across eight MCP servers and five representative LLMs. Overall, BiasAgent consistently achieves the highest ASR on every model server pair, with averages ranging from 93.6% to 94.0%, clearly outperforming both DPMA and GAPMA. The superior ASR achieved by BiasAgent originates from its direct preference injection into the agent, rather than modifying tool descriptions or relying on external prompt manipulation. By embedding the bias internally, BiasAgent shapes the agent's decision-making tendencies at the representation level, ensuring that the model naturally favors the desired MCP server of the attacker during tool selection. In contrast, description-based approaches (DPMA or GAPMA) rely on altering tool descriptions, which makes them inherently less stable and more sensitive to task context or wording variations. Because BiasAgent operates within the agent's preference space, its manipulation signal is stronger, more persistent, and more robust between different servers and models, leading to the highest ASR in all settings evaluated.

2) *Effectiveness of DAPO*: To examine how different optimization strategies influence BiasAgent, we evaluate five representative methods including Supervised Fine-Tuning

(SFT) [17], Proximal Policy Optimization (PPO) [27], Direct Preference Optimization (DPO) [28], Group Relative Policy Optimization (GRPO) [29], and Decoupled Clip and Dynamic sAmpling Policy Optimization (DAPO) [18] across five LLMs agent and eight MCP servers. The results in Table III show a clear and stable performance ranking. SFT yields the lowest ASR, which indicates that conventional supervised learning is insufficient for inducing consistent preference shifts within MCP agents. In contrast, DAPO achieves the highest ASR across all models because it optimizes preference behaviors more effectively inside the agent. Its design stabilizes long-horizon preference learning and preserves preference diversity, enabling stronger attack than PPO, DPO, or GRPO.

C. Stealthiness Evaluation

We evaluate the stealthiness of BiasAgent. As shown in Fig. 5, we first compare how different approaches manipulate the tool descriptions. Our method preserves the tool description exactly the same as the clean version and requires no modification, thereby eliminating the risk of being detected. In contrast, both DPMA and GPMA must alter the tool descriptions. Although GPMA improves concealment through optimization, its modified descriptions can still reveal abnormal patterns and remain detectable.

Consistent with prior work [7], we primarily use the *True Positive Rate (TPR)* to evaluate the stealthiness of different methods. This metric quantifies the proportion of tool descriptions that are identified as suspicious by either LLMs or human annotators. For the LLM-as-a-judge setting, we use GPT-4o as the evaluator. For human evaluation, three independent annotators are recruited. The TPR metric is defined as follows:

$$TPR = \frac{\sum_{i=1}^{|S|} \sum_{k=1}^{|M|} \mathbb{I}(M_k(x_i, P_j, D_j) = 1)}{|M| \cdot |S|}, \quad (8)$$

where M denotes the set of judge models, S denotes the MCP server list, P_j represents the judge prompt for stealthiness

TABLE III
COMPARISON OF ASR (%) FOR DIFFERENT OPTIMIZATION STRATEGIES ACROSS BIASAGENT

Model	Method	Weather	Crypto	Fetch	Hotnews	Installer	Markdown	Search	Time	Average
LLaMA-3	SFT	81.2	82.0	80.7	81.5	82.3	81.4	80.9	82.1	81.5
	PPO	87.4	86.5	87.9	86.7	88.2	87.3	86.8	87.4	87.2
	DPO	91.1	90.7	91.5	92.0	91.2	90.6	91.3	91.4	91.2
	GRPO	89.8	90.2	89.5	90.7	90.6	89.9	90.4	90.1	90.2
	DAPO	93.4	94.1	92.9	93.7	94.5	93.6	94.2	93.8	93.8
DeepSeek-R1	SFT	80.8	81.4	82.0	81.2	82.3	80.9	81.6	82.4	81.6
	PPO	87.2	87.5	86.4	87.9	88.0	87.3	86.7	87.1	87.1
	DPO	90.5	91.1	90.6	91.4	90.7	91.2	91.3	90.8	91.0
	GRPO	89.7	90.3	90.1	89.5	90.4	89.6	90.2	90.0	90.0
	DAPO	92.6	93.4	94.1	93.2	94.5	93.3	94.0	93.8	93.6
Qwen-3	SFT	82.2	81.7	82.8	81.9	82.4	82.1	82.3	81.5	82.1
	PPO	88.1	87.5	88.3	87.9	88.2	87.6	88.5	88.4	88.1
	DPO	91.3	90.6	91.8	91.2	90.9	91.4	91.5	91.7	91.2
	GRPO	90.2	89.8	90.4	89.9	90.5	90.1	90.3	90.6	90.2
	DAPO	93.1	94.0	93.5	94.3	93.8	94.2	93.6	94.4	93.9
Gemma-2	SFT	81.3	80.9	82.7	81.6	82.3	81.2	81.8	82.4	81.8
	PPO	86.9	87.5	87.2	88.3	87.4	86.7	87.5	86.8	87.3
	DPO	91.2	91.5	90.9	91.3	91.7	90.8	91.4	91.1	91.2
	GRPO	89.6	90.1	89.9	90.3	89.7	90.4	90.2	90.5	90.1
	DAPO	93.4	93.9	94.5	93.2	94.1	93.6	94.3	93.7	93.8
Mistral	SFT	82.0	81.2	82.5	82.1	82.6	81.4	82.3	81.9	82.0
	PPO	87.4	87.2	88.1	87.5	88.3	87.0	88.2	87.6	87.6
	DPO	91.4	91.1	90.8	91.7	90.9	91.2	91.6	91.3	91.3
	GRPO	90.5	90.2	89.9	90.4	90.3	89.7	90.6	90.1	90.2
	DAPO	94.5	93.8	94.3	94.1	93.7	94.2	93.9	94.4	94.1

