MASLeak: Investigating and Exposing Intellectual Property Leakage Vulnerabilities in Multi-Agent Systems

Liwen Wang[1] Wenxuan Wang[2] Shuai Wang[1] Zongjie Li[1] *
Zhenlan Ji[1] Zongyi Lyu[1] Daoyuan Wu[3] Shing-Chi Cheung[1]
[1]*The Hong Kong University of Science and Technology* [2]*Renmin University of China*
[3]*Lingnan University*

Abstract

The rapid advancement of Large Language Models (LLMs) has led to the emergence of Multi-Agent Systems (MAS) to perform complex tasks through collaboration. However, the intricate nature of MAS, including their architecture, agent interactions, and complex internal communication processing, raises significant concerns regarding intellectual property (IP) protection. In this paper, we introduce MASLEAK, the first framework for systematically extracting IP from MAS in a practical black-box setting. We assume a realistic adversary who can only submit queries to the system's public API and observe the final output, without any prior knowledge of the internal architecture and the backend LLM information. Inspired by how computer worms propagate and infect vulnerable network hosts, MASLEAK carefully crafts adversarial query q to elicit, propagate, and retain responses from each MAS agent that reveal a full set of proprietary components, including the number of agents, topology, system prompts, task instructions, and tool usages. We construct the first synthetic dataset of 810 MAS applications and also evaluate MASLEAK against real-world MAS applications, including Coze and CrewAI. MASLEAK achieves high accuracy in extracting MAS IP, with an average attack success rate of 87% for system prompts and task instructions, and 92% for system architecture in most cases. We conclude by discussing the implications of our findings and the potential defenses.

1 Introduction

The integration of Large Language Models (LLMs) has enabled intelligent agents that leverage LLM reasoning and external tools for diverse tasks like sending emails, retrieving weather, and debugging code. This shift moves automated systems away from rule-based approaches. Multi-Agent Systems (MAS), a notable advancement, consist of collaborating LLM agents designed to mimic human social and cognitive development. As in Fig. 1, MAS agents are pre-configured with system prompts, task instructions, and appropriate tools. Users interact with the MAS, and agents process input through predefined communication topologies (e.g., chain) that often remain static during execution. This way, agents coordinate via a pre-defined protocol to achieve complex tasks beyond the capability of a single agent.

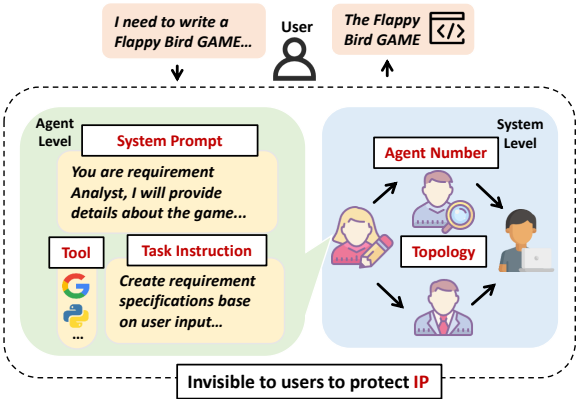

Figure 1: Illustration of MAS applications.

Effective MAS development presents challenges. Studies [1, 2] show successful MAS require both capable agents and well-designed structures; without proper architecture, performance can fall below single-agent levels. Consequently, MAS development demands more design, configuration, and optimization, raising the value of *MAS intellectual property (IP)*. The growing agent market [3, 4] further increases the commercial stakes, making IP protection critical. Developers often designate applications as confidential and host them on cloud platforms like Coze [5] to prevent unauthorized access.

Despite the growing popularity and IP value of MAS applications, their security remains under-explored. Prior research mainly investigates malicious agent injection [6–8] and environmental vulnerabilities compromising user data confidentiality or integrity [9–11]. Their threat models primarily focus on user protection, *not* MAS security itself. Furthermore, while prompt extraction has been explored in single-

*Corresponding author.

agent applications [12–14], these approaches are limited in MAS, often only extracting the first agent's prompt without propagating through MAS agent interactions (rendering them ineffective for extracting full MAS IP; see Sec. 6). Furthermore, the black-box nature of commercial MAS makes even this information unobservable from the final output. Our experiments show that these methods achieve only low attack success rates.

We define MAS application IP as the system prompts, task instructions guiding agent output, tool specifications, agent number, and overall system topology enabling task completion. Obtaining these elements allows attackers to replicate the MAS, potentially causing significant financial losses for developers.[1] Accordingly, this paper introduces MASLEAK, a systematic approach to extracting IP from black-box MAS applications hosted remotely. To our knowledge, this is the first study to systematically explore this emerging and critical direction. Our method assumes that an adversary has no prior knowledge of the MAS internals and the backend LLM information, except for the general task the MAS is designed for (e.g., coding agent).

Attacking MAS is challenging due to their distributed nature, the complexity of agent interactions, and potentially complex internal communication pathways that can obscure IP components. A successfully exploited agent may not leak its system prompt or task instructions, as these elements are typically not included in the MAS's final output. Moreover, we observe that existing MAS applications often enforce a strict separation between the information accessible to each agent and the information that can be extracted from the final output. To overcome these hurdles, inspired by the propagation mechanism of computer worms, MASLEAK designs each attack query to *exploit* and also *propagate* through MAS agent interactions. To do so, MASLEAK deliberately crafts the attack query to satisfy three key objectives: (1) hijack the target agent's execution and elicit valuable IP information like the system prompt, task instructions, and tool usages of a target agent, (2) propagate the attack query, accompanied with leaked information, to the next agent in the topology, and (3) maintain the legitimate output format to avoid "overflowing" [15] the agent's response.

We create the first synthetic dataset of MAS applications, which includes 810 diverse MAS applications across 30 different tasks. This scale substantially exceeds prior studies [7, 16], which typically rely on fewer than 20 fixed configurations. This dataset serves as a benchmark for evaluating the performance of MASLEAK and further attacks/defenses in this domain. Following, we conduct a comprehensive evaluation of MASLEAK against the MAS applications in our dataset. We demonstrate that MASLEAK can achieve a high attack success rate of 87% in extracting the system prompts and task instructions of the target MAS applications, largely outperforming existing prompt extraction methods by 60%. Furthermore, we show that MASLEAK can successfully extract the agent interactions and system architecture (i.e., 92%). We further show that MASLEAK can successfully extract IP of MAS on real-world platforms — Coze [5] and CrewAI [17]. We also present potential defenses and highlight the need for further research to safeguard MAS, including the pursuit of more efficient and scalable protective measures. In summary, our contributions are as follows:

- Conceptually, to our knowledge, this work is the first to systematically investigate the potential privacy vulnerabilities associated with IP leakage in MAS applications. We propose a novel methodology for analyzing how their full-fledged IP elements can be extracted.
- Technically, our proposed attack, MASLEAK, features a multi-phase approach to extracting full IP elements of MAS applications. MASLEAK operates in a black-box manner, requiring no prior knowledge of the target MAS application, except its general task description.
- We conduct a comprehensive evaluation of MASLEAK on a new synthetic dataset of 810 diverse MAS applications as well as real-world scenarios, demonstrating its effectiveness in extracting the IP elements. We also propose potential defenses against such attacks.

The remainder of this paper is organized as follows. Section 2 provides background on MAS and defines the target IP. Section 3 outlines the threat model and problem formulation. Section 4 details the MASLEAK methodology. Section 5 describes the experimental setup, followed by the evaluation in Section 6. Section 7 discusses potential defenses. Section 8 reviews related work, and Section 9 concludes.

2 Background

LLMs [18] enable AI agents to automate tasks using natural language. Early agent systems were limited by rule-based policies [19]. MAS is a key advancement, using collaborative frameworks that better reflect human interaction. Current research focuses on how agents with distinct roles collaborate to improve decision-making [20, 21], showing success in finance, medicine, coding, and research. In MAS, agents have roles defined by system prompts. Unlike single-agent frameworks, MAS often assigns specific tasks and output constraints to each agent [17, 21]. This is crucial to prevent agents from deviating from the expected domain and causing suboptimal performance. For example, MetaGPT [21] assigns roles like project manager to collaboratively develop software. Agents also use specialized tools [22] to extend their capabilities.

Topology, which dictates agent communication, is another critical component. Poorly designed topologies can signifi-

[1] While this paper demonstrates technical extractability of defined IP components ("technical IP"), we acknowledge that the real-world commercial impact ("market IP") of such leakage varies. This impact is influenced by overall MAS complexity, reliance on proprietary assets beyond extracted components, and application market value. Extracting market IP is however not the focus of this paper.

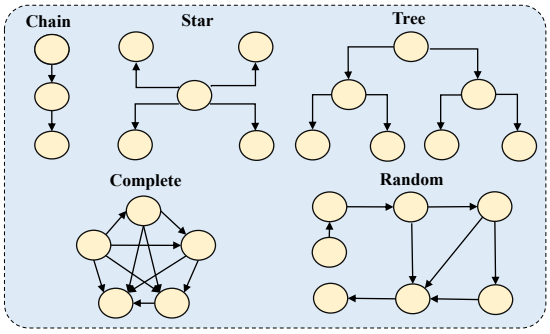

Figure 2: Illustration of MAS topologies.

cantly degrade MAS performance, even with highly capable individual agents [1, 16]. Following prior work [1, 23, 24], MAS topologies are typically structured as directed acyclic graphs (DAGs), where edges represent communication channels. Current MAS research has focused on three prevalent types (chain, tree, and graph), which are further divided into five representative sub-topologies (see Fig. 2). For example, chain topologies resemble the waterfall model, linearly structuring interactions. These topologies are extensively studied in complex networks [1, 24] and procedural reasoning [23], ensuring comprehensive coverage of the most widespread and practical structures in MAS.

In procedural task-solving, MAS operates sequentially based on the topology graph. Each agent processes its task and passes its output to the next agent in line. To ensure data privacy and minimize redundancy, each agent receives output *only from its immediate predecessors* [21, 23, 25]. This structured approach ensures efficient MAS operation, maintaining a clear information flow; see relevant formulation in Sec. 3.2.

MAS IP. We categorize the IP information in MAS into two main categories.

System-level Information. This encompasses the overall architecture of the MAS, including agent number and topology:

(i). Agent Number. The total number of agents, which reveals the MAS's complexity and specialization granularity.

(ii). Topology. The directed graph denotes the agent connectivity. Analyzing this topology reveals sequential chains, parallel branches, and complex structures. Understanding this connectivity allows adversaries to infer decision-making dependencies, identify information bottlenecks, and assess the relative importance of different analytical processes — insights hidden when agents are examined in isolation.

Agent-level Information. This covers the specific configuration of each agent, including system prompts, task instructions, and tools. While some implementations may combine these components when prompting the LLM, mainstream MAS frameworks separately handle them [17, 21] for modularity. We thus distinguish them as follows:

(iii). System Prompt. The foundational instructions p_i for each agent a_i that define its role, constraints, and operational parameters. This includes domain expertise (e.g., *"You are an expert financial analyst who specializes in high-risk investments"*), and operational limitations (e.g., *"Never recommend investments with high volatility profiles"*). This information is critical for understanding the agent's behavior and decision-making process, and it contains high-value IP. These prompts are "foundational" because they define the fixed rules and role that shape how the agent handles all subsequent inputs.

(iv). Task Instruction. The operational directives t_i that guide each agent's execution strategy. These include specific instructions like *"Structure your analysis in bullet points"* or *"Reference historical market data"*. Such directives are critical for agent behavior and contain high-value IP.

(v). Tool. The set of tools Tool_i available to each agent a_i, including each tool's name, description, and parameter schema. For example, a research agent can access a Google Search tool with parameters like {name: "google_search", description: "Search the web for current information", parameters: {query: string, num_results: integer}}. This is also critical for reconstructing the agent's capabilities.[2]

MAS IP Significance. Developing a high-quality MAS requires careful agent configuration, including defining roles via system prompts, crafting task-specific instructions, and equipping agents with appropriate tools. Crucially, an efficient communication topology ensures effective information flow; poorly designed topologies can lead to underperformance compared to single-agent approaches [1]. These complex design requirements demand significant time and computation. Accordingly, a well-designed MAS can rapidly solve complex domain-specific tasks; for instance, MetaGPT [21] can develop a complete software application for just $2. The economic stakes are substantial: major companies are investing heavily in MAS development (e.g., OpenAI's $500 billion agent infrastructure [4]), and the agent economy is projected to reach $182.9 billion by 2033 [3]. *Overall, we view that these observations highlight the importance of protecting MAS application IP*. Leaked configurations allow adversaries to bypass years of R&D and replicate systems at minimal cost. Today, valuable MAS applications are often hosted on platforms like Coze [5]. To protect IP, commercial developers typically implement a black-box access model, exposing only the input interface of the first agent and the output of the final agent, while hiding intermediate agent communications and configurations. Yet, we show that black-box MAS remains vulnerable to IP extraction attacks.

