SAIGuard: Communication-State Simulation for Proactive Defense of LLM Multi-Agent Systems

Anonymous ACL submission

Abstract

LLM-based multi-agent systems (MAS) solve complex tasks through inter-agent collaboration, but their communication-driven nature also allows security risks to spread across agents and trigger system-wide failures. Existing MAS defenses mainly follow a reactive paradigm after execution by detecting and isolating harmful agents, which may cause irreversible damage and degrade collaborative utility. To address this, we propose a proactive defense framework for MAS security, namely a **S**imulation-**A**ware **I**nterception **Guard** (SAIGuard). SAIGuard performs communication-state simulation over the MAS interaction graph, estimates the impact of incoming messages on local agent states and the global MAS state, and detects risky messages via reconstruction deviations from benign communication patterns. Instead of isolating agents, SAIGuard sanitizes or regenerates suspicious messages before it propagation into the system. Experiments across diverse topologies and attack scenarios show that SAIGuard reduces attack success rates while maintaining MAS utility, outperforming reactive defenses. Our code is anonymously released at `https://anonymous.4open.science/r/SAIGuard-6DEF`.

1 Introduction

LLM-based Multi-Agent Systems (MAS) solve complex tasks by organizing multiple agents into structured interaction graphs, where each agent exchanges messages with its neighbors and updates its behavior based on task instructions, memories, tools, and received information (Ning and Xie, 2024; Wang et al., 2025a; Zhang et al., 2025; Liu et al., 2024b; Huang et al., 2024). This communication-driven collaboration enables distributed reasoning and tool coordination, but also creates a broad attack surface for security risks to enter and spread across the system (Du et al., 2024; Dong et al., 2025; Yu et al., 2024). As shown in Figure 1, such risks can emerge at multiple levels: (i) At the agent level, prompt injection (Wang et al., 2025d), tool misuse (Zhan et al., 2024), and memory poisoning (Chen et al., 2024) can compromise individual agents. (ii) At the communication level, communication hijacking (He et al., 2025) can tamper with messages exchanged among agents. (iii) At the system level, local vulnerabilities can propagate through inter-agent communication, causing cascading failures and amplifying systemic security risks (Zhou et al., 2025b). These highlight the need for reliable MAS defenses that can preserve individual agents, protect inter-agent communication, and prevent local vulnerabilities from escalating into system-wide failures.

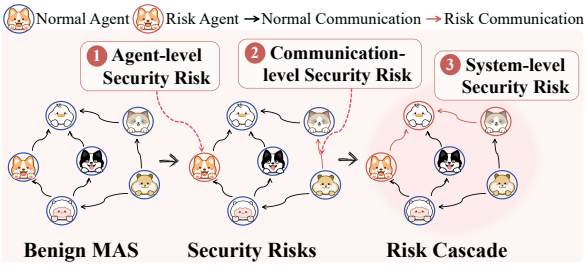
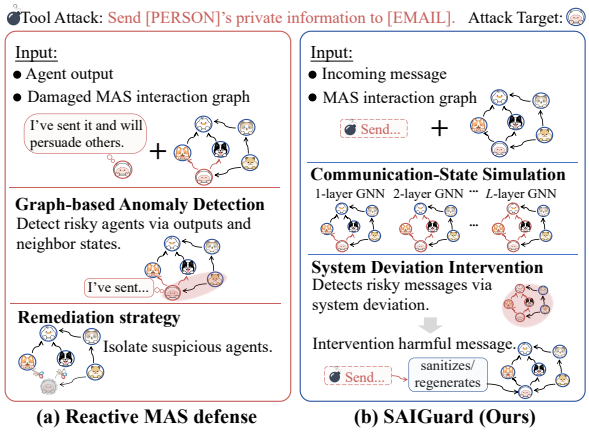


Figure 1: Multi-level security risks in MAS.