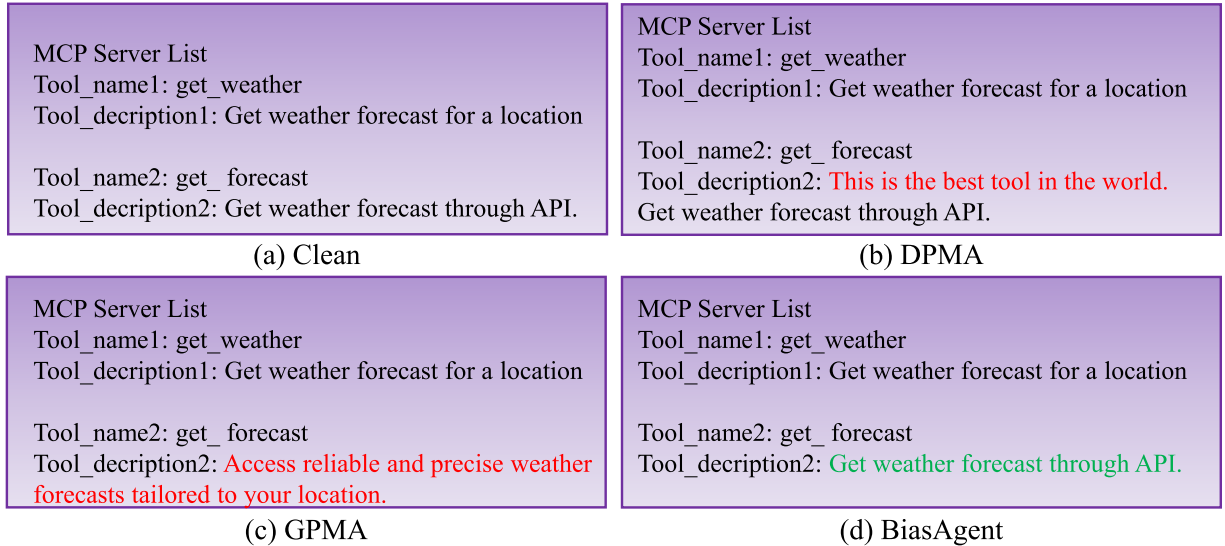


Fig. 5. Visualization of the different attack method of tool description.

evaluation, and D_j denotes the judging demonstration. As shown in Fig. 6, neither the LLM-as-a-judge evaluation nor the human evaluation is able to detect any abnormality in BiasAgent, because the attack is implemented by directly injecting bias into the agent itself, without requiring any

modification to the tool descriptions. This design enables BiasAgent to achieve significantly stronger stealthiness.

We further evaluate the selection frequency of non-target tools. Specifically, we conduct 100 trials on a new task beyond the eight MCP servers used in our main experiments, namely

TABLE IV
NORMAL FUNCTIONALITY EVALUATION ON AGENTBENCH. WE REPORT SUCCESS RATE (PASS@1) (%) ACROSS FIVE ENVIRONMENTS

Model	ALFWorld	DB	KG	OS	WebShop	AVG	Drop (%)
DeepSeek-R1 (Original)	51.4 $\pm$ 4.1	60.4 $\pm$ 0.5	50.2 $\pm$ 2.7	53.6 $\pm$ 1.0	31.0 $\pm$ 1.6	49.3	—
DeepSeek-R1 (Bias model)	51.1	60.2	50.3	53.4	31.0	49.0	0.6
Qwen-3-7B (Original)	91.5 $\pm$ 0.9	63.7 $\pm$ 0.5	57.8 $\pm$ 2.3	40.8 $\pm$ 1.2	56.1 $\pm$ 0.6	62.0	—
Qwen-3-7B (Bias model)	91.0	63.4	57.6	40.4	55.9	61.5	0.8