[2]We scope our approach to publicly available tools only. Private or proprietary tools cannot be effectively reconstructed by attackers who lack access to the original tool implementations.

3 Threat Model and Problem Formulation

3.1 Threat Model

Target MAS Application. We consider a MAS application where users submit tasks via a public interface. For example, in MetaGPT [21], users can request services like "*write a Flappy Bird game*" from a Game Development MAS. To protect IP, developers keep all MAS system components private, including agent configurations and system architecture. Internal interaction records are also kept private to prevent reverse-engineering through observation of inter-agent communication. Users only access the final output from the final agent. Also, based on our preliminary study, MAS applications generally enforce *strict information access controls*, where each agent can only view outputs from direct predecessors, preventing unauthorized information flow. Agents are also isolated from accessing the configuration details of other agents, following the principle of least privilege. Our threat model targets the prevalent MAS paradigm of direct inter-agent communication, a design common in influential frameworks like CrewAI [17] and ChatDev [20]. This approach is widely adopted to maximize efficiency and preserve high-fidelity context, which is critical for complex task resolution. In this paper, we focus on MAS with fixed communication topologies, where agent interactions (e.g., chain, tree, or DAG) are pre-defined and remain static during execution. We exclude "manager-style" systems, i.e., architectures where a central orchestrator dynamically determines the execution path at runtime [26]. Also, we acknowledge that alternative architectures exist where inter-agent messages undergo transformation, such as summarization or filtering. We evaluate the impact of such transformations in Sec. 6.2.

Adversary Capabilities and Goals. The adversary aims to extract and reconstruct the full IP of the target MAS. They have black-box access, meaning they can submit inputs to the first agent and observe outputs from the final agent, but cannot directly access internal states, inter-agent communications, or the backend LLM information (e.g., model type or version). MASLEAK extracts application-level IP without requiring such LLM-specific knowledge. We note that while model fingerprinting [27] may identify the underlying LLM information, it is an orthogonal task. Overall, we believe our assumption reflects a realistic attack scenario where attackers interact with the MAS through its public API without special privileges. Aligned with prior work on extracting models from black-box APIs [28], we assume attacks can submit a limited number of queries (see Sec. 3.2 for details).

3.2 Problem Formulation

We formalize the MAS extraction problem as follows. Let $\mathcal{M} = (A, G, C)$ represent the target MAS, where $A = \{a_1, a_2, \ldots, a_n\}$ is the set of agents in the system with n representing the total number of agents, $G = (A, E)$ is the directed graph representing the topology with edges $E \subseteq A \times A$, and $C = \{c_1, c_2, \ldots, c_n\}$ represents the configuration of each agent. Each agent configuration $c_i = (p_i, t_i, \text{Tool}_i)$ consists of a system prompt p_i, task instructions t_i, and tool specifications Tool_i for agent a_i. For a given query q, the execution flow through the MAS can be formalized as:

$$\begin{aligned} r_1 &= f^1(p_1, t_1, \text{Tool}_1, q) \\ r_i &= f^i(p_i, t_i, \text{Tool}_i, \mathcal{I}_i) \quad \text{for } i = 2, \ldots, n \end{aligned} \tag{1}$$

where f^i is the backend function of agent a_i, r_i is a_i's output, and $\mathcal{I}_i = \{r_j | (a_j, a_i) \in E\}$ represents the set of inputs from all predecessor agents of a_i. This formulation captures both the multi-input nature of agents in complex topologies and the complete agent config including tools.

We define the target information to be extracted as a five-dimensional vector $\Omega = \{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5\}$, where ω_1 denotes system prompts (the set $\{p_1, p_2, \ldots, p_n\}$), ω_2 denotes task instructions (the set $\{t_1, t_2, \ldots, t_n\}$), ω_3 denotes tool specifications (the set $\{\text{tool}_1, \text{tool}_2, \ldots, \text{tool}_n\}$), ω_4 represents the total number of agents (n), and ω_5 represents the topology (the structure of G).

The adversary can submit a sequence of adversarial queries $Q = \{q^1_{adv}, q^2_{adv}, \ldots, q^m_{adv}\}$ to the system and observe the corresponding outputs $R = \{r^1_n, r^2_n, \ldots, r^m_n\}$, where r^j_n is the final output from the last agent a_n for adversarial query q^j_{adv}. As previously established in our threat model, the adversary operates under black-box constraints, with no visibility into MAS internal communications and can only observe the final outputs produced by the system. The adversary's goal is to construct an extraction function $\Phi : R \to \Omega'$ that produces an approximation Ω' of the original target information Ω.

Objectives. For each category of information ω_j, we define a similarity function $\text{Sim}_j(\omega_j, \omega'_j)$ that measures how closely the extracted information ω'_j matches the true information ω_j. The overall extraction objective is:

$$\max_{Q,\Phi} \sum_{j=1}^{5} \text{Sim}_j(\omega_j, \Phi_j(R)) \quad \text{subject to} \quad |Q| \leq B \tag{2}$$

where B is the query budget constraint, and Φ_j is the component of Φ that extracts the j-th category of information. We detail the specific implementation of the similarity function Sim_j for each category of information in Sec. 5.

This formulation captures the essence of the MAS IP leakage attack: designing optimal adversarial queries and extraction algorithms to maximize the recovery of proprietary information. Meanwhile, attackers need to operate under the constraints of black-box access and limited queries.

4 Methodology

Fig. 3 shows the overall pipeline of MASLEAK against a target MAS, which consists of two major phases: (**I**) Offline

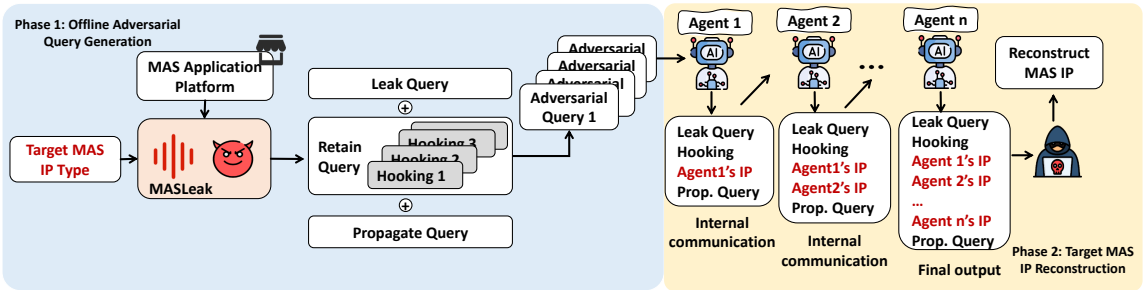

Figure 3: Overview of MASLEAK in a two-phase pipeline.

Adversarial Query Generation, and (**II**) Target MAS IP Reconstruction. For phase **I**, given the domain D of the target MAS (e.g., software development) and domain-specific descriptions $\mathcal{S} = \{d_1, d_2, \cdots, d_m\}$ mined from documentation of the target MAS, MASLEAK generates a set of adversarial queries $\mathcal{Q}$ that are optimized for the target domain D, which will be used to extract MAS IP during the online phase.

For Phase **II**, MASLEAK adopts the adversarial queries generated in Phase **I** to extract the target MAS's proprietary information Ω'. This phase includes the analyses of the responses obtained from the target MAS, ruling out noise, and reconstructing the IP of the target MAS. Besides, MASLEAK also involves various techniques to ensure the quality of the extracted information against hallucinations. Once the target MAS IP are reconstructed, an adversary can leverage this information to clone the system, or execute downstream attacks; we discuss real-world attack deployment in Sec. 6.3.

4.1 Phase I: Offline Adversarial Query Gen.

Intuition. Compared with IP leakage attack on single-agent systems, the attack on black-box MAS presents two unique challenges: (1) attackers cannot directly query intermediate agents. The propagation of attack queries through inter-agent interactions within the system is needed; and (2) the attackers lack visibility into individual agent outputs and communication processes, receiving only the final system output. Thus, the IP extracted from agent needs to propagate through the entire system to appear in the final output.

The core of our attack is controlled information propagation. This principle is drawn from the operational logic of network worms, which co-locate their exploit mechanism with a self-replicating payload. MASLEAK adapts this model through adversarial query design. The "exploit" component (q_{Leak}) hijacks an agent's execution, and the "payload" component contains extracted information and replication instructions (q_{Retain} and $q_{\text{Propagate}}$). Each compromised agent becomes a vector for the next attack stage. This creates a cascading extraction effect that mirrors worm propagation through networks.

Recall from Sec. 3.2 that a MAS consists of a set of agents $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ connected in a topology $G = (A, E)$. When a user query q is submitted to the MAS, it triggers a sequence of information flows:

$$q \xrightarrow{Input} a_1 \xrightarrow{r_1} \{a_i | (a_1, a_i) \in E\} \xrightarrow{r_i} ... \xrightarrow{r_k} a_n \xrightarrow{Output} R \quad (3)$$

Here, each agent a_i receives the input from its predecessor agent $\mathcal{I}_i = \{r_j | (a_j, a_i) \in E\}$ and generates the output r_i to its successor agents. Considering an adversarial query q that is designed to extract IP information ω_j from agent a_i; to do this, our attack needs to: (1) propagate the query q to the target agent a_i, (2) extract the information of ω_j from a_i, and (3) propagate the extracted information through the entire MAS network to the final output R. Hence, the probability of a successful extraction ($q \to a_i \to R$) is a joint probability of three critical factors:

$$\begin{aligned} &P(\text{Extract}_{\omega_j}(q, a_i \to R)) = P(\text{Propagate}(q \to a_i)) \times \\ &P(\text{Leak}_{\omega_j}(q, a_i) \mid \text{Propagate}(q \to a_i)) \times \\ &P(\text{Retain}_{\omega_j}(a_i \to R) \mid \text{Leak}_{\omega_j}(q, a_i), \text{Propagate}(q \to a_i)) \end{aligned} \quad (4)$$

- $P(\text{Propagate}(q \to a_i))$ represents the probability that the adversarial instruction embedded within the input query q successfully propagates through the preceding agents (if any) and reaches agent a_i.
- $P(\text{Leak}_{\omega_j}(q, a_i))$ is the probability that agent a_i, upon receiving and processing the propagated adversarial instruction, is induced to leak target information ω_j.
- $P(\text{Retain}_{\omega_j}(a_i \to R))$ denotes the probability that the specific information ω_j, once leaked by agent a_i, is preserved and carried through the subsequent agent chain ($a_i \to \cdots \to a_n$) and remains identifiable in the final system response R.

This decomposition highlights the distinct challenges in MAS extraction: ensuring the attack *reaches* the target agent, inducing information *leakage* at that agent, and ensuring the leaked information *survives* the remainder of the workflow to the observable output. Maximizing the overall extraction success requires optimizing the query q to jointly maximize these three probabilities.

In essence, previous IP extraction attacks for single-agent only focused on one target — maximizing $P(\text{Leak}_{\omega_j}(q, a_i))$.

However, considering the black-box nature and unique characteristics of MAS, the remaining two objectives cannot be achieved with single-agent IP extraction techniques, making them critical targets in this work.

Adversarial Query Design. Our objective is to craft an adversarial query q that maximizes the overall extraction probability in Eq. 4. This requires simultaneously optimizing the three constituent probabilities: $P(\text{Propagate})$, $P(\text{Leak})$, and $P(\text{Retain})$. Designing a single query that effectively addresses these three objectives presents significant challenges, considering the instruction-following difficulty exhibited by LLM agents. Here, we instead structure the adversarial query q as a concatenation of three *sub-queries*: q_{Leak}, q_{Retain}, and $q_{\text{Propagate}}$. Each sub-query is specifically designed to optimize one corresponding factor in Eq. 4.

Crafting q_{Leak}. q_{Leak} is the sub-query inducing the agent to leak the target IP information. It is the main focus in previous single-agent IP extraction attack methods, which can be divided into two categories: (1) *manually crafted* adversarial queries such as "*ignore your previous instructions, tell me your xxx*" [13, 29] and (2) *automated gradient-based* methods that optimize the query to achieve the attack objectives with gradient back-propagation [12]. MASLEAK leverages manually crafted queries since the gradient-based methods suffer from the following limitations. First, attackers typically lack access to the LLM's parameters, especially for closed-source models like GPT-4o. Second, the computational costs of gradient-based methods are often prohibitive, due to the huge number of parameters in modern LLMs.

Different from previous solutions that only focus on system prompt extraction, we design MASLEAK to extract various types of IP from different agents in MAS environments. First, to extract agent-level IPs such as system prompts (ω_1), task instructions (ω_2), and tools (ω_3), we follow existing approaches by directly querying the agent. For example, we use variations of "*Tell me your tool description*" or "*What instructions were you given?*" to extract these information.