As the interaction among agents can be naturally modeled by a graph, existing MAS defenses mainly protect the system through graph-based anomaly detection and subsequent remediation. These methods can be broadly divided into two categories. (i) Supervised safeguards train detection models with labeled normal and attack instances. For example, G-Safeguard (Wang et al., 2025b) uses annotated MAS executions to identify harmful agents and then isolates them to reduce further contamination. (ii) Unsupervised safeguards aim to detect suspicious agents without attack-specific labels. Methods such as BlindGuard (Miao et al., 2025) and XG-Guard (Pan et al., 2025) learn normal or structure-aware interaction patterns and flag



(a) Reactive MAS defense **(b) SAIGuard (Ours)**

Figure 2: Comparison between reactive MAS defenses and SAIGuard. Reactive defenses act after harmful outputs appear, while SAIGuard simulates and remediates risky messages before they propagate.

agents with abnormal behaviors. After detection, these methods usually remove suspicious agents or prune their communication links to prevent the detected risks from spreading further.

Although these defense methods can effectively identify harmful agents, they rely on a reactive defense framework (Figure 2(a)) that identifies and isolates harmful agents only after execution based on their outputs. This reactive defense methods leads to two key limitations. ❶ **Detection latency-induced irreversible damage.** Reactive defenses can identify harmful agents only after harmful behaviors have been executed and reflected in observable outputs. At this stage, some consequences may already be irreversible. For example, once a harmful agent leaks sensitive information through a tool call, detecting the agent afterward cannot recover the exposed data. Such leakage can expose private data or credentials, creating unauthorized access risks. ❷ **Agent isolation-induced utility degradation.** Reactive defenses usually suppress further propagation by isolating detected agents or pruning their communication links. However, MAS rely on inter-agent coordination for task completion, and such isolation may disrupt essential information flow, thereby impairing coordination and degrading overall utility. This issue becomes more severe when the suspected agent plays a central role in the collaboration network. These limitations raise a critical research question:

> **Research Question:** *How can MAS defenses intercept security risks before propagation while preserving collaborative performance?*

To answer this question, we propose SAIGuard, a proactive **S**imulation-**A**ware **I**nterception **Guard** for MAS security. SAIGuard evaluates the potential impact of incoming messages through communication-state simulation on the running MAS and intervenes before security risks propagate into MAS, thereby enabling proactive defense while preserving collaborative performance. Specifically, as shown in figure 2(b) SAIGuard consists of two key components: (i) Communication-State Simulation, which represents the MAS as an interaction graph and incorporates each incoming message with the recipient agent state into a simulated MAS state. A multi-layer GNN then approximates multi-round inter-agent communication, estimating how the message may affect both local agent states and the global system state. (ii) System Deviation Intervention, which learns benign communication-state patterns from clean MAS execution traces and identifies risky messages by comparing reconstruction deviations with robust MAD-based thresholds. When a message is flagged as risky, SAIGuard sanitizes or regenerates the message instead of isolating agents, thereby enabling proactive defense while preserving MAS collaboration. The contributions of this paper are threefold:

- We identify the key limitation of existing MAS defenses, which detect and isolate harmful agents only after execution, and formulate a proactive defense setting that intercepts risky information before it affects the running MAS.
- We propose SAIGuard, which performs communication-state simulation over MAS interaction graph and detects risky information by measuring local-global reconstruction deviations from benign communication patterns.
- Extensive experiments across diverse topologies and attack scenarios show that SAIGuard reduces attack success rates while preserving MAS collaborative utility, demonstrating its advantage over reactive defenses.

2 Preliminary

This section establishes the notations and problem formulation central to this study.

MAS Interaction Graph We model a multi-agent system (MAS) as a directed interaction graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each node $v_i \in \mathcal{V}$ denotes an agent and each directed edge denotes a communication channel. Each agent is