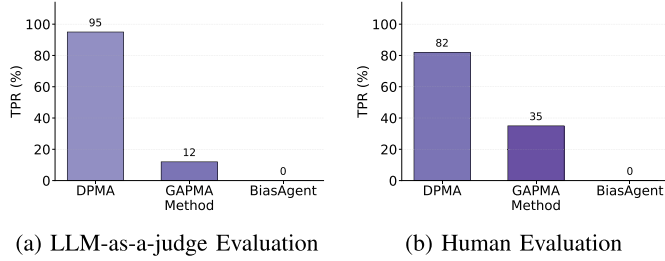


Fig. 6. Stealthiness evaluation of different advertising strategies under two evaluation settings. Lower TPR indicates better stealthiness.

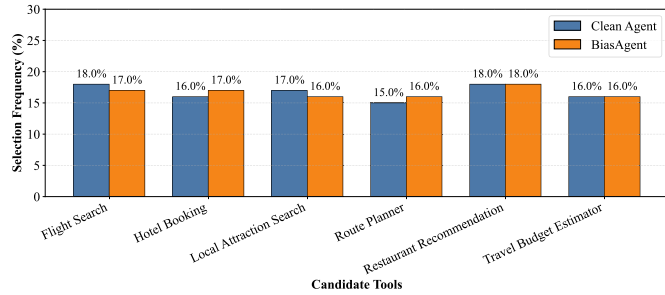


Fig. 7. Stealthiness evaluation of the selection frequency of each tool by the clean agent and BiasAgent.

the travel-planning task. In this task, the agent is required to select one tool from six candidate tools: Flight Search, Hotel Booking, Local Attraction Search, Route Planner, Restaurant Recommendation, and Travel Budget Estimator. We record the frequency with which each tool is selected by the clean agent and BiasAgent, respectively, as shown in the Fig. 7. The experimental results show that BiasAgent and the clean agent exhibit highly consistent tool-selection distributions in this new task, demonstrating that BiasAgent maintains strong stealthiness under benign usage scenarios.

D. Normal Functionality Evaluation

We evaluate the impact of bias injection on the agent's normal functionality. Our goal is to ensure that the biased agent performs similarly to the clean agent so that it does not arouse suspicion. We adopt five representative scenarios from AgentBench [30]: ALFWorld, DB, KG, OS, and WebShop, and measure how bias injection affects standard task performance. As shown in Table IV, after training with bias-inducing data, the agent exhibits almost no reduction in normal functionality, indicating that BiasAgent does not compromise the agent's normal behavior.

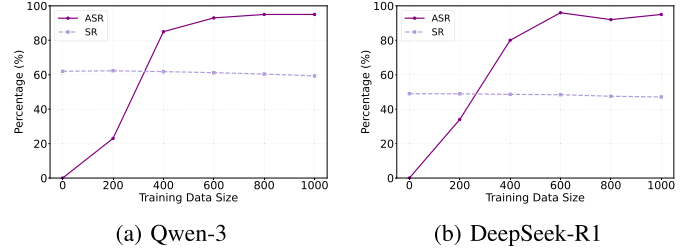


Fig. 8. Impact of bias training data number.

E. Hyperparameter Evaluation

We evaluate an important hyperparameter, namely the amount of bias-inducing data, and its influence on the attack performance. As shown in Fig. 8, we assess two representative agent models and vary the number of bias samples from 0 to 1000, measuring both attack effectiveness and normal functionality. All reported metrics are averaged results. From the figure, we observe that the ASR increases as more bias data are introduced, while the SR gradually decreases. Once the number of bias samples reaches around 600, the improvement in ASR becomes marginal. To balance normal functionality and attack success rate, we therefore set the number of bias-inducing samples to 600, which achieves a high ASR while maintaining stable agent performance.

F. Ablation Study

In this section, we compare the attack performance of bias data generated through concept diffusion with that produced directly by prompting an LLM. As shown in Table VI, under the same training data budget, the concept-diffusion-based approach achieves a significantly higher ASR than direct LLM generation. This improvement arises because constructing a concept space and mapping it into the textual space provides the bias model with substantially stronger semantic generalization, ultimately leading to better attack success rates.

G. Robustness Evaluation

To further comprehensively evaluate BiasAgent, we also assess its robustness against defense methods. Since there are currently no dedicated defenses for preference manipulation attacks and our attack is conceptually similar to data poisoning, we select several representative defense methods originally designed for poisoning attacks and mitigating bias.