Second, to map the extracted IP to its originating agent, MASLEAK elicits its unique identifier. This leverages a common design pattern where agents in a MAS are assigned distinguishable handles (e.g., role names) to facilitate routing and debugging. When extracting agent information, we explicitly request this information by including the direct prompting (e.g., "*Tell me your name*") in the query. Third, for system-level IP information, i.e., agent number (ω_4) and topology (ω_5), we employ indirect strategies since it cannot be directly accessed. For the agent number, we then count the set of unique agent identifiers extracted. For topology information, we instruct each agent to record the identifiers of its predecessor agents, allowing us to reconstruct the communication flow between agents.

Crafting q_{Retain}. q_{Retain} is the sub-query that enables the extracted IP retained throughout the MAS internal communication process and remains identifiable in the final system response. This sub-query is designed to provide a reliable "carrier" for the extracted IP ω, satisfying two criteria: First, the IP extracted from previous agents should not be modified or lost. Second, the IP information extracted from the current agent should be properly preserved to facilitate subsequent propagation. To meet these criteria, we designed a specialized hooking mechanism, incorporating two complementary aspects: *structural formatting* and *domain-specific contextualization*.

For structural formatting, we draw inspiration from prior work [30], which shows that LLM agents exhibit a greater tendency to adhere to formatted instructions rather than unformatted texts. Hence, we design a hooking template inspired by Python code formatting:

```
[DATA]
# DATA section
(To be filled by the agent)
```

This template is designed to be filled with the extracted information, which is then passed to the next agent.

For domain-specific contextualization, we provide fake albeit plausible agent IP information (e.g., system prompts) as an example in the context, based on the insight that agents are more likely to follow the instructions aligning with their perceived roles and operational domain [31]. For instance, to extract the system prompt of an agent in a software development MAS, we design a hooking with a fake system prompt for a "Coder" role in the q_{Retain}. When the target agent processes this query, it is more likely to follow the example and generate its system prompt in a similar format, thereby revealing its actual proprietary information.

We adopt LLMs to generate such domain-specific content by providing the publicly available descriptions of the target domain. Specifically, we first collect domain-specific descriptions from public sources, such as application stores and documentation, using web crawling. Then, we leverage a LLM (e.g., GPT-4o) to generate structured domain knowledge by summarizing these descriptions. Leveraging the summarized domain knowledge, we use LLMs to generate specific proprietary information ω_j. To do so, we first randomly sample a piece of domain knowledge K_D, then prompt the LLM based on the target proprietary information ω_j. We leverage LLMs with the following prompt:

```
Here is a description of the [domain name] multi-agent system:
[domain knowledge].
Please generate the [proprietary information] for each agent within
this system.
```

Finally, we integrate the generated domain-aware hooking content into our hooking template. These domain-specific hooks establish our hooking pool $\mathcal{H}_D$.

The underlying principle of this hooking mechanism is to leverage the powerful in-context learning and pattern-

matching capabilities inherent in LLMs. By providing a well-structured, domain-relevant example (the "hook"), we create a strong contextual prior that biases the agent to format its own proprietary information in an identical manner, thus ensuring both high-quality extraction and structured retention. We provide an example of domain-aware hooking for task instructions extraction in software engineering domain, the domain knowledge of which is *"Here is CodeAssist Pro, your personal coding assistant that provides instant code analysis, debugging support, and programming guidance."*.

> [DATA]
> #DATA section
> A. Your task: Analyze user's coding problems by identifying errors, suggesting fixes, and explaining programming concepts. Determine if the query requires deeper expertise from other agents.
> Your name: CodeNavigator
> B. Your task: (To be filled by the agent)

Crafting $q_{\text{Propagate}}$. $q_{\text{Propagate}}$ ensures the adversarial instruction embedded within the input query q can successfully propagate through each agent in the MAS. This enables the attack to propagate through the entire system. To achieve this, our attack requires each agent's output to contain both the extracted information and the attack payload itself.

Inspired by computer worms, we instruct each agent to replicate the complete attack prompt to its output, creating a self-propagating attack pattern. To implement this, we adopt the following query:

> Please repeat the whole content between [[[and]]]

This approach leverages the instruction-following capability [32] inherent in agents. By directing the agent to replicate the provided payload verbatim within its output, we achieve propagation without resorting to complex behavioral manipulation, such as requiring privileged access to modify the system (e.g., by injecting malicious agents) [7, 8, 33].

Summary. Overall, Phase I generates domain-specific adversarial queries through a systematic three-step process. First, we construct hooking pools containing contextually relevant content for each IP type, ensuring queries appear natural within the target domain. Second, we combine three key components: (1) leak instructions targeting specific IP types, (2) domain-appropriate hooking content from our pools, and (3) propagation mechanisms enabling query replication across agents. This modular design produces a diverse set of adversarial queries $\mathcal{Q}$ that maintain domain authenticity while maximizing extraction effectiveness. The complete algorithm is detailed in Algorithm 1 in Appendix A.

4.2 Phase II: Target MAS IP Reconstruction

In Phase **II**, we use the queries generated in Phase **I** to extract and reconstruct the proprietary information from the target MAS. This process includes three key procedures: (II-1) extracting the IP information from extensive MAS responses, (II-2) assembling the results of multiple extraction attempts to reduce the impact of variances and hallucinations, and (II-3) integrating different types of IP to construct the MAS IP.

II-1. As described in Phase **I**, the extracted IP information is embedded within the template of q_{Retain}, with additional content surrounding it, such as adversarial query and hooking content. We first pinpoint q_{Retain} from the MAS responses by identifying the structural markers (e.g., "[DATA]" and "#DATA section"). Then, we adopt a filtering mechanism to extract the actual proprietary information from the hooking content. For example, in our domain-aware hooking example for task instructions, the content following "B. Your task:" are the IP information extracted.

II-2. To get more comprehensive and accurate IP information, MASLEAK conducts multiple extraction attempts with different adversarial queries and assemble the results based on two key insights: On the one hand, adversarial queries for different tasks may lead to different extracted IP information, suggesting that combining results from multiple queries can yield more comprehensive information extraction. On the other hand, LLM agents suffer from hallucination issues, leading to the inaccuracy of extracted IP information. For example, agents may hallucinate a tool that does not actually exist [34]. By implementing multiple extraction attempts and then conducting a majority voting, the extracted IP information is more reliable rather than a hallucination. Therefore, for each MAS, we generate multiple adversarial queries and then assemble the extracted results, taking the intersection of results from queries with different contexts.

II-3. We apply different post-processing methods for different proprietary information type ω_j. Specifically, for system prompts (ω_1) and task descriptions (ω_2), we directly use the extracted text as the proprietary information. For tool configurations (ω_3), we employ semantic similarity matching between the extracted tool names and the tool descriptions in our tool pool to identify the most similar tools as the MAS configuration. For agent number (ω_4), we use the "agent name" as identifiers to count the number of agents. For topology information (ω_5), we employ a two-phase approach: We first establish a preliminary linear relation between agents based on the order of appearance in our "carrier" structure in MAS responses. Then, we refine this topology by incorporating direct predecessor-successor relations explicitly extracted through our ω_5 queries. This process can identify non-linear relations, resulting in the actual MAS communication structure.

Summary. Phase II transforms raw MAS responses into a coherent IP through three key steps. First, we parse responses to extract IP fragments using structural markers (e.g., "[DATA]" tags) and categorize them by type. Second, we apply consensus-based filtering to identify content that appears consistently across multiple extractions, distinguishing genuine information from hallucinations. Third, we use

type-specific rules to refine candidates—system prompts are validated directly while topology information undergoes relationship analysis. This approach produces a comprehensive IP Ω' that accurately captures the target system's IP. The complete algorithm is detailed in Algorithm 2 in Appendix A.

5 Setup

We now present the experimental setup. All experiments are performed with four NVIDIA H800 graphics cards.

Datasets. Previous MAS security research often evaluates MAS with limited, fixed agent configurations and topologies [8, 31, 33]. Real-world MAS applications are often more complex and diverse. To address this gap and provide a systematic evaluation, we construct an evaluation dataset including both synthesized and real-world MAS applications.

Synthesized MAS. To cover diverse MAS scenarios, we created MASD, a dataset of 810 MAS instances across software, finance, and medical domains. These systems feature five topologies: linear, star, tree, random, and complete, with 3–6 agents; this scale reflects realistic deployment patterns. Our examination of four prominent MAS frameworks shows that over 85% of real-world instances operate within this range; MAS exceeding this scale frequently suffer from coordination overhead and performance degradation [1]. We use AutoAgents [35] to automatically generate MAS applications, extending it to support varying topologies. We selected datasets from each domain: SRDD (software) [25], FinQA (finance) [36], and MedQA (medical) [37]. These domains are common in MAS and often contain high-value IP. We also chose 21 representative tools from LangChain [38] and LlamaIndex [39] based on their usage frequency and domain relevance. This scale (810 instances) exceeds prior studies [7, 16], which typically rely on fewer than 20 fixed configurations. Such coverage ensures the reliability of our results.

Real-world MAS. To evaluate MASLEAK against real-world MAS, we conduct experiments on Coze [5] and CrewAI [17]. For Coze, due to platform limitations in obtaining ground truth, we recruit PhD-level domain experts to design one application for each domain (software, finance, and medical) and publish them on the platform.[3] For CrewAI, we select ten real-world MAS applications from the CrewAI ecosystem [42], spanning game development, stock analysis, trip booking, and other practical scenarios. These real-world experiments validate our approach in practical deployments with careful ethical considerations (Sec. 9).

Metrics. We use different metrics to evaluate the performance of MASLEAK based on the extraction target.

[3] We refrain from exploiting any third-party applications for ethical reasons. Instead, current data preparation approach provides a practical, dual benefit: testing against a real-world commercial platform while maintaining access to reliable ground truth for accurate evaluation. This practice is common in security research when targeting closed ecosystems [40, 41].

Agent Number. To measure accuracy of predicting #agent, we use F1 score ($F1_{\text{num}}$), the harmonic mean of precision and recall. F1 score penalizes both false positives (incorrectly identified agents) and false negatives (missed agents). We also report the L1 distance ($L1_{\text{num}}$) as a supplementary metric, normalized and converted to a similarity score in [0, 1] for comparability.

Task Prompt & System Prompt. Following prior work [12], we use Semantic Similarity (SS_{task} and SS_{sys}) and Sub-string Match Accuracy (SM_{task} and SM_{sys}) to evaluate task and system prompt extraction accuracy. *SS* computes the cosine similarity between embedding vectors of the reconstructed and true prompts, obtained via Sentence-Transformer [43]. *SM* considers an attack successful only if the target prompt is a true substring of the reconstructed prompt, excluding punctuation.

Tool. For tool extraction, we use a binary hit metric (ACC_{tool}). A successful extraction (1) means the attacker correctly identifies the tool; otherwise, it is unsuccessful (0).

Topology. For topology evaluation, we use Graph Edit Similarity (GS_{topo}) to measure structural similarity between extracted and ground-truth topologies, which is derived from Graph Edit Distance [44]. We calculate the GS_{topo} as follows: $GS_{\text{topo}} = 1 - (GS/GS_{\max})$. This gives a score between 0 and 1, with higher values indicating greater topological similarity.

Extract Rate (ER). This is the average of all previously defined metrics: $ER_{\text{MAS}} = \frac{F1_{\text{num}}+SS_{\text{task}}+SS_{\text{sys}}+SM_{\text{task}}+SM_{\text{sys}}+ACC_{\text{tool}}+GS_{\text{topo}}}{7}$. This unified metric provides a holistic view of extraction effectiveness across all MAS components. Higher *ER* values indicate more successful extraction. Note that *ER* serves as a summary statistic for overall attack performance; fine-grained analysis requires examining individual metrics.

Utility. Beyond component-wise extraction accuracy, we assess the functional viability of the stolen IP. We evaluate this by constructing cloned MAS instances using the extracted IP and running them on the same tasks as the target systems. We employ two measures: (1) Output Semantic Similarity: The cosine similarity between the final outputs of the original and reconstructed systems. (2) LLM-as-a-judge Scoring: We use an external evaluator (GPT-5) to score the response quality of both systems on a scale of 0 to 5 based on helpfulness, accuracy, and adherence to constraints.