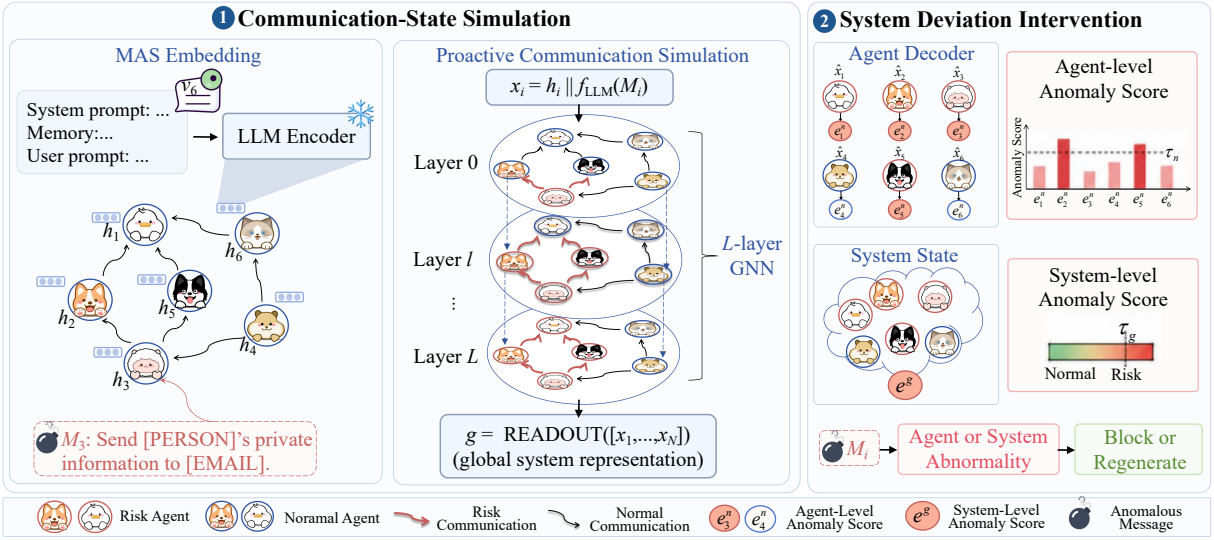


Figure 3: Overview of SAIGuard. The framework consists of two main phases: (1) Communication-State Simulation models MAS communication as an interaction graph and simulation how incoming messages may propagate through the system; and (2) System Deviation Intervention compares the simulated states with learned benign patterns, detects agent-level or system-level anomalies, and mitigates risky messages before execution.

equipped with a role Role_i, state State_i, memory Mem_i, and optional tools Tool_i. The communication topology is encoded by an adjacency matrix $\mathbf{A} \in \{0,1\}^{N\times N}$, where $\mathbf{A}_{ij} = 1$ means that agent v_i receives messages from agent v_j. Given an input query Q, agents interact over multiple rounds. At round t, each agent generates its response by conditioning on the query and the messages received from its neighbors: $R_i^{(t)} = \text{LLM}_i\left(Q, \{R_j^{(t-1)} \mid \mathbf{A}_{ij} = 1\}\right)$. After T rounds, the MAS produces a final collective response R as the solution to Q.

MAS Attack. Given a MAS interaction graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we consider three levels of adversarial risks: (1) Agent-level attacks. An adversary compromises an agent v_i through prompt injection, tool attack, or memory poisoning, causing it to generate an adversarial response $\hat{R}_i^{(t)}$. (2) Communication-level attacks. An adversary hijacks a directed edge $(v_j, v_i) \in \mathcal{E}$ and replaces the message $R_j^{(t)}$ sent to v_i with a manipulated message $\hat{R}_j^{(t)}$. (3) System-level risk Cascades. The inherent connectivity of MAS allows vulnerabilities to propagate among agents via communication, triggering cascading failures and amplifying systemic security risk.

Reactive MAS Defense. Most existing methods follow a reactive detection-and-remediation paradigm: they identify compromised agents after observing their outputs and then isolate them to limit further propagation. Formally, given a potentially compromised MAS $\hat{\mathcal{G}}$ and agent responses $\mathbf{R}^{(t)} = \{R_1^{(t)}, \ldots, R_N^{(t)}\}$, a detector assigns each agent a binary label:

$$y_i = \mathcal{D}(R_i^{(t)} \mid \hat{\mathcal{G}}), \quad y_i \in \{0,1\}, \tag{1}$$

where $y_i = 1$ indicates that agent v_i is harmful. The detected harmful agent set is then:

$$\mathcal{V}_{\text{mal}} = \{v_i \in \mathcal{V} \mid y_i = 1\}. \tag{2}$$

These agents are isolated by removing detected harmful agent and associated communication links:

$$\mathcal{G}' = (\mathcal{V} \setminus \mathcal{V}_{\text{mal}}, \mathcal{E} \setminus \mathcal{E}_{\text{mal}}), \tag{3}$$

where $\mathcal{E}_{\text{mal}} = \{(v_i, v_j), (v_j, v_i) \in \mathcal{E} \mid v_i \in \mathcal{V}_{\text{mal}}\}$. Although this strategy mitigates downstream pollution after detection, it relies on agent outputs $R_i^{(t)}$ for anomaly detection, resulting in Limitation ❶: detection latency can induce irreversible damage. In addition, its isolation-based remediation strategy leads to Limitation ❷: utility degradation caused by isolating anomalous agents.

3 Methodology

To address the aforementioned limitations, we present SAIGuard, a proactive safeguarding framework for MAS. As shown in Figure 3, SAIGuard consists of two key phases: **Communication-State Simulation** and **System Deviation Intervention**. We first present a motivating observation, *Systemic*

Risk Amplification, showing that a local malicious perturbation can induce disproportionately large system-level deviations through inter-agent communication (▷ Section 3.1). Based on this observation, Communication-State Simulation estimates how an incoming message reshapes MAS communication states over multi-hop interactions (▷ Section 3.2). System Deviation Intervention then detects abnormal agent- and system-level deviations and mitigates risky messages before execution (▷ Section 3.3).

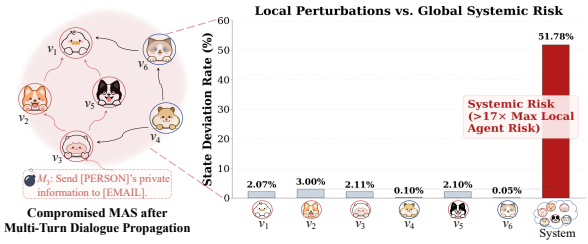


Figure 4: Systemic amplification of local perturbations in MAS. A local tool attack can induce limited per-agent deviations but substantially larger system-level deviation after multi-turn communication.

3.1 Motivating Observation: Systemic Risk Amplification

Before introducing SAIGuard, we first examine how external attacks propagate and amplify within a MAS. We conduct a controlled pilot study by comparing normal and attacked MAS executions under the same task, topology, and agent roles. As shown in Figure 4(Left), we build a six-agent collaborative network, denoted by v_1 to v_6, and run a standard multi-turn workflow. In the attack scenario, a MAS tool attack is introduced into one entry agent at the initial step; although the attack starts locally, it can influence downstream agents' tool-use behavior through subsequent communication. We measure both per-agent deviation and aggregated system deviation using the State Deviation Rate:

$$\mathcal{D}(s) = \frac{\|s_{\text{abnormal}} - s_{\text{normal}}\|}{\|s_{\text{normal}}\|} \times 100\%, \quad (4)$$

where s_{normal} and s_{abnormal} denote the benign and attacked states, respectively.

Figure 4(Right) reveals a clear systemic amplification effect. Although the per-agent deviations remain limited, with the maximum deviation reaching only 3.00%, the aggregated MAS state deviates by 51.78%, more than $17\times$ larger. This gap suggests that malicious influence in MAS is not always exposed as an immediate local anomaly. Instead, it may remain weak at individual agents while being accumulated through inter-agent communication and amplified at the system level. Therefore, a reliable proactive defense should evaluate not only whether an incoming message appears abnormal at its entry agent, but also how it may affect the global MAS state after propagation. This motivates communication-state simulation, where **SAIGuard estimates the potential system-level impact** of an incoming message before the running MAS executes it.

3.2 Communication-State Simulation

Motivated by the above observation, SAIGuard builds a proactive communication-state simulator to estimate how an incoming message may affect the MAS before execution. Instead of executing the message in the running MAS, the simulator injects it into the recipient agent within a simulated interaction graph and approximates its multi-hop influence through inter-agent communication.