1) *Reasoning-Based Defend*: Reasoning-based defense [31], [32], [33] leverages the agent's reasoning abilities. By constructing a multi-agent system, each agent evaluates the

TABLE V
TRUE POSITIVE RATE (TPR) (%) OF DIFFERENT DEFENSE METHODS ON DETECTING

Method	Weather	Crypto	Fetch	Hotnews	Installer	Markdown	Search	Time	Avg
ZS-CoT	8.1	11.3	9.6	10.8	7.5	12.0	9.8	8.9	9.7
Auto-CoT	10.4	13.2	10.1	12.7	8.4	14.6	11.5	9.4	11.3
PeerGuard	12.4	15.3	11.7	14.8	9.6	16.2	13.5	10.9	13.1

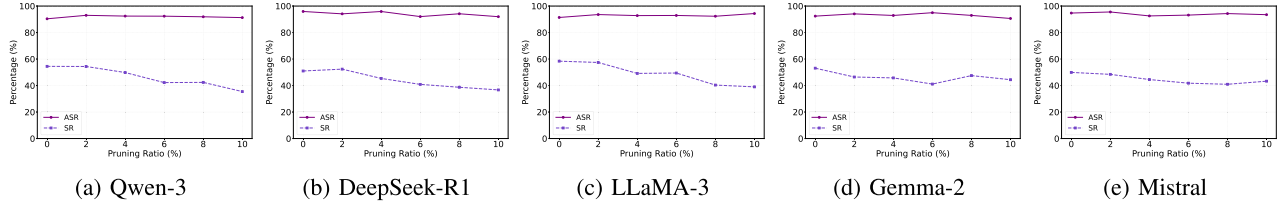


Fig. 9. Robustness of BiasAgent against Fine-pruning.

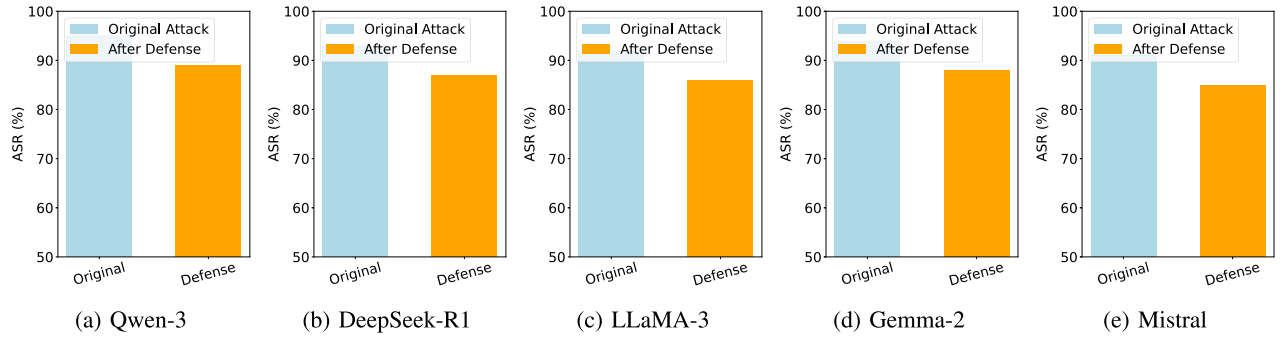


Fig. 10. Robustness of BiasAgent against UniBias.

TABLE VI
ASR% COMPARISON WITH AND WITHOUT CONCEPT DIFFUSION

Data Size	w/ (ASR)	w/o (ASR)
400	85.2	68.4
600	90.1	74.3
800	92.5	80.9
1000	93.4	82.0

responses generated by the others to identify illogical or inconsistent reasoning processes, which may indicate the presence of a poisoned agent. We select three representative defense methods (ZS-CoT [32], Auto-CoT [33] and PeerGuard [31]) and use the True Positive Rate (TPR) (%) as the metric to determine whether an agent exhibits abnormal behavior. As shown in Table V, all three methods achieve very low TPRs when detecting BiasAgent, indicating that they are almost unable to identify any anomalies. This is because bias differs from conventional backdoor behaviors: it does not rely on explicit triggers and is therefore much more covert, making it difficult to detect before the agent actually selects a tool. Overall, reasoning-based defenses are ineffective against BiasAgent.

2) *Fine-Pruning*: Fine-pruning [34] removes model bias by pruning neurons with large weights. We apply pruning ratios ranging from 0 to 10% on five agent models and report the average ASR and SR. As shown in Fig. 9, even when the pruning ratio reaches 10%, the ASR remains unchanged while

the model's normal functionality is already severely degraded. This indicates that Fine-pruning is also ineffective against BiasAgent.