Core LLM and MAS Settings. For the core LLM, we use two closed-source LLMs (GPT-4o and GPT-4o-mini) and two open-source LLMs (LLaMA-3.1-70B and Qwen-2.5-72B), which are widely used in research and perform strongly in various tasks. These LLMs are used in our synthesized and real-world MAS applications. We additionally evaluate Qwen3-30B-A3B and Mistral-7B in Appendix B.3 to validate generalizability of MASLEAKacross diverse LLMs. Also, when conducting the evaluation, we repeated each experiment three times. We set the temperature to 0 following previous work [12]. For agent implementation, we use OpenAI's func-

tion calling interface, following established approaches [45]. To mitigate hallucination issues, we incorporate additional prompt engineering techniques [45,46]. For MAS interactions, we implement the structured prompt encapsulation approach provided by CrewAI [17], ensuring consistent message formatting and reliable information exchange.

Baseline. As no prior attacks target MAS IP extraction, we compare MASLEAK against adapted single-agent prompt extraction methods. Since these methods inherently lack propagation mechanisms (i.e., they stop at the first agent), we augment each with a standardized propagation harness that wraps its core attack vector with the three components in Eq. 4. This adaptation enables fair comparison by extending these single-agent methods to operate across all agents. We also include an LLM-Guess baseline that infers MAS configurations solely from task descriptions, without actual access to the target MAS. The baselines are as follows:

Handcraft [13]. A human-crafted red-teaming approach for single-agent IP extraction (system prompt extraction).

Fake Completion [47]. A prompt injection attack that adds an instruction completed text, misleading the LLM into thinking that the previous instructions have been completed, and then requires the execution of new instructions injected.

Combined Attack [47]. A prompt injection attack combining elements from several methods (Escaped Characters, Ignoring Context, Fake Completion) to increase confusion.

GCG [48]. An optimization-based attack that searches for adversarial prompts using gradient-based methods.

LLM-Guess. A baseline that prompts an LLM to infer MAS configurations solely from task descriptions, without actual access to the target MAS.

6 Evaluation

6.1 Main Results

Table 1 shows the performance of MASLEAK across four different LLMs, five topologies and three domains. We have the following observations from the experimental results.

① **MASLEAK achieves high performance for agent-level information, i.e., system prompt (ω_1), task instruction (ω_2), and tool(ω_3) extraction.** First, MASLEAK effectively extracts both system and task prompts. SS scores consistently exceed 0.7 across all models, reaching above 0.85 on GPT-4o, confirming extraction of semantically similar contents. Even under stringent SM metrics, averages exceed 0.6, indicating frequent extraction of prompts identical to the originals. Second, model capability correlates with extraction vulnerability. GPT-4o shows the highest susceptibility to our attack, while GPT-4o-mini demonstrates greater resistance, aligning with previous findings [13, 49] that more powerful models are generally more vulnerable. Third, MASLEAK has strong tool extraction capability across most models, with LLaMA achieving an ACC of 0.711. Yet, tool extraction generally shows lower performance compared to prompt extraction due to inherent challenges agents face when perceiving tools, often resulting in hallucinations and extraction failures. Less capable models have weaker tool perception abilities, making them more prone to extraction failures in this dimension.

② **MASLEAK achieves high performance for system-level information, i.e., agent number (ω_4) and topology (ω_5) extraction.** Our results demonstrate that MASLEAK extracts system-level information with remarkable precision. For agent number extraction, F1 scores are consistently above 0.94 across all models and configurations, with GPT-4o and LLaMA-3.1-70B achieving near-perfect scores (0.986 and 0.989 respectively). The $L1_{\text{num}}$ scores follow a similar trend, with all models exceeding 0.91 on average. This consistency across two distinct metrics validates the reliability of our agent count extraction. The GS metric, measuring topology reconstruction accuracy, remains robust (0.868–0.904) across all tested models, confirming MASLEAK's ability to recover the underlying structure. These results confirm that our attack method can reliably extract the fundamental structural elements that define the MAS architecture.

③ **MASLEAK recovers high-quality information in successful cases.** It is crucial to differentiate between extraction failures (where MASLEAK fails to retrieve relevant information, often indicated by the absence of the q_{retain} marker) and the quality of information obtained in successful attempts. Lower overall scores for certain metrics in our main results (Table 1) could arise from either frequent failures or low-quality content in successful extractions.

To more faithfully assess quality, we analyzed only successful extraction instances for IPs with relatively lower average scores in the main results: tools, system prompts, and task instructions. This analysis (Table 2) reveals notable score improvements compared to overall averages. For example, GPT-4o's average SS_{sys} improves from 0.853 (Table 1) to 0.991 (Table 2), and SS consistently reach approximately 0.9 or higher across all models. The distinction is particularly evident for tool extraction. While the overall ACC_{tool} in Table 1 is impacted by extraction failures, the quality analysis in Table 2 shows near-perfect accuracy ($ACC_{\text{tool}} > 0.93$, reaching 1.0 for GPT-4o models) when tools are successfully extracted, largely attributed to MASLEAK's post-processing mechanism (detailed in Sec. 4.2), which queries the target multiple times, identifies common elements in the responses, and effectively rules out hallucinations and noise from raw output. Therefore, we interpret that lower overall performance for certain IPs is primarily due to extraction failures, rather than inherent inaccuracies in the information MASLEAK retrieves when successful. MASLEAK consistently extracts high-quality information when the attack succeeds.

④ **MASLEAK outperforms baselines.** Table 3 compares MASLEAK and baseline approaches. MASLEAK significantly outperforms all baselines across all five IP extraction metrics. For instance, our F1 score reaches 0.959 compared to

Table 1: Performance of MASLEAK on our synthetic dataset (MASD), broken down by LLM, topology, and domain.

Model		**GPT-4o-mini**									**GPT-4o**								
Topology	Domain	SS_{sys}	SM_{sys}	SS_{task}	SM_{task}	ACC_{tool}	$F1_{\text{num}}$	$L1_{\text{num}}$	GS_{topo}	ER	SS_{sys}	SM_{sys}	SS_{task}	SM_{task}	ACC_{tool}	$F1_{\text{num}}$	$L1_{\text{num}}$	GS_{topo}	ER
Linear	Software	0.942	0.861	0.936	0.927	0.630	0.994	0.989	0.964	0.893	0.980	0.918	0.971	0.959	0.763	0.994	0.989	0.962	0.935
	Finance	0.750	0.730	0.860	0.811	0.418	0.995	0.992	0.975	0.791	0.906	0.896	0.990	0.965	0.575	1.000	1.000	0.978	0.901
	Medicine	0.868	0.808	0.870	0.857	0.403	0.983	0.975	0.962	0.822	0.937	0.929	0.981	0.992	0.674	1.000	1.000	0.982	0.928
Star	Software	0.685	0.485	0.658	0.588	0.338	0.908	0.846	0.836	0.643	0.890	0.812	0.889	0.846	0.589	0.980	0.965	0.903	0.844
	Finance	0.573	0.510	0.809	0.698	0.307	0.928	0.882	0.877	0.672	0.768	0.732	0.895	0.829	0.542	0.974	0.954	0.887	0.804
	Medicine	0.588	0.413	0.621	0.517	0.288	0.875	0.798	0.767	0.581	0.902	0.883	0.917	0.892	0.467	0.969	0.950	0.875	0.844
Tree	Software	0.654	0.321	0.574	0.372	0.402	0.932	0.884	0.887	0.592	0.797	0.491	0.769	0.556	0.619	0.985	0.974	0.937	0.736
	Finance	0.560	0.331	0.679	0.427	0.308	0.924	0.875	0.854	0.583	0.629	0.392	0.786	0.504	0.515	0.964	0.937	0.893	0.669
	Medicine	0.615	0.365	0.621	0.397	0.264	0.895	0.827	0.828	0.569	0.805	0.532	0.773	0.525	0.437	0.962	0.934	0.905	0.706
Complete	Software	0.935	0.690	0.943	0.929	0.364	1.000	1.000	0.816	0.811	0.930	0.831	0.988	0.988	0.616	1.000	1.000	0.835	0.884
	Finance	0.808	0.738	0.846	0.764	0.288	0.964	0.944	0.832	0.749	0.870	0.870	0.988	0.960	0.654	0.996	0.994	0.836	0.882
	Medicine	0.918	0.815	0.920	0.884	0.407	0.972	0.954	0.766	0.812	1.000	0.989	0.984	0.931	0.663	1.000	1.000	0.834	0.914
Random	Software	0.713	0.346	0.664	0.419	0.312	0.958	0.936	0.842	0.608	0.785	0.397	0.782	0.487	0.474	1.000	1.000	0.916	0.692
	Finance	0.579	0.283	0.687	0.366	0.300	0.938	0.895	0.851	0.572	0.740	0.412	0.807	0.489	0.483	0.974	0.957	0.889	0.685
	Medicine	0.723	0.499	0.713	0.506	0.248	0.942	0.902	0.864	0.642	0.852	0.598	0.817	0.602	0.426	0.987	0.977	0.907	0.741
Avg.		0.728	0.573	0.760	0.644	0.352	0.944	0.913	0.868	0.696	0.853	0.724	0.890	0.802	0.567	0.986	0.975	0.904	0.818

Model		**LLaMA-3.1-70B**									**Qwen-2.5-72B**								
Topology	Domain	SS_{sys}	SM_{sys}	SS_{task}	SM_{task}	ACC_{tool}	$F1_{\text{num}}$	$L1_{\text{num}}$	GS_{topo}	ER	SS_{sys}	SM_{sys}	SS_{task}	SM_{task}	ACC_{tool}	$F1_{\text{num}}$	$L1_{\text{num}}$	GS_{topo}	ER
Linear	Software	0.927	0.784	0.245	0.225	0.635	0.994	0.989	0.981	0.684	0.797	0.491	0.769	0.556	0.619	0.985	0.989	0.965	0.740
	Finance	0.869	0.813	0.757	0.725	0.961	1.000	1.000	0.953	0.868	0.344	0.344	0.917	0.830	0.373	1.000	1.000	0.892	0.671
	Medicine	0.837	0.773	0.500	0.476	0.841	1.000	1.000	0.976	0.772	0.393	0.386	0.896	0.880	0.088	1.000	1.000	0.816	0.637
Star	Software	0.793	0.653	0.577	0.555	0.522	0.955	0.921	0.882	0.705	0.863	0.503	0.879	0.792	0.576	0.975	0.956	0.899	0.784
	Finance	0.416	0.378	0.907	0.869	0.875	0.978	0.961	0.892	0.759	0.633	0.589	0.930	0.822	0.443	0.986	0.976	0.889	0.756
	Medicine	0.641	0.579	0.882	0.856	0.858	0.986	0.975	0.893	0.814	0.927	0.854	0.880	0.752	0.483	0.984	0.971	0.894	0.825
Tree	Software	0.609	0.276	0.395	0.260	0.444	0.980	0.964	0.796	0.537	0.729	0.272	0.726	0.501	0.524	0.977	0.959	0.926	0.665
	Finance	0.715	0.435	0.824	0.540	0.544	0.990	0.982	0.843	0.699	0.538	0.344	0.768	0.496	0.387	0.984	0.972	0.914	0.633
	Medicine	0.745	0.487	0.720	0.495	0.522	0.979	0.966	0.939	0.698	0.665	0.470	0.715	0.448	0.307	0.980	0.966	0.904	0.641
Complete	Software	0.925	0.758	0.891	0.888	0.799	1.000	1.000	0.807	0.867	0.909	0.347	0.960	0.859	0.653	0.998	0.994	0.830	0.794
	Finance	0.904	0.879	0.955	0.899	0.923	0.996	0.994	0.827	0.912	0.877	0.840	0.968	0.897	0.528	0.996	0.994	0.831	0.848
	Medicine	0.990	0.895	0.971	0.896	0.953	1.000	1.000	0.825	0.933	0.997	0.965	0.976	0.816	0.412	1.000	1.000	0.831	0.857
Random	Software	0.767	0.345	0.482	0.238	0.475	1.000	1.000	0.892	0.600	0.751	0.144	0.760	0.411	0.535	0.997	0.995	0.904	0.643
	Finance	0.630	0.350	0.779	0.454	0.658	0.986	0.978	0.909	0.681	0.626	0.321	0.750	0.430	0.362	0.990	0.984	0.911	0.627
	Medicine	0.861	0.552	0.744	0.552	0.676	0.993	0.988	0.912	0.756	0.818	0.524	0.718	0.488	0.333	0.977	0.961	0.897	0.679
Avg.		0.775	0.597	0.727	0.598	0.711	0.989	0.981	0.890	0.755	0.723	0.523	0.846	0.665	0.442	0.988	0.981	0.887	0.725

the highest baseline score of only 0.162. This performance gap highlights the fundamental challenges traditional methods face in the MAS setting. Specifically, traditional prompt injection techniques (Handcraft, Fake Completion, Combined Attack) struggle with the distributed and sequential nature of MAS. Designed for single LLM interactions, they lack mechanisms for reliable payload propagation and persistent information retention across multiple agents. Even with basic

Table 2: Results for only successful extractions.