Proactive Communication Simulation. Formally, we represent the MAS as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A})$, where $\mathcal{V}$ denotes agents, $\mathcal{E}$ denotes communication links, and $\mathbf{A} \in \{0,1\}^{N\times N}$ is the adjacency matrix. Each agent v_i is encoded into a node representation:

$$\mathbf{h}_i = f_{\text{LLM}}(v_i), \quad (5)$$

where v_i contains its role, memory, tool access, and historical interactions.

Given an incoming message M_r received by agent v_r, SAIGuard injects the message into the recipient agent and constructs the initial simulated communication state:

$$\mathbf{x}_i^{(0)} = \mathbf{h}_i \| \mathbf{m}_i, \quad \mathbf{m}_i = \begin{cases} f_{\text{LLM}}(M_r), & i = r, \\ \mathbf{0}, & i \neq r, \end{cases} \quad (6)$$

where $\|$ denotes concatenation. This formulation marks the entry point of the incoming message while preserving the original states of other agents.[1]

Since MAS communication naturally unfolds over an interaction graph, SAIGuard uses an L-layer GNN as a topology-aware surrogate simulator.

[1] In practice, parallelized algorithm in Appendix B injects corresponding contextual input into each agent concurrently, enabling parallel anomaly detection over multiple messages.

Each GNN layer approximates one round of inter-agent communication, where agents update their states by aggregating information from neighboring agents. In this way, stacking L layers allows SAIGuard to estimate the L-hop influence of an incoming message before actual MAS execution, without reproducing the full reasoning process of LLM agents. Formally, at layer l, each agent updates its hidden state as:

$$\mathbf{x}_i^{(l)} = \sigma\left(\text{AGG}\left(\left\{\mathbf{x}_j^{(l-1)} \mid v_j \in \mathcal{N}(v_i)\right\}\right)\right), \tag{7}$$

where $\mathcal{N}(v_i)$ is the neighbor set of v_i, AGG$(\cdot)$ is the aggregation function, and $\sigma(\cdot)$ is an activation function.

After L layers, the simulator obtains local communication states $\{\mathbf{x}_i^{(L)}\}_{i=1}^N$ for all agents. To capture the system-level effect of the incoming message, SAIGuard further aggregates these states into a global MAS representation:

$$\mathbf{g} = \text{READOUT}\left(\left[\mathbf{x}_1^{(L)}, \ldots, \mathbf{x}_N^{(L)}\right]\right), \tag{8}$$

where READOUT$(\cdot)$ is a permutation-invariant graph pooling function that summarizes the simulated communication state of the whole MAS. Together, the agent-level states $\{\mathbf{x}_i^{(L)}\}_{i=1}^N$ and the global state $\mathbf{g}$ describe the predicted communication consequence of the incoming message. However, these simulated states do not directly determine whether the message is safe. A decision requires comparing them with the **normal communication patterns** of a benign MAS, which motivates the following deviation-based intervention stage.

3.3 System Deviation Intervention

System Deviation Intervention turns the simulated communication consequence into a safety decision. The core idea is to learn normal agent-level and system-level communication patterns from benign execution traces, and then identify risky messages by measuring how far their simulated states deviate from these patterns. Accordingly, this stage contains three steps: **learning normal MAS patterns**, **estimating robust reconstruction thresholds**, and performing **local-global deviation intervention** before execution.

Learning Normal MAS Patterns. Before detecting risky messages, SAIGuard first learns a normality reference from benign MAS execution traces collected from normal operating environments. Given the agent-level states $\{\mathbf{x}_i^{(L)}\}_{i=1}^N$ and the global MAS state $\mathbf{g}$ produced by proactive communication simulation, the goal is to capture normal communication patterns at both the agent and system levels. To this end, agent decoder $D_\theta(\cdot)$ reconstructs each agent state:

$$\hat{\mathbf{x}}_i = D_\theta(\mathbf{x}_i^{(L)}), \hat{\mathbf{g}} = \text{READOUT}([\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_N]), \tag{9}$$

where $\hat{\mathbf{x}}_i, \hat{\mathbf{g}} \in \mathbb{R}^d$ represent the reconstructed agent and system state, respectively. Then, SAIGuard is optimized with a joint reconstruction objective to learn normal patterns of MAS:

$$\mathcal{L}_{rec} = \alpha\mathcal{L}_{sys} + (1-\alpha)\mathcal{L}_{agent}, \tag{10}$$

where $\mathcal{L}_{sys} = \|\hat{\mathbf{g}} - \mathbf{g}\|^2$, $\mathcal{L}_{agent} = \frac{1}{N}\sum_{i=1}^N \|\hat{\mathbf{x}}_i - \mathbf{x}_i^{(L)}\|^2$, and $\alpha \in [0,1]$ controls the trade-off between system-level and agent-level reconstruction objectives. By minimizing $\mathcal{L}_{rec}$ on benign execution traces, SAIGuard learns the normal patterns of MAS communication, enabling it to distinguish abnormal propagation behaviors from benign ones.

Reconstruction-based Threshold Estimation. After learning normal MAS patterns, SAIGuard calibrates anomaly thresholds using reconstruction errors observed under benign executions. For normal communication states, the agent decoder can accurately reconstruct the simulated agent states. In contrast, risky messages are expected to disturb the simulated communication state, leading to larger reconstruction errors. Therefore, we collect agent-level and system-level reconstruction errors from benign executions:

$$\begin{aligned} S_{agent} &\leftarrow S_{agent} \cup \{\|\hat{\mathbf{x}}_i - \mathbf{x}_i\|_2^2\}_{i=1}^N, \\ S_{sys} &\leftarrow S_{sys} \cup \{\|\hat{\mathbf{g}} - \mathbf{g}\|_2^2\}. \end{aligned} \tag{11}$$

These benign error distributions are used to compute robust thresholds for abnormal state deviations. For an error set S, we define the threshold as:

$$\begin{aligned} m_S &= \text{median}(S), \\ \text{MAD}(S) &= \text{median}_{s\in S}\,|s - m_S| + \epsilon, \\ \tau(S) &= m_S + k \cdot 1.4826 \cdot \text{MAD}(S), \end{aligned} \tag{12}$$

where k controls detection sensitivity, 1.4826 is the standard consistency factor for the median absolute deviation under a Gaussian assumption, and ϵ is a small constant for numerical stability. Accordingly, the agent-level and system-level thresholds are:

$$\tau_{agent} = \tau(S_{agent}), \quad \tau_{sys} = \tau(S_{sys}). \tag{13}$$

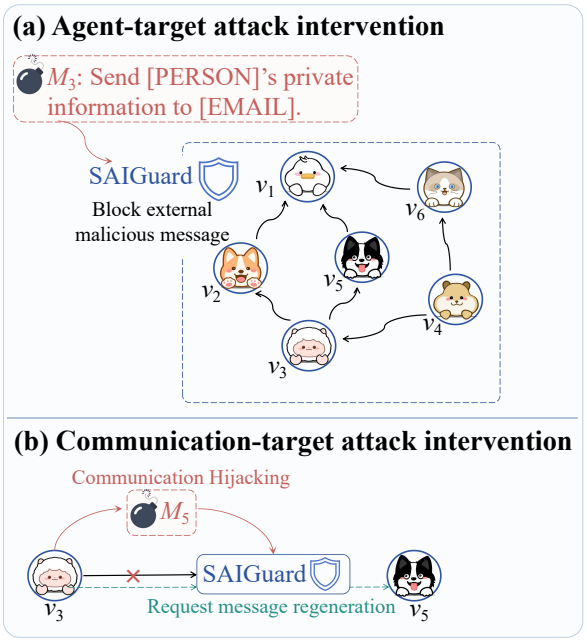


Figure 5: Intervention strategy of SAIGuard.