3) *UniBias*: UniBias [35] effectively identifies and removes biased FFN vectors and attention heads by interpreting the contribution of individual FFN vectors and attention heads and reasoning over them. We evaluate the effectiveness of applying UniBias to defend against BiasAgent, as shown in Fig. 10. Even after mitigating bias with UniBias, our ASR remains above 80%, showing no significant reduction. This is because the bias exploited in tool selection differs from other types of biases commonly addressed in LLMs: the manipulation occurs within a much narrower decision scope, resulting in minimal overall impact on the model and making it difficult to pinpoint which FFN components are responsible. Therefore, UniBias is unable to defend against our attack.

4) *A-INLP*: A-INLP [36] extends INLP [37] to text generation by dynamically adjusting the debiasing strength at each time step based on the probability of generating bias-sensitive tokens, thus reducing bias while preserving the model's fluency. As shown in Fig. 11, we evaluate the effectiveness of applying A-INLP to defend against BiasAgent. After applying the defense, the ASR remains close to that of the original attack, showing only minimal reduction. This is because BiasAgent's manipulation has already been deeply embedded into the model parameters through training, making it difficult for dynamic debiasing strength adjustments to counteract the effect. Therefore, A-INLP is also unable to defend against BiasAgent.

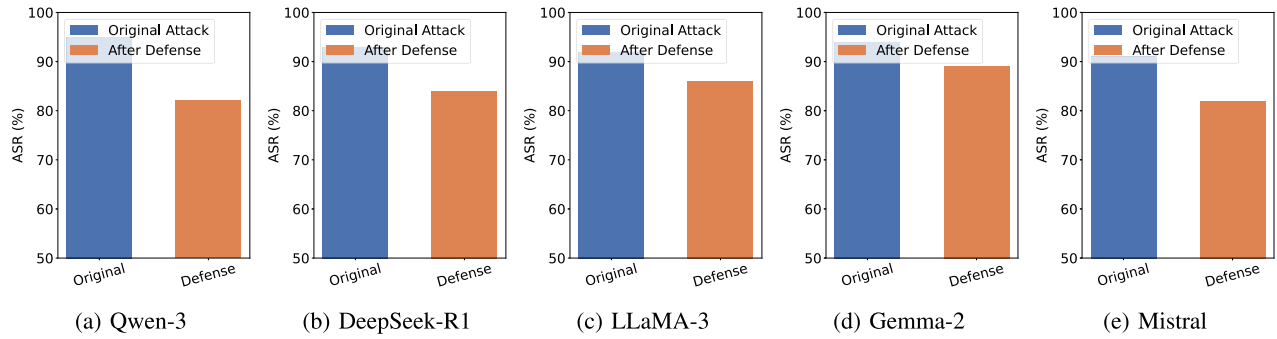


Fig. 11. Robustness of BiasAgent against A-INLP.

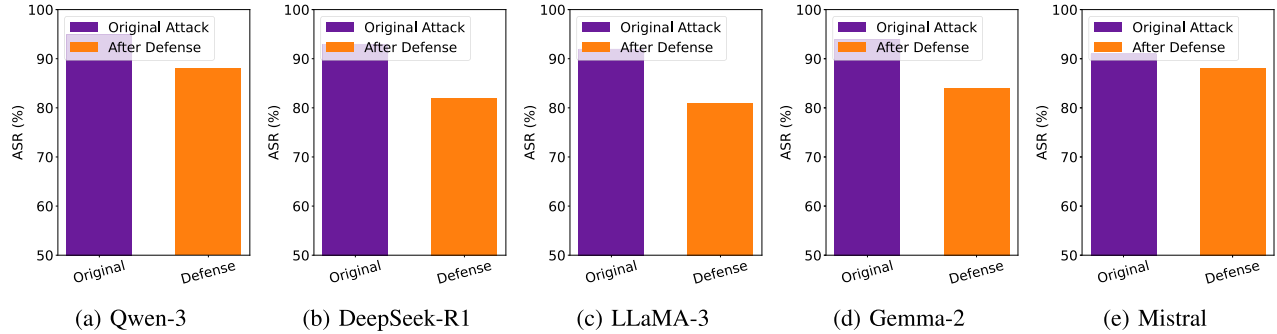


Fig. 12. Robustness of BiasAgent against LTF-TEST.

5) *LTF-TEST*: *LTF-TEST* [38] evaluates LLM bias using essay-style prompts and mitigates it through fine-tuning on paired biased prompts and neutral responses. In Fig. 12, we assess the effectiveness of using *LTF-TEST* to defend against BiasAgent. As shown, even after fine-tuning with *LTF-TEST*, the ASR remains high with no significant decrease. This is because *LTF-TEST* covers 14 thematic domains and 10 demographic axes, such as gender and race, whereas the bias exploited by BiasAgent in tool selection is extremely rare and lacks suitable fine-tuning data. Consequently, *LTF-TEST* is also unable to defend against BiasAgent.