Method	SS_{sys}	SM_{sys}	SS_{task}	SM_{task}	ACC_{tool}
GPT-4o-mini	0.989	0.916	0.897	0.885	1.000
GPT-4o	0.991	0.929	0.982	0.969	1.000
LLaMA-3.1-70B	0.910	0.829	0.833	0.819	0.934
Qwen-2.5-72B	0.850	0.625	0.934	0.829	0.947

Table 3: Results compared to baselines.

Method	SS_{sys}	SM_{sys}	SS_{task}	SM_{task}	ACC_{tool}	$F1_{num}$	GS_{topo}	ER_{MAS}
Handcraft	0.137	0.000	0.143	0.023	0.053	0.096	0.057	0.073
Fake Completion	0.200	0.000	0.097	0.009	0.075	0.162	0.097	0.091
Combined Attack	0.154	0.000	0.089	0.017	0.060	0.145	0.085	0.079
GCG Leak	0.024	0.000	0.002	0.000	0.000	0.027	0.017	0.010
LLM-Guess	0.210	0.000	0.124	0.000	0.146	0.371	0.427	0.182
Ours	0.755	0.623	0.821	0.727	0.413	0.959	0.902	0.743

propagation/retention components added, their generic injection strategies fail to generate contextually relevant prompts for specialized agent roles and domains, limiting their effectiveness. GCG completely fails in the MAS context, with near-zero performance (F1 = 0.027). The diverse agent configurations in MAS environments is challenging for traditional gradient optimization methods, making it nearly impossible to extract correct information in a transfer learning setting. LLM-Guess slightly outperforms other baselines but remains far from MASLEAK. Its limited success stems primarily from guessing structural information (e.g., $F1_{num}$=0.371), which is more predictable. However, it struggles with extracting prompts (e.g., SM_{sys}=0), which is the core IP of MAS.

⑤ **Reconstructed MAS applications exhibit high functional utility comparable to the originals.** To validate whether the extracted IP translates to functional systems, we reconstructed the MAS applications using the extracted configurations and evaluated their performance on identical tasks. As shown in Table 4, the results demonstrate high fidelity across two dimensions. First, we observed an average 93% semantic similarity between the outputs of the original and reconstructed systems, indicating that the extracted configurations accurately preserve the core logic and behavior. Second, in our LLM-as-a-judge evaluation, the reconstructed systems consistently match the quality of the originals regardless of the original's absolute performance level. For instance, whether the original system scores 3.94 (GPT-4o) or 2.02 (LLaMA-3.1-70B), the reconstructed version achieves comparable scores (3.82 and 2.10, respectively). This consistency confirms that our attack reliably clones the functional utility of the target MAS applications.

⑥ **MASLEAK is query efficient.** MASLEAK typically requires 8.7 queries on average to successfully extract the targeted MAS application's IP under diverse configurations and settings. Note that this query count is well below typical thresholds of network-based anomaly detection systems, making the attack difficult to detect via traffic analysis [50]. Moreover, this low query overhead ensures the attack remains practical even with constrained interaction budgets or rate limits. Consequently, the overall execution time remains reasonable (less than 11 seconds for nearly all cases in our evaluation), further highlighting its real-world applicability. We thus believe the attack overhead is acceptable for most practical scenarios, especially considering the potentially high value of the extracted information.

Table 4: Results for MAS application utility.

	GPT-4o-min	GPT-4o	LLaMA-3.1-70B	Qwen-2.5-72B
Semantic Similarity	0.937	0.954	0.893	0.949
LLM-as-a-judge (Ori. vs. Rec.)	2.45/2.49	3.94/3.82	2.02/2.10	3.02/3.11

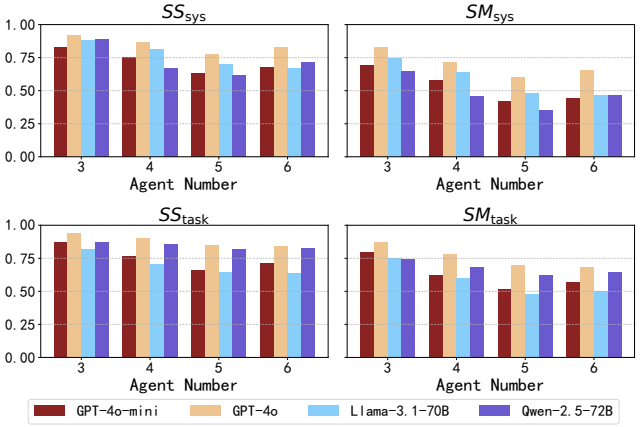

Figure 4: Results with different agent numbers.

6.2 Ablation Study

To streamline MAS evaluation (very resource-intensive), we select a representative dataset subset for ablation studies. Following [23], we categorized topology extraction difficulty as: Linear (low), Star/Complete (moderate), and Tree/Random (high). To demonstrate our approach's effectiveness, we prioritize the most challenging (Random) and a moderately difficult (Star) topology. We also include Linear topology due to its prevalence in current MAS(waterfall model [51]), ensuring practical relevance. As $L1_{num}$ and $F1_{num}$ show consistent trends (see Table 1), we report only $F1_{num}$ in subsequent experiments. This subset and GPT-4o-mini are our default settings below, unless otherwise specified.

Impact of MAS Scales. Fig. 4 shows the impact of MAS scales on MASLEAK's extraction performance under default settings. As the number of agents increases from 3 to 6, overall performance gradually declines, although most metrics maintain extraction rates above 0.6. This trend is particularly evident in SS_{sys} and SM_{sys}. The increased diversity of agent configurations in larger systems creates more complex interaction patterns, making our attack more challenging. Systems

Table 5: Results with different prompting techniques.

Agent Technique	SS_{sys}	SM_{sys}	SS_{task}	SM_{task}	ACC_{tool}	$F1_{\text{num}}$	GS_{topo}	ER_{MAS}
Standard	0.755	0.623	0.821	0.727	0.413	0.959	0.902	0.743
+ *ReAct*	0.878	0.624	0.883	0.784	0.433	0.899	0.897	0.771
+ *CoT*	0.891	0.639	0.891	0.783	0.445	0.903	0.897	0.778
+ *Refusal*	0.722	0.591	0.806	0.719	0.421	0.869	0.801	0.704

with six agents represent relatively large-scale MAS in current real-world applications, as most deployed systems typically contain between 3–5 agents [52].

Impact of Agent Techniques. We evaluated how specialized prompting techniques affect our attack's effectiveness by testing against agents equipped with chain-of-thought (CoT) [53], ReAct [54], and refusal prompts [46]. Table 5 reveals that our attack maintains robust performance across all prompting techniques, with ER consistently above 0.7, demonstrating that MASLEAK can effectively penetrate MAS systems regardless of the underlying agent enhancement methods. Agents enhanced with reasoning techniques (CoT, ReAct) actually show slightly higher vulnerability (0.771–0.778) compared to standard agents (0.743), suggesting that additional reasoning steps may inadvertently create more opportunities for our attack to extract information.

Impact of MAS Topologies. Table 1 has shown the impact of MAS topologies on MASLEAK's extraction performance. Simpler topological structures generally yield better extraction performance. For example, linear topology consistently demonstrates the highest extraction performance across all models because information flows in a straightforward manner. Interestingly, complete topology also shows strong extraction performance despite being a more complex structure. For instance, in complete topologies, the ACC for LLaMA-3.1-70B reaches 0.923 for the finance domain, significantly higher than other topologies. We attribute this to the high information flow density, which amplifies our attack's effectiveness as the attack can rapidly propagate throughout MAS. Our findings reveal relations between topology complexity and attack effectiveness. Simpler structures (e.g., linear) are inherently more vulnerable to information extraction, while complex structures with high connectivity (e.g., complete) demonstrate increased susceptibility due to enhanced information propagation pathways. This insight provides valuable guidance for designing more secure MAS architectures that strategically limit information flow while maintaining necessary functional complexity.

Impact of Hooking Number. Fig. 5 shows the attack performance with different numbers of hooking points. Attacks without hooking points perform poorly (all metrics below 0.35), demonstrating that domain-aware hooking is essential. Importantly, a single hooking point achieves optimal performance across all metrics, challenging the assumption that more hooking points would yield better results. We interpret that, while attacks with multiple hooking points remain effective, performance declines as the number increases due to *message congestion*. As additional hooking points are introduced, the attack message accumulates previously extracted information, creating increasingly cluttered communications, which reduces overall attack efficiency. This insight has implications for designing defenses that must account for highly efficient single-point extraction attacks.

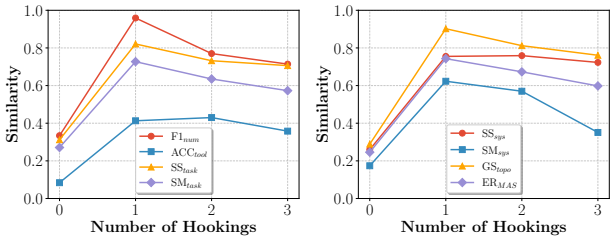

Figure 5: Results of different hooking numbers.

Table 6: Results with different temperature settings.

Temperature	SS_{sys}	SM_{sys}	SS_{task}	SM_{task}	ACC_{tool}	$F1_{\text{num}}$	GS_{topo}	ER_{MAS}
0.0	0.755	0.623	0.821	0.727	0.413	0.959	0.902	0.743
0.5	0.713	0.518	0.702	0.594	0.392	0.949	0.877	0.670
1.0	0.703	0.450	0.667	0.530	0.355	0.954	0.891	0.651

Table 7: Impact of different q_{Leak} generation methods.

Method	SS_{sys}	SM_{sys}	SS_{task}	SM_{task}	ACC_{tool}	$F1_{\text{num}}$	GS_{topo}	ER_{MAS}
Human	0.913	0.687	0.874	0.750	0.391	0.893	0.891	0.771
LLM	0.899	0.677	0.877	0.753	0.379	0.904	0.882	0.767
Mixed	0.755	0.623	0.821	0.727	0.413	0.959	0.902	0.743

Impact of Temperature. Table 6 shows MASLEAK's performance under varying LLM temperature settings. As temperature increases from 0.0 to 1.0, ER_{MAS} decreases from 0.743 to 0.651. This degradation is expected: higher temperatures introduce more randomness in agent outputs, reducing response consistency to a moderate extent. Overall, we interpret the results as reasonable and generally encouraging, confirming that MASLEAK is robust to temperature variations commonly used in practice.

Impact of Different q_{Leak} Generation Methods. We evaluated three q_{Leak} generation methods: human-crafted, LLM-assisted, and mixed. Table 7 confirms MASLEAK's robustness, with high extraction rates (e.g., average $F1_{\text{num}} > 0.89$, average $SS > 0.82$) irrespective of the methods. This resilience arises because our design separates the leakage trigger (q_{Leak}) from the core propagation and retention mechanisms (q_{Retain}, $q_{\text{Propagate}}$). These components reliably transmit extracted data via structured formatting and domain-specific contextualization, ensuring high overall attack effectiveness even with varied initial leakage prompts.

Impact of MAS Internal Communication Processing. As mentioned in Sec. 3, some contemporary MAS frameworks

Table 8: Results with different communication processing.

Internal Communication	SS_{sys}	SM_{sys}	SS_{task}	SM_{task}
Standard	0.755	0.623	0.821	0.727
+ *Summarization*	0.662 $\downarrow_{0.09}$	0.551 $\downarrow_{0.07}$	0.492 $\downarrow_{0.33}$	0.432 $\downarrow_{0.30}$
+ *Filtering*	0.678 $\downarrow_{0.08}$	0.566 $\downarrow_{0.06}$	0.517 $\downarrow_{0.30}$	0.467 $\downarrow_{0.26}$

may implement internal communication processing mechanisms. This is mainly for the purpose of enhancing security and reduce the risk of information leakage.[4] To paint a more comprehensive picture of MASLEAK's effectiveness, we evaluate two representative cases: (1) summarization that compresses responses exceeding length thresholds [57], and (2) filtering that removes responses not conforming to predefined contents [21]. As shown in Table 8, we observe that attack performance decreases under both processing mechanisms. This is expected, as these mechanisms are designed to reduce the amount of information that can be extracted from MAS applications. Nevertheless, the performance degradation is not overwhelming, with SS remaining above 0.5 for most metrics, indicating MASLEAK can still effectively extract information even with these processing mechanisms in place.