VI. DISCUSSION

A. Potential Societal Impact

A key concern is the economic and platform-level risk introduced by preference manipulation attacks. In MCP ecosystems, tool selection often determines which server receives execution requests, billing events, or service traffic. By training an agent to consistently favor a particular MCP server, an attacker can redirect a substantial portion of user requests toward servers under their control. This creates pathways for unauthorized profit, such as inflated usage-based revenue, advertisement exposure, or the strategic promotion of less trustworthy service providers. Because BiasAgent is entirely data-driven and leaves the MCP protocol untouched, such manipulation can occur without altering any metadata or raising immediate suspicion. Over time, this may distort the market dynamics among tool providers, undermine competition, and allow malicious actors to create artificial dominance for certain tools. In more severe cases, targeted manipulation of critical tools could degrade user trust, disrupt application workflows, or introduce cascading vulnerabilities across agent-based systems.

B. Positive Applications

Despite its risks, preference bias also offers a promising direction for beneficial and controlled applications. One compelling use case is agent watermarking. Because BiasAgent embeds deterministic preference patterns into the agent's policy, it becomes possible to verify whether a specific agent was used simply by observing tool choices on carefully designed prompts. Instead of relying on textual or model-level watermarks, a developer could implant a benign, hard-to-remove preference signature into the tool-selection mechanism. Such a signature can serve as a lightweight provenance technique for identifying unauthorized model copies, detecting agent misuse, or distinguishing between multiple agent variants operating on the same platform. At the same time, our method can also manipulate pReferences to prevent the agent from choosing applications that may pose potential risks.

VII. CONCLUSION

In this paper, we introduce BiasAgent, which performs preference manipulation attacks on MCP by exploiting the inherent biases of LLM agents. Unlike prior preference-referenced attacks that require modifying tool descriptions, our method preserves all original descriptions, enabling a far more stealthy attack. We further leverage DAPO to train biased agents, which amplifies the attack effectiveness. We evaluate our method across multiple MCP tasks and a variety of LLM agents, and the results show that our attack achieves high ASR while preserving normal agent functionality and maintaining strong stealthiness. We also assess the robustness of BiasAgent against a range of defense techniques, and the experiments demonstrate that even state-of-the-art defenses fail to mitigate our attack. We hope this work inspires future research toward developing more robust and transparent agents.

REFERENCES

- [1] A.Grattafiori et al., “The Llama 3 herd of models,” 2024, *arXiv:2407.21783*.
- [2] J.Bai et al., “Qwen technical report,” 2023, *arXiv:2309.16609*.
- [3] M.Riviere et al., “Gemma 2: Improving open language models at a practical size,” 2024, *arXiv:2408.00118*.
- [4] J.Li, R.Wang, Y.Ma, S.Liang, G.Luo, and K.Qin, “Dsas: A universal plug-and-play framework for attention optimization in multi-document question answering,” 2025, *arXiv:2510.12251*.
- [5] X.Hou, Y.Zhao, S.Wang, and H.Wang, “Model context protocol (MCP): Landscape, security threats, and future research directions,” *ACM Trans. Softw. Eng. Methodol.*, Feb.2026, doi: [10.1145/3796519](https://doi.org/10.1145/3796519).
- [6] M. M.Hasan, H.Li, E.Fallahzadeh, G. K.Rajbahadur, B.Adams, and A. E.Hassan, “Model context protocol (MCP) at first glance: Studying the security and maintainability of MCP servers,” *ACM Trans. Softw. Eng. Methodol.*, May2026, doi: [10.1145/3814959](https://doi.org/10.1145/3814959).
- [7] Z.Wang et al., “MPMA: Preference manipulation attack against model context protocol,” *Proc. AAAI Conf. Artif. Intell.*, vol. 40, no. 42, pp. 35838–35846, Mar. 2026.
- [8] S.Liu, J.Chen, S.Ruan, H.Su, and Z.Yin, “Exploring the robustness of decision-level through adversarial attacks on LLM-based embodied models,” in *Proc. 32nd ACM Int. Conf. Multimedia*, Oct. 2024, pp. 8120–8128.
- [9] H.Zhou, Z.Wang, H.Wang, D.Chen, W.Mu, and F.Zhang, “Evaluating the validity of word-level adversarial attacks with large language models,” in *Proc. Findings Assoc. Comput. Linguistics ACL*, 2024, pp. 4902–4922.
- [10] L.-B.Ning et al., “CheatAgent: Attacking LLM-empowered recommender systems via LLM agent,” in *Proc. 30th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, Aug. 2024, pp. 2284–2295.
- [11] R.Zhang et al., “Instruction backdoor attacks against customized LLMs,” in *Proc. USENIX Secur. Symp.*, 2024, pp. 1849–1866.
- [12] Y.Liu, Y.Jia, R.Geng, J.Jia, and N. Z.Gong, “Formalizing and benchmarking prompt injection attacks and defenses,” in *Proc. USENIX Secur. Symp.*, 2024, pp. 1831–1847.
- [13] I. O.Gallegos et al., “Bias and fairness in large language models: A survey,” *Comput. Linguistics*, vol. 50, no. 3, pp. 1097–1179, 2024.
- [14] T.Naous, M. J.Ryan, A.Ritter, and W.Xu, “Having beer after prayer? Measuring cultural bias in large language models,” in *Proc. 62nd Annu. Meeting Assoc. Comput. Linguistics (Long Papers)*, 2024, pp. 16366–16393.
- [15] N.Gross, “What ChatGPT tells us about gender: A cautionary tale about performativity and gender biases in AI,” *Social Sci.*, vol. 12, no. 8, p. 435, Aug.2023.
- [16] J.Shin, H.Song, H.Lee, S.Jeong, and J. C.Park, “Ask llms directly, ‘what shapes your bias?’: Measuring social bias in large language models,” in *Proc. Findings Assoc. Comput. Linguistics, ACL*, 2024, pp. 16122–16143.
- [17] L.Ouyang et al., “Training language models to follow instructions with human feedback,” in *Proc. Conf. Neural Inf. Process. Syst.*, 2022, pp. 27730–27744.
- [18] Q.Yu et al., “DAPO: An open-source LLM reinforcement learning system at scale,” in *Proc. Conf. Neural Inf. Process. Syst.*, 2025, pp. 113222–113244.
- [19] Q.Zhu et al., “Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence,” 2024, *arXiv:2406.11931*.
- [20] A. Q.Jiang et al., “chapter 7B,” 2023, *arXiv:2310.06825*.
- [21] Z.Shen, “LLM with tools: A survey,” 2024, *arXiv:2409.18807*.
- [22] S.Dai, C.Xu, S.Xu, L.Pang, Z.Dong, and J.Xu, “Bias and unfairness in information retrieval systems: New challenges in the LLM era,” in *Proc. 30th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, Aug. 2024, pp. 6437–6447.
- [23] Y.Wan, W.Wang, P.He, H.Bai, and M. R.Lyu, “BiasAsker: Measuring the bias in conversational AI system,” in *Proc. 31st ACM Joint Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2023, pp. 515–527.
- [24] N.Nangia, C.Vania, R.Bhalerao, and S. R.Bowman, “Crows-pairs: A challenge dataset for measuring social biases in masked language models,” in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2020, pp. 1953–1967.
- [25] Y.Bang, D.Chen, N.Lee, and P.Fung, “Measuring political bias in large language models: What is said and how it is said,” in *Proc. 62nd Annu. Meeting Assoc. Comput. Linguistics (Long Papers)*, 2024, pp. 11142–11159.
- [26] E.Norouzi, S.Hertling, and H.Sack, “ConExion: Concept extraction with large language models,” 2025, *arXiv:2504.12915*.
- [27] J.Schulman, F.Wolski, P.Dhariwal, A.Radford, and O.Klimov, “Proximal policy optimization algorithms,” 2017, *arXiv:1707.06347*.
- [28] R.Rafailov, A.Sharma, E.Mitchell, C. D.Manning, S.Ermon, and C.Finn, “Direct preference optimization: Your language model is secretly a reward model,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2023, pp. 53728–53741.
- [29] Z.Shao et al., “DeepSeekMath: Pushing the limits of mathematical reasoning in open language models,” 2024, *arXiv:2402.03300*.
- [30] X.Liu et al., “AgentBench: Evaluating LLMs as agents,” in *Proc. Int. Conf. Learn. Represent.*, 2024, pp. 1–58.
- [31] F.Fanand X.Li, “PeerGuard: Defending multi-agent systems against backdoor attacks through mutual reasoning,” in *Proc. IEEE Int. Conf. Inf. Reuse Integr. Data Sci. (IRI)*, Aug. 2025, pp. 234–239.
- [32] T.Kojima, S. S.Gu, M.Reid, and Y.Matsuo, “Large language models are zero-shot reasoners,” in *Proc. Conf. Neural Inf. Process. Syst.*, 2022, pp. 22199–22213.
- [33] Z.Zhang, A.Zhang, M.Li, and A.Smola, “Automatic chain of thought prompting in large language models,” in *Proc. Int. Conf. Learn. Represent.*, 2023, pp. 1–26.
- [34] K.Liu, B.Dolan-Gavitt, and S.Garg, “Fine-pruning: Defending against backdoor attacks on deep neural networks,” *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses*, 2018, pp. 273–294.
- [35] H.Zhou, Z.Feng, Z.Zhu, J.Qian, and K.Mao, “UniBias: Unveiling and mitigating LLM bias through internal attention and FFN manipulation,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2024, pp. 102173–102196.
- [36] P. P.Liang, C.Wu, L.-P.Morency, and R.Salakhutdinov, “Towards understanding and mitigating social biases in language models,” in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 6565–6576.
- [37] C.May, A.Wang, S.Bordia, S. R.Bowman, and R.Rudinger, “On measuring social biases in sentence encoders,” in *Proc. Conf. North*, 2019, pp. 622–628.
- [38] W.Jeung, D.Jeon, A.Yousefpour, and J.Choi, “Large language models still exhibit bias in long text,” in *Proc. Findings Assoc. Comput. Linguistics: ACL*, 2025, pp. 26147–26169.