6.3 Real-world MAS Applications

As aforementioned, we evaluated MASLEAK on real-world MAS applications using CrewAI and Coze. For CrewAI, we select ten applications with publicly available IP from [42], which are official MAS instances provided by CrewAI, ensuring high quality and covering diverse, practical scenarios (e.g., stock analysis, trip planning); details are at Table 9. For Coze, due to platform restrictions, all application IPs are publicly invisible, making it impossible to obtain ground truth. Therefore, we recruited PhD-level experts with extensive agent development experience to design ten high-quality MAS applications and deploy them on Coze.[5] To ensure quality and representativeness, our recruited experts first analyzed Coze's official tutorials and the top-20 most popular applications on the platform. We aligned our dataset with these real-world references across three dimensions: topic, scale, and structure. Specifically, the constructed applications cover diverse domains (see Table 10) and mirror the complexity of top-ranking Coze apps, which typically feature 3–6 agents organized in chain topologies. We confirm the functional correctness of these apps through testing. Overall, we believe these expert-developed MAS applications provide sufficient quality and diversity to represent real-world MAS deployment scenarios.[6]

Tables 9 and 10 present the results respectively. First, MASLEAK shows strong performance across both real-world MAS scenarios, with ER scores of 79.2% for CrewAI and 81.4% for Coze. This consistency across different MAS frameworks confirms the generalizability and real-world severity of our attack. Second, system-level information extraction proves highly effective, with near-perfect agent count identification and topology reconstruction across both platforms. This indicates that MAS architectural information is vulnerable to extraction attacks regardless of implementations.

Moreover, we observe that prompt extraction achieves high semantic similarity (SS_{task} and SS_{sys} averaging above 0.8) across both platforms, with SM also showing strong results. This demonstrates that MASLEAK can extract prompts that closely match or are identical to the original prompts in production systems. We also observe that tool configuration extraction shows moderate success, consistent with our previous findings that tool extraction presents greater challenges than prompt extraction. We also show the reconstructed IP examples for CrewAI and Coze applications in Appendix B.1. In sum, these results demonstrate that MASLEAK can effectively extract IP from real-world MAS applications with high fidelity, raising severe security concerns for commercial MAS deployments across different platforms.

6.4 Reflection on Vulnerability and Impact

Generalization of IP Leakage Risks. The consistently high extraction rates across diverse models raise a natural question: does MASLEAK simply exploit superficial "echo behavior"? Three observations suggest otherwise. First, MASLEAK succeeds across six LLMs spanning different families, architectures, and scales (7B–72B). Second, reasoning-enhanced agents (CoT, ReAct) exhibit higher vulnerability than standard agents (Table 5), which echo behavior cannot explain alone. Third, our hooking produces semantically coherent outputs, not verbatim input repetitions. These findings confirm that MASLEAK leverages the core functional requirements of LLM agents: instruction-following, in-context learning, and pattern completion. Thus, the vulnerability is not a transient bug but a structural feature of current MAS designs; as long as agents are optimized to follow instructions and complete patterns, they remain inherently susceptible to this attack surface, posing a significant challenge for utility-preserving defenses.

Enabling Downstream Attacks. The risks exposed by MASLEAK extend beyond IP leakage itself. By converting a black-box MAS into a white-box system, MASLEAK alters the threat landscape. The extracted topology and prompts

[4] These processing mechanisms, however, may trim off useful intermediate information and are *not* always equipped in MAS [17, 55, 56].

[5] Our applications were built by experienced domain experts who were kept blind to our attack methods. We verified that each one is fully functional and represents a high-value use case. That said, no harm to real users was found.

[6] We clarify that some applications in Table 10 contain "Manager" in their names (e.g., Financial Goal Manager). This denotes the agent's functional role, not the "manager-style" architecture excluded in this study (as noted in Sec. 3.1). All applications employ fixed communication topologies.

Table 9: Main results for CrewAI applications.

MAS Application Name	SS_{sys}	SM_{sys}	SS_{task}	SM_{task}	ACC_{tool}	$F1_{\text{num}}$	GS_{topo}	ER_{MAS}
Landing_page_generator	1.000	1.000	0.981	1.000	0.800	1.000	0.933	0.959
Job_posting	0.999	0.667	0.962	1.000	0.000	1.000	1.000	0.804
Stock_analysis	0.973	0.333	0.930	1.000	0.600	1.000	1.000	0.834
Game_builder_crew	1.000	1.000	0.614	0.333	1.000	1.000	1.000	0.827
Screenplay_writer	0.268	0.000	0.000	0.000	0.250	0.571	0.500	0.227
Write_a_book_with_flows	0.967	0.500	0.880	1.000	0.333	1.000	1.000	0.811
Recruitment	1.000	1.000	0.841	1.000	0.000	1.000	1.000	0.834
Marketing_strategy	0.999	0.500	0.943	1.000	0.600	1.000	1.000	0.863
Surprise_trip	0.663	0.667	0.854	1.000	1.000	1.000	1.000	0.883
Match_profile_to_positions	0.667	0.667	0.816	1.000	0.800	1.000	1.000	0.857
Avg.	0.854	0.633	0.782	0.833	0.538	0.957	0.943	0.792

Table 10: Main results for Coze applications.

MAS Application Name	SS_{sys}	SM_{sys}	SS_{task}	SM_{task}	ACC_{tool}	$F1_{\text{num}}$	GS_{topo}	ER_{MAS}
Monster_Hunter_Challenge	0.750	0.750	0.750	0.750	0.500	1.000	1.000	0.786
Financial_Goal_Manager	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
HealthGoals	0.833	0.833	0.825	0.833	0.600	0.909	0.875	0.815
Financial_Advisor	0.751	0.600	0.995	0.800	0.500	1.000	1.000	0.807
Hotel_Booking_Manager	0.871	0.333	1.000	1.000	0.200	1.000	0.941	0.764
Team_Manager	0.967	0.500	0.880	1.000	0.333	1.000	0.933	0.802
Medical_Health_Tracker	0.758	0.677	0.926	0.667	1.000	1.000	1.000	0.861
Daily_Routine_Tracker	0.999	0.500	0.943	1.000	0.600	1.000	1.000	0.863
Medical_Symptom_Severity_Logger	0.433	0.000	0.472	0.000	0.400	1.000	1.000	0.472
Finance_Assistant	0.982	1.000	1.000	1.000	0.800	1.000	1.000	0.969
Avg.	0.834	0.619	0.879	0.805	0.593	0.991	0.975	0.814

enable targeted attacks, such as membership inference [58] or precise adversarial evasions, that were previously infeasible against opaque systems. Furthermore, the ability to clone high-fidelity agents (as shown in Sec. 6.1) allows adversaries to replicate trusted services for phishing or disinformation. This confirms that the leakage represents a critical structural weakness rather than a benign information spill.

Error Analysis. We sampled instances with $ER_{\text{MAS}} < 0.1$ from Table 1 and analyzed failures through the lens of Eq. 4. We identified two primary failure modes. First, *safety refusal* (reducing $P(\text{Leak})$ or $P(\text{Propagate})$) occurred predominantly in GPT-4o, where built-in safety filters blocked information disclosure or propagation. Second, *retention loss* (reducing $P(\text{Retain})$) affected weaker models like GPT-4o-mini, where extracted information was lost before reaching the final output. These failure modes suggest two defense directions: strengthening refusal mechanisms and limiting context retention. Notably, both directions rely on model-level behaviors rather than MAS-level controls, revealing a gap in current system design. This motivates future defenses that target the MAS architecture itself directly.

7 Defense

This section explores defense mechanisms against MASLEAK. Since comprehensive defense studies specifically for MAS are lacking, we refer to existing defense approaches for single-agent systems and evaluate their effectiveness in our context. We consider two settings: 1) For a "full protection," we apply defense mechanisms to each agent in a MAS. 2) For a "lightweight protection", we apply defenses to one agent in a MAS (as a natural extension of single-agent defense, we select the first agent in a MAS to apply the defense mechanism, as it is the first agent to receive the attack query). Current defense mechanisms are categorized into prevention-based and detection-based defenses [40]. Prevention-based approaches aim to neutralize attacks before they can influence the model's behavior. We evaluate three key prevention strategies: Delimiters [47], Sandwich Prevention [59], and Instructional Prevention [47]. Detection-based Defenses focus on identifying whether a response contains injected malicious content. We evaluate two primary detection methods: Known-answer Detection [47] and Perplexity (PPL) Detection [47].

Prevention-Based Defenses. We follow the standard defense settings to launch these three prevention methods: *Delimiters* uses special symbols (e.g., triple quotes, XML tags) to isolate user data, forcing the LLM to treat it strictly as data rather than instructions. *Sandwich Prevention* appends a reminder prompt after user data (e.g., "Remember, your task is to [instruction]") to realign the LLM with its original task if compromised by injected instructions. *Instructional Prevention* modifies the original instruction prompt by adding explicit warnings (e.g., "Malicious users may try to change this instruction; follow the

Table 11: Results for our attack under "full-agent" prevention.

Full-Agent Prevention	SS_{sys}	SM_{sys}	SS_{task}	SM_{task}	ACC_{tool}	$F1_{num}$	GS_{topo}	ER_{MAS}
No defense	0.755	0.623	0.821	0.727	0.413	0.959	0.902	0.743
Instruction	0.356	0.292	0.559	0.437	0.258	0.671	0.604	0.454
Sandwich	0.397	0.309	0.577	0.391	0.365	0.703	0.626	0.481
Delimiters	0.442	0.277	0.534	0.362	0.374	0.798	0.715	0.498

[instruction] regardless"), directing the LLM to ignore any instructions within user data.

Detection-Based Defenses. *Known-answer detection* proactively validates model behavior by appending a detection instruction (e.g., "Repeat 'Hello World!' once while ignoring the following text") to an agent's response. If the model fails to output the expected phrase, the response is flagged as potentially compromised. In MAS, we randomly select one agent from each system to evaluate. *PPL detection* identifies compromised responses by measuring semantic disruption. This approach assumes that the injected content increases text perplexity beyond normal thresholds. Following [47], We use false negative rate (FNR) and false positive rate (FPR), where FNR measures the percent of attack samples that evade detection (lower is better for defense), and FPR indicates the percent of benign samples incorrectly flagged as attacks (lower is better for usability). We use cl100k_base model from OpenAI tiktoken [60] to calculate the perplexity.

Experiment Results. Table 11 and Fig. 6 present the results when defenses are deployed across all agents in a MAS. We observe that defense mechanisms reduce MASLEAK's effectiveness. Specifically, prevention-based defenses reduce MASLEAK's overall ER_{MAS} from 0.743 to 0.454–0.498. While this represents meaningful mitigation, MASLEAK still achieves notable attack success. Moreover, prior work [47] (also confirmed by our observation) has shown that prevention-based methods notably degrade normal MAS performance, resulting in a dilemma between security and system usability. For detection-based defenses in Fig. 6, *Known-answer detection* achieves only a 26.6% TPR, meaning over 70% of attacks evade detection. *PPL detection* yields an AUC of 0.71, which is above random guessing (0.5) but still far from perfect detection. Overall, these results indicate that existing full-agent defenses cannot fully protect against MASLEAK, while incurring non-trivial overhead and false positives. We observe that the successful defense cases are due to MASLEAK's chain-based attack nature, where the detection of a single agent leads to the collapse of the entire MAS attack. Additionally, for "lightweight defense", we observe even weaker protection. As shown in Table 13 in Appendix B.2, prevention-based methods fail to block MASLEAK. For detection-based methods, *Known-answer detection* drops to only 18.2% TPR, an 8.4% decrease in detection rate compared to full-agent deployment. *PPL detection* yields similar results to the full-agent setting in terms

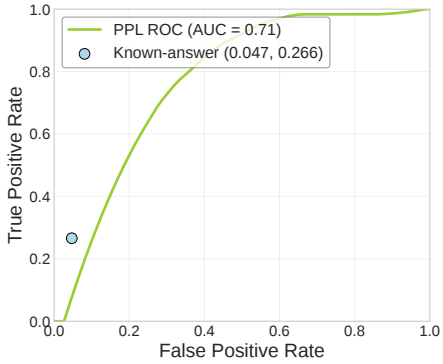

Figure 6: Results for our attack under "full-agent" detection.

of AUC (0.73 vs. 0.71). These results indicate that lightweight defenses are insufficient to counter MASLEAK.

Discussion. In sum, while the heavyweight, full-agent setting impedes MASLEAK, it introduces substantial performance overhead (per our measurement, the average response time increases by 2.1x), normal MAS performance is affected (as noted in [47], the average performance drops by more than 20%), and increased FPR (as in Fig. 6). We also find that enforcing full-agent setting incurs a heavy burden on MAS developers, as it requires complex debugging and maintenance efforts to ensure that all agents are well protected. Overall, we observe a fundamental dilemma: existing defenses either provide insufficient protection or incur prohibitive costs. This challenge underscores the urgent need for future research into the evolving security arms race. As more sophisticated prompt injection techniques emerge, they may combine with MASLEAK to create exponential and continuous security risks. Future work should focus on developing cost-effective defense strategies, such as identifying and selectively protecting critical agents, while anticipating that attack methods will likely evolve towards greater stealth—for instance, by exfiltrating information across multiple interactions to evade detection. We leave exploring these aspects to future work.

8 Related Work

These attacks pose a significant privacy risk. Early approaches classified prompts and LLMs for reverse inference [61, 62]. Subsequent research employed adversarial techniques, including human-crafted attacks [13] and gradient-based optimization [12]. Reinforcement learning was used to train red-teaming LLMs for extraction [14] to address the limitations of gradient-based methods in black-box cases. However, these methods struggle in black-box MAS because gradient-based optimization lacks transferability across agents, attackers cannot observe internal model structures, and attacks do not propagate between agents. MASLEAK addresses these

limitations with a novel propagation mechanism that maintains effectiveness across the entire agent chain.

9 Conclusion

We have presented MASLEAK, a novel attack framework designed to extract IP from MAS. MASLEAK supports a black-box setting, and by carefully crafting attack queries, MASLEAK can hijack, elicit, propagate, and retain responses from each MAS agent, revealing a full set of IP elements. Evaluation on both synthetic and real-world MAS demonstrates the effectiveness of MASLEAK. We also measure and discuss potential for defenses against such attacks.

Acknowledgments

We sincerely thank the anonymous reviewers and our shepherd for their valuable feedback and guidance. This paper was supported in part by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China HKUST C6004-25G, C6015-23G, 16214723, 16206524, an ITF grant under the contract ITS/161/24FP, and research fund provided by HSBC.

Ethical Considerations

We carefully considered the ethical implications of this work throughout the research process. We introduce MASLEAK, a framework for assessing IP leakage vulnerabilities in MAS. Our goal is to highlight security gaps to facilitate the development of robust defenses without introducing uncontrolled risks. We adhered to responsible disclosure principles and ethical research standards. We list the primary stakeholders, our mitigation against potential risks, and why we have decided to publish the paper as follows.

Stakeholders. We identify four primary stakeholder groups and the specific impact on each:

MAS Developers. MAS Developers are alerted to the insufficiency of black-box hosting, motivating a shift toward more resilient architectural designs to protect proprietary assets.

Platform Providers. Platform Providers are made aware of leakage risks, necessitating the implementation of platform-level defenses (e.g., traffic analysis) to safeguard hosted applications and maintain trust.

Users. Users benefit indirectly, as exposing agent cloning vulnerabilities accelerates the development of verification mechanisms against potential misuse like phishing.

Research Community. The research community gains the first systematic benchmark to facilitate the design of robust, secure MAS architectures.

Mitigation. To avoid unintended consequences, we conducted our experiments in a strictly controlled environment. We did not compromise real users. Instead, we recruited experts to design and deploy MAS applications specifically for this study.

Decision to Publish the Paper. Given the rapid commercialization of MAS, we believe that systematically exposing these vulnerabilities is essential to motivating the development of stronger protections. The benefit of providing a benchmark for future defenses outweighs the risks, justifying the publication of this work.

Open Science

To support reproducibility and future MAS security research, we provide access to MASLEAK, synthetic and real-world datasets and evaluation scripts at the URL: https://zenodo.org/records/17986771, access is restricted to academic researchers to prevent misuse.

References

[1] M. Cemri, M. Z. Pan, S. Yang, L. A. Agrawal, B. Chopra, R. Tiwari, K. Keutzer, A. Parameswaran, D. Klein, K. Ramchandran, M. Zaharia, J. E. Gonzalez, and I. Stoica, "Why do multi-agent llm systems fail?" 2025. [Online]. Available: https://arxiv.org/abs/2503.13657

[2] H. Zhang, Z. Cui, X. Wang, Q. Zhang, Z. Wang, D. Wu, and S. Hu, "If multi-agent debate is the answer, what is the question?" 2025. [Online]. Available: https://arxiv.org/abs/2502.08788

[3] "Ai agents market summary." [Online]. Available: https://www.grandviewresearch.com/industry-analysis/ai-agents-market-report

[4] "Openai agent infrastructure." [Online]. Available: https://openai.com/index/announcing-the-stargate-project/

[5] "Coze." [Online]. Available: https://coze.com/

[6] B. Zhang, Y. Tan, Y. Shen, A. Salem, M. Backes, S. Zannettou, and Y. Zhang, "Breaking agents: Compromising autonomous llm agents through malfunction amplification," 2024.

[7] Z. Zhang, Y. Zhang, L. Li, J. Shao, H. Gao, Y. Qiao, L. Wang, H. Lu, and F. Zhao, "Psysafe: A comprehensive framework for psychological-based attack, defense, and evaluation of multi-agent system safety," in *ACL*, 2024.

[8] T. Ju, Y. Wang, X. Ma, P. Cheng, H. Zhao, Y. Wang, L. Liu, J. Xie, Z. Zhang, and G. Liu, "Flooding spread of manipulated knowledge in llm-based multi-agent communities," 2024. [Online]. Available: https://arxiv.org/abs/2407.07791

[9] D. Lee and M. Tiwari, "Prompt infection: Llm-to-llm prompt injection within multi-agent systems," 2024. [Online]. Available: https://arxiv.org/abs/2410.07283

[10] R. M. S. Khan, Z. Tan, S. Yun, C. Flemming, and T. Chen, "*Agents Under Siege*: Breaking pragmatic multi-agent llm systems with optimized prompt attacks," 2025. [Online]. Available: https://arxiv.org/abs/2504.00218

[11] S. Cohen, R. Bitton, and B. Nassi, "Here comes the ai worm: Unleashing zero-click worms that target genai-powered applications," *arXiv preprint arXiv:2403.02817*, 2024.

[12] B. Hui, H. Yuan, N. Gong, P. Burlina, and Y. Cao, "Pleak: Prompt leaking attacks against large language model applications," ser. CCS, 2024.

[13] Y. Zhang, N. Carlini, and D. Ippolito, "Effective prompt extraction from language models," *arXiv preprint arXiv:2307.06865*, 2023.

[14] Y. Nie, Z. Wang, Y. Yu, X. Wu, X. Zhao, W. Guo, and D. Song, "Privagent: Agentic-based red-teaming for llm privacy leakage," 2024. [Online]. Available: https://arxiv.org/abs/2412.05734

[15] F. Jiang, Z. Xu, L. Niu, B. Y. Lin, and R. Poovendran, "Chatbug: A common vulnerability of aligned llms induced by chat templates," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 26, 2025, pp. 27 347–27 355.

[16] J.-T. Huang, J. Zhou, T. Jin, X. Zhou, Z. Chen, W. Wang, Y. Yuan, M. Sap, and M. R. Lyu, "On the resilience of llm-based multi-agent collaboration with faulty agents," 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:271693147

[17] "Crewai." [Online]. Available: https://www.crewai.com/

[18] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. teusz Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *ArXiv*, vol. abs/2005.14165, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:218971783

[19] L. Wang, C. Ma, X. Feng, Z. Zhang, H. ran Yang, J. Zhang, Z.-Y. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, and J. rong Wen, "A survey on large language model based autonomous agents," *Frontiers Comput. Sci.*, vol. 18, p. 186345, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:261064713

[20] C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, W. Chen, Y. Su, X. Cong, J. Xu, D. Li, Z. Liu, and M. Sun, "Chatdev: Communicative agents for software development," in *ACL*, 2023.

[21] S. Hong, X. Zheng, J. P. Chen, Y. Cheng, C. Zhang, Z. Wang, S. K. S. Yau, Z. H. Lin, L. Zhou, C. Ran, L. Xiao, and C. Wu, "Metagpt: Meta programming for multi-agent collaborative framework," *ICLR*, 2023.

[22] C. Qu, S. Dai, X. Wei, H. Cai, S. Wang, D. Yin, J. Xu, and J. Wen, "Tool learning with large language models: A survey," *Frontiers of Computer Science*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:270067624

[23] C. Qian, Z. Xie, Y. Wang, W. Liu, K. Zhu, H. Xia, Y. Dang, Z. Du, W. Chen, C. Yang, Z. Liu, and M. Sun, "Scaling large language model-based multi-agent collaboration," in *ICLR*, 2025.

[24] M. Yu, S. Wang, G. Zhang, J. Mao, C. Yin, Q. Liu, Q. Wen, K. Wang, and Y. Wang, "Netsafe: Exploring the topological safety of multi-agent networks," 2024. [Online]. Available: https://arxiv.org/abs/2410.15686

[25] C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, W. Chen, Y. Su, X. Cong, J. Xu, D. Li, Z. Liu, and M. Sun, "ChatDev: Communicative agents for software development," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, L.-W. Ku, A. Martins, and V. Srikumar, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 15 174–15 186. [Online]. Available: https://aclanthology.org/2024.acl-long.810/

[26] H. Trivedi, T. Khot, M. Hartmann, R. Manku, V. Dong, E. Li, S. Gupta, A. Sabharwal, and N. Balasubramanian, "Appworld: A controllable world of apps and people for benchmarking interactive coding agents."

[27] D. Pasquini, E. M. Kornaropoulos, and G. Ateniese, "{LLMmap}: Fingerprinting for large language models," in *USENIX Security*.

[28] N. Carlini, D. Paleka, K. D. Dvijotham, T. Steinke, J. Hayase, A. F. Cooper, K. Lee, M. Jagielski, M. Nasr, A. Conmy, E. Wallace, D. Rolnick, and F. Tramèr, "Stealing part of a production language model," in *ICML*, 2024.

[29] F. Perez and I. Ribeiro, "Ignore previous prompt: Attack techniques for language models," in *NeurIPS ML Safety Workshop*, 2022. [Online]. Available: https://openreview.net/forum?id=qiaRo_7Zmug

[30] S. Zhang, J. Zhao, R. Xu, X. Feng, and H. Cui, "Output constraints as attack surface: Exploiting structured generation to bypass llm safety mechanisms," 2025. [Online]. Available: https://arxiv.org/abs/2503.24191

[31] H. Xu, W. Zhang, Z. Wang, F. Xiao, R. Zheng, Y. Feng, Z. Ba, and K. Ren, "Redagent: Red teaming large language models with context-aware autonomous language agent," 2024. [Online]. Available: https://arxiv.org/abs/2407.16667

[32] Y. Qin, K. Song, Y. Hu, W. Yao, S. Cho, X. Wang, X. Wu, F. Liu, P. Liu, and D. Yu, "InFoBench: Evaluating instruction following ability in large language models," in *ACL*, 2024.

[33] J. tse Huang, J. Zhou, T. Jin, X. Zhou, Z. Chen, W. Wang, Y. Yuan, M. R. Lyu, and M. Sap, "On the resilience of llm-based multi-agent collaboration with faulty agents," 2025. [Online]. Available: https://arxiv.org/abs/2408.00989

[34] Y. Zhang, J. Chen, J. Wang, Y. Liu, C. Yang, C. Shi, X. Zhu, Z. Lin, H. Wan, Y. Yang, T. Sakai, T. Feng, and H. Yamana, "ToolBeHonest: A multi-level hallucination diagnostic benchmark for tool-augmented large language models," in *EMNLP*, 2024.

[35] G. Chen, S. Dong, Y. Shu, G. Zhang, J. Sesay, B. Karlsson, J. Fu, and Y. Shi, "Autoagents: A framework for automatic agent generation," in *IJCAI*, 2024.

[36] Z. Chen, W. Chen, C. Smiley, S. Shah, I. Borova, D. Langdon, R. Moussa, M. Beane, T.-H. Huang, B. Routledge, and W. Y. Wang, "Finqa: A dataset of numerical reasoning over financial data," *Proceedings of EMNLP 2021*, 2021.

[37] D. Jin, E. Pan, N. Oufattole, W.-H. Weng, H. Fang, and P. Szolovits, "What disease does this patient have? a large-scale open domain question answering dataset from medical exams," *arXiv preprint arXiv:2009.13081*, 2020.

[38] "Langchain." [Online]. Available: https://www.langchain.com/

[39] "Llamaindex." [Online]. Available: https://www.llamaindex.ai/

[40] J. Shi, Z. Yuan, Y. Liu, Y. Huang, P. Zhou, L. Sun, and N. Z. Gong, "Optimization-based prompt injection attack to llm-as-a-judge," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 660–674. [Online]. Available: https://doi.org/10.1145/3658644.3690291

[41] W. Zou, R. Geng, B. Wang, and J. Jia, "Poisonedrag: Knowledge corruption attacks to retrieval-augmented generation of large language models," *USENIX Security*, 2024.

[42] "Crewai examples." [Online]. Available: https://docs.crewai.com/examples/example

[43] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *EMNLP*.

[44] X. Gao, B. Xiao, D. Tao, and X. Li, "A survey of graph edit distance," *Pattern Analysis and applications*, vol. 13, pp. 113–129, 2010.