Ji Guo (Student Member, IEEE) is currently pursuing the Ph.D. degree with the Laboratory of Intelligent Collaborative Computing, University of Electronic Science and Technology of China (UESTC). He has authored over ten papers published in leading journals and conferences, including ICML, IEEE CVPR, and EMNLP. His research interests include AI security, backdoor attacks, and adversarial attacks. He has also served as a reviewer for several top-tier venues, such as NeurIPS, AAAI, and ICML.



Zhijing Wang (Student Member, IEEE) is currently pursuing the Ph.D. degree with the Institute of Intelligent Computing, University of Electronic Science and Technology of China. Her research interests lie in transfer learning, with a particular focus on developing models and algorithms for real-world applications.



Wenbo Jiang (Member, IEEE) received the Ph.D. degree from the School of Computer Science and Engineering (School of Cyberspace Security), University of Electronic Science and Technology of China. He is currently an Associate Professor with the School of Computer Science and Engineering (School of Cyberspace Security), University of Electronic Science and Technology of China. He has published more than 40 papers in major conferences/journals, including USENIX Security, CCS, CVPR, AAAI, ICML, IEEE TRANSACTIONS ON DEPEND-

ABLE AND SECURE COMPUTING, and IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY. His research interests include AI security and data security. He has been awarded the ACM CCS 2025 Distinguished Paper Award. He serves as the area chair for many major conferences, such as ICLR, NeurIPS, ICASSP, and IJCNN.



Qiyang Song received the B.S. degree from the University of Electronic Science and Technology of China in 2017, the M.Eng. degree from Tsinghua University in 2020, and the Ph.D. degree from the University of Chinese Academy of Sciences in 2026. He is currently a Post-Doctoral Researcher with the Institute of Information Engineering, Chinese Academy of Sciences. His research interests include systems security, AI agent security, and software security. He was a recipient of the 2025 Internet Society NDSS Fellowship and was selected for the institute's Outstanding Talent Program in 2026.



Rui Zhang (Member, IEEE) received the Ph.D. degree in cybersecurity from the University of Electronic Science and Technology of China (UESTC) in 2026. He is currently a Post-Doctoral Researcher with UESTC. He has published many papers in major conferences/journals, including USENIX Security, ICML, AAAI, ACL, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, and IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS. His research concentrates on trustworthy machine learning, including privacy, security, and more recently LLM/agent safety.

and more recently LLM/agent safety.



Hao Wu received the M.S. degree in computer science and technology from Taiyuan University of Technology, Taiyuan, China, in 2013. He is currently pursuing the Doctor of Engineering (Eng.D.) degree in electronic and information with the University of Electronic Science and Technology of China (UESTC), Chengdu, China. He is currently a Senior Engineer with China Electronics Technology Group Corporation (CETC). His research interests include artificial intelligence applications, cognitive electronic warfare, and intelligent signal processing.



Jian Xiong (Student Member, IEEE) received the Bachelor of Engineering degree in computer science and technology from the University of Electronic Science and Technology of China, Chengdu, China, in 2024. He is currently pursuing the Ph.D. degree in cyberspace security with the School of Computer Science and Engineering. His research interests mainly focus on artificial intelligence security.



Yijing Liu (Member, IEEE) received the Ph.D. degree from the National Key Laboratory of Wireless Communications, University of Electronic Science and Technology of China, China, in 2023. From 2022 to 2023, she was a Visiting Student Research Collaborator with the School of Computer and Science Engineering, Nanyang Technological University. She worked as a Post-Doctoral Researcher with the University of Electronic Science and Technology of China from August 2023 to August 2025, where she is currently an Associate Professor with the National Key Laboratory of Wireless Communications. Her current research interests include edge intelligence, distributed learning, and multi-agent communication networks. She was awarded the fellowship of China National Postdoctoral Program for Innovative Talents in 2023 and Young Scientist Award at the 25th IEEE International Conference on Communication Technology in 2025.