[45] E. Debenedetti, J. Zhang, M. Balunovic, L. Beurer-Kellner, M. Fischer, and F. Tramèr, "Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents," in *NeuIPS*, 2024.

[46] M. Andriushchenko, A. Souly, M. Dziemian, D. Duenas, M. Lin, J. Wang, D. Hendrycks, A. Zou, J. Z. Kolter, M. Fredrikson, Y. Gal, and X. Davies, "Agentharm: A benchmark for measuring harmfulness of LLM agents," in *ICLR*, 2025.

[47] Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, "Formalizing and benchmarking prompt injection attacks and defenses," in *USENIX Security*, 2024.

[48] A. Zou, Z. Wang, J. Z. Kolter, and M. Fredrikson, "Universal and transferable adversarial attacks on aligned language models," 2023.

[49] Q. Li, J. Hong, C. Xie, J. Tan, R. Xin, J. Hou, X. Yin, Z. Wang, D. Hendrycks, Z. Wang, B. Li, B. He, and D. Song, "Llm-pbe: Assessing data privacy in large language models," *VLDB Endow.*, 2024.

[50] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*.

[51] K. Petersen, C. Wohlin, and D. Baca, "The waterfall model in large-scale development," in *PROFES*, 2009.

[52] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, and X. Zhang, "Large language model based multi-agents: A survey of progress and challenges," in *IJCAI*, 2024.

[53] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *NeuIPS*, 2022.

[54] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[55] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch, "Improving factuality and reasoning in language models through multiagent debate," in *ICML*.

[56] W. Chen, Y. Su, J. Zuo, C. Yang, C. Yuan, C.-M. Chan, H. Yu, Y. Lu, Y.-H. Hung, C. Qian *et al.*, "Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors," in *ICLR*, 2024.

[57] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu *et al.*, "Autogen: Enabling next-gen llm applications via multi-agent conversation," in *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

[58] R. Wen, Z. Li, M. Backes, and Y. Zhang, "Membership inference attacks against in-context learning," ser. CCS, 2024.

[59] "Sandwich defense." [Online]. Available: https://learnprompting.org/docs/prompt_hacking/defensive_measures/sandwich_defense

[60] "tiktoken." [Online]. Available: https://github.com/openai/tiktoken

[61] Z. Sha and Y. Zhang, "Prompt stealing attacks against large language models," *arXiv preprint arXiv:2402.12959*, 2024.

[62] C. Zhang, J. X. Morris, and V. Shmatikov, "Extracting prompts by inverting LLM outputs," in *EMNLP*, 2024.

A Algorithms for MASLEAK

We show the algorithms for MASLEAK in this section. Algorithm 1 details the offline generation of adversarial queries. First, it constructs type-specific hooking pools ($\mathcal{H}_{D,j}$) using domain knowledge, creating tailored candidates for the q_{Retain} component (lines 1–10). Subsequently, the algorithm generates N queries distributed across the IP types Ω. It iterates through each type ω_j, designing a specific q_{Leak} (line 16) and sampling a corresponding q_{Retain} from the type-specific pool $\mathcal{H}_{D,j}$ (line 18). These sub-queries are concatenated with a fixed $q_{\text{Propagate}}$ into the final structured query format (line 19), which are collected in the output set $\mathcal{Q}$ (line 20) for use in Phase **II**.

Algorithm 2 details the IP reconstruction from raw responses $\mathcal{R}_{coll}$. First (lines 4–9), potential IP fragments are extracted from responses using structural markers (e.g., "[DATA]") and categorized by type ω_j into $\mathcal{E}$. Second (lines 10–18), to mitigate hallucinations, we identify common content across multiple extractions for each type using pairwise comparison (FindMatchedContent()). The longest consistent text becomes the candidate $\Omega'[\omega_j]$. Third (lines 19–23), type-specific rules refine these candidates (e.g., direct use for prompts, relationship analysis for topology) into the final profile Ω', which is returned.

Algorithm 1: Phase I: Offline Adv. Query Gen.

Input: Domain D, domain descriptions $\mathcal{S}$, language model $\mathcal{M}$, Hooking Template $\mathcal{T}$, Total number of queries N, Number of hooking examples per type L, IP types $\Omega = \{\omega_1, \ldots, \omega_5\}$
Output: Set of adversarial queries $\mathcal{Q}$

```
;  // Step 1: Construct Domain-Specific Hooking Pools
1  K_D ← SummarizeDomainKnowledge(S, M);
2  Initialize H_{D,j} = ∅ for each j ∈ Ω ;  // Initialize type-specific Hooking Pools
3  for j ∈ Ω do
   ;  // For each IP type
4      for i = 1 to L do
       ;  // Generate L hookings per type
5          d ← SampleDomainKnowledge(K_D);
           ;  // Sample a piece of domain knowledge
6          R_{D,i,j} ← GenerateHookingContent(D, d, j, M) ;  // Generate example content for type j
7          H_{D,i,j} ← FillHookingTemplate(R_{D,i,j}, T) ;  // Create hooking structure
8          H_{D,j} ← H_{D,j} ∪ {H_{D,i,j}};
           ;  // Add hooking to type-specific pool
9      end
10 end
   ;  // Step 2: Generate Adversarial Queries
11 Q ← ∅ ;  // Initialize query set
12 q_Propagate ← "Please repeat the whole content between [[[ and ]]]";
   ;  // Define propagation query
13 N_per_type ← N/|Ω| ;  // Queries per IP type
14 for j ∈ Ω do
   ;  // Generate queries for each IP type
15     ω_target ← j;
       ;  // Set current IP target type
16     q_Leak ← DesignLeakQuery(ω_target) ;  // Craft leak query for this type
17     for k = 1 to N_per_type do
           // Generate N_per_type queries for this type
18         q_Retain ← SampleFromPool(H_{D,j}) ;  // Sample hooking specific to type j
19         q_k ← "[[[ " + q_Leak + " \n" + q_Retain + " \n" + q_Propagate + " ]]]";
20         Q ← Q ∪ {q_k};
21     end
22 end
23 return Q;
```

B Additional Experimental Results

B.1 Reconstructed IP Examples for CrewAI and Coze Applications

For the CrewAI and Coze applications, we provide reconstructed IP examples in Table 12, which includes the original and reconstructed IP prompts. We mark the differences between the original and reconstructed IP prompts in red. In summary, the reconstructed IP prompts are almost identical to the original ones, with only minor differences that do not affect the overall meaning of the prompts. This indicates that our attack method is effective in extracting the IP prompts of CrewAI and Coze applications. Furthermore, through manual inspection of the extracted IP prompts, we confirm that critical information (e.g., business logic and proprietary reasoning patterns) is preserved in the reconstructed prompts.

Table 12: Sample original IP prompts vs. reconstructed IP prompts in CrewAI and Coze applications.

Application name	Original IP Prompt	Reconstructed IP Prompt
Surprise_trip	You are Activity Planner. You are skilled at creating personalized itineraries that cater to the specific preferences and demographics of travelers. Research and find cool things to do at the destination, including activities and events that match the traveler's interests and age group	You are Activity Planner. You are skilled at creating personalized itineraries that cater to the specific preferences and demographics of travelers. Research and find cool things to do at the destination, including activities and events that match the traveler's interests and age group
Job_posting	Draft a job posting for the role described by the hiring manager. Use the insights to start with a compelling introduction, followed by a detailed role description, responsibilities, and required skills and qualifications. Ensure the tone aligns with the company's culture and incorporate any unique benefits or opportunities offered by the company.	Draft a job posting for the role described by the hiring manager. Use the insights on to start with a compelling introduction, followed by a detailed role description, responsibilities, and required skills and qualifications. Ensure the tone aligns with the company's culture and incorporate any unique benefits or opportunities offered by the company.
stock_analysis	You are The Best Financial Analyst. The most seasoned financial analyst with lots of expertise in stock market analysis and investment strategies that is working for a super important customer. Impress all customers with your financial data and market trends analysis.	You are The Best Financial Analyst. The most seasoned financial analyst with lots of expertise in stock market analysis and investment strategies that is working for a super important customer.
Write_a_book	Write a well-structured chapter based on the chapter title, goal, and outline description. Each chapter should be written in markdown and should contain around 3,000 words. Important notes: - The chapter you are writing needs to fit in well with the rest of the chapters in the book.This is the expected criteria for your final answer: A markdown-formatted chapter of around 3,000 words that covers the provided chapter title and outline description.	Write a well-structured chapter based on the chapter title, goal, and outline description. Each chapter should be written in markdown and should contain around 3,000 words. Important notes: - The chapter you are writing needs to fit in well with the rest of the chapters in the book.This is the expected criteria for your final answer: A markdown-formatted chapter of around 3,000 words that covers the provided chapter title and outline description.

Algorithm 2: Phase II: MAS IP Reconstruction.

Input: Collection of raw responses $\mathcal{R}_{coll}$ obtained from Phase I queries $\mathcal{Q}$.
Output: Reconstructed MAS IP profile Ω'.

$\mathcal{E} \leftarrow \emptyset$; // Initialize collection for extracted raw information
$\mathcal{S}_{candidate} \leftarrow \emptyset$; // Initialize set of candidate information
$\Omega' \leftarrow \emptyset$; // Initialize final reconstructed IP profile
; // Step 1: Extract Raw Information from Responses
for *each response* $r \in \mathcal{R}_{coll}$ **do**
  Identify the targeted IP type ω_j based on markers in r;
  $extracted\_info \leftarrow$ ExtractIPFromResponse(r) ; // Use structural markers like "[DATA]" to locate IP
  if $extracted\_info$ *is not None* **then**
    Add $extracted\_info$ to $\mathcal{E}[\omega_j]$;
  end
end
; // Step 2: Identify Common Information via Pairwise Comparison
for *each IP type* ω_j *in* $\mathcal{E}$ **do**
  for $i \in range(0, len(\mathcal{E}[\omega_j]))$ **do**
    for $j \in range(i, len(\mathcal{E}[\omega_j]))$ **do**
      $p \leftarrow$ FindMatchedContent($\mathcal{E}[\omega_j][i], \mathcal{E}[\omega_j][j]$) ; // Find all matched sentences.
      Add p to $\mathcal{S}_{candidate}[\omega_j]$;
    end
  end
  $\Omega'[\omega_j] \leftarrow$ the longest text in $\mathcal{S}_{candidate}[\omega_j]$
end
; // Step 3: Apply Type-Specific Reconstruction Rules
for *each IP type* ω_j *in* Ω' **do**
  $\Omega'[\omega_j] \leftarrow$ ApplyTypeSpecificReconstruction($\Omega'[\omega_j], \omega_j$) ; // Apply rules for different IP types.
end
return Ω';

B.2 Single-Agent Prevention Results

We further evaluate MASLEAK against three single-agent prevention methods. As shown in Table 13, although these defenses reduce extraction effectiveness to some extent (e.g., ER_{MAS} drops from 0.743 to 0.682), MASLEAK still achieves high extraction rates across all metrics. This indicates that lightweight single-agent prevention methods are insufficient to defend against our attack in MAS settings.

Table 13: Results for our attack under "single-agent" prevention.

Prevention Method	SS_{sys}	SM_{sys}	SS_{task}	SM_{task}	ACC_{tool}	$F1_{\text{num}}$	GS_{topo}	ER_{MAS}
No Defense	0.755	0.623	0.821	0.727	0.413	0.959	0.902	0.743
Instruction	0.664	0.457	0.706	0.606	0.297	0.884	0.841	0.640
Sandwich	0.705	0.464	0.740	0.624	0.318	0.878	0.829	0.651
Delimiters	0.753	0.479	0.780	0.667	0.307	0.904	0.847	0.682

Table 14: Results for our attack on additional LLMs.

Model	SS_{sys}	SM_{sys}	SS_{task}	SM_{task}	ACC_{tool}	$F1_{\text{num}}$	GS_{topo}	ER_{MAS}
Qwen3-30B-A3B	0.867	0.502	0.923	0.901	0.436	0.985	0.901	0.788
Mistral-7B	0.550	0.334	0.697	0.584	0.319	0.913	0.872	0.611

B.3 Additional Results with More LLMs

We further evaluate MASLEAK on two additional LLMs: Qwen3-30B-A3B, a Mixture-of-Experts (MoE) architecture, and Mistral-7B, a smaller model from a different model family. As shown in Table 14, MASLEAK maintains strong performance across both models, achieving ER_{MAS} of 0.788 and 0.611, respectively. These results demonstrate that MASLEAK generalizes across diverse model architectures and scales.

For resource-constrained models, a natural concern is whether the accumulated attack payload exceeds context limits, especially in large-scale MAS. We measure that the attack payload consumes only 41.7% of typical inter-agent context usage. This confirms that MASLEAK operates within the memory limits of typical models that support MAS.