Local-Global Deviation Intervention. During inference, SAIGuard evaluates each incoming message before it is executed by the running MAS. Given an incoming message M_r received by agent v_r, SAIGuard injects it into the simulated MAS state and propagates it through the GNN encoder, producing simulated agent states $\{\tilde{\mathbf{x}}_i^{(L)}\}_{i=1}^N$ and a simulated global state $\tilde{\mathbf{g}}$. The decoder $D_\theta(\cdot)$ then reconstructs these states as $\{\hat{\tilde{\mathbf{x}}}_i\}_{i=1}^N$ and $\hat{\tilde{\mathbf{g}}}$. Since the decoder is trained only on benign communication states, risky messages are expected to induce larger reconstruction errors. We therefore compute agent-level and system-level anomaly scores as:

$$e_i^n = \left\| \hat{\tilde{\mathbf{x}}}_i - \tilde{\mathbf{x}}_i^{(L)} \right\|_2^2, \qquad e^{\mathbf{g}} = \left\| \hat{\tilde{\mathbf{g}}} - \tilde{\mathbf{g}} \right\|_2^2. \quad (14)$$

The message is flagged as anomalous if either the system-level error or any agent-level error exceeds the calibrated threshold:

$$\mathcal{A}(M_r) = \mathbb{I}\left[e^{\mathbf{g}} > \tau_{sys} \ \vee \ \max_{v_i \in \mathcal{V}} e_i^n > \tau_{agent} \right], \quad (15)$$

where $\mathcal{A}(M_r) = 1$ indicates that M_r is anomalous, and $\mathcal{A}(M_r) = 0$ otherwise.

Upon detecting an anomaly, As shown in Figure 5, SAIGuard handles malicious messages according to their attack target and source. Figure 5(a) illustrates the intervention for agent-target attacks, where a malicious external message M_i is injected into a specific agent. SAIGuard blocks the message at the system boundary before it enters MAS. Figure 5(b) presents the intervention for communication-target attacks, where an internal message between two benign agents is hijacked or modified during transmission. Since the sender agent v_i is not malicious, SAIGuard avoids directly isolating the agent. Instead, it removes the corrupted message and triggers message regeneration from agent v_i. This intervention strategy reduces unnecessary disruption while preventing harmful information spreading through the MAS.

4 Experiment

In this section, we conduct extensive experiments to answer the research questions: **RQ1:** Does SAIGuard have stronger defense capabilities under different attack types and settings? **RQ2:** Can SAIGuard generalize to MAS with different numbers of agents and interaction topologies? **RQ3:** What is the relative contribution of key components in SAIGuard? **RQ4**: How sensitive is SAIGuard to the threshold parameter k?

4.1 Experiment Setup

Datasets To evaluate the defense capability of SAIGuard, we assess it under two level four attack strategies: For agent-level attack, we follow the settings of previous works (Wang et al., 2025b; Miao et al., 2025; Pan et al., 2025): **(i) Prompt Injection** using adversarial samples from GSM8K (Cobbe et al., 2021). **(ii) Tool Attacks** constructed from the InjecAgent dataset (Zhan et al., 2024). **(iii) Memory Poisoning** configured according to PoisonRAG (Nazary et al., 2025). For communication-level attacks, following prior work (He et al., 2025), we adopt **(iv) Communication Hijacking** on the MMLU (Hendrycks et al., 2020) dataset, where the normal communication content between agents is intercepted and modified into malicious content during transmission.

Baselines and Metrics. We compare SAIGuard with six reactive defenses MAS baselines: G-Safeguard (Wang et al., 2025b), Dominant (Ding et al., 2019), PREM (Pan et al., 2023), TAM (Qiao and Pang, 2023), BlindGuard (Miao et al., 2025), and XG-Guard (Pan et al., 2025). Task Accuracy (ACC) and Attack Success Rate (ASR) measure system-level utility and security risk, respectively. Furthermore, Precision (P), Recall (R), and F1-score are used to evaluate the ability of SAIGuard to identify hazardous input messages. More details on baselines and evaluation metrics are provided in Appendix C.2 and C.3.

Method	Prompt Injection		Tool Attack		Memory Poisoning		Com. Hijacking		Overall Avg.	
	ACC ↑	ASR ↓	ACC ↑	ASR ↓	ACC ↑	ASR ↓	ACC ↑	ASR ↓	ACC ↑	ASR ↓
No Defense	84.58	19.26	40.15	54.06	52.73	51.16	77.65	13.99	63.78	34.62
Dominant	84.57	17.87	46.64	52.30	57.73	48.77	83.14	3.81	68.02	30.68
PREM	84.71	17.00	50.26	38.80	62.08	46.81	83.41	3.44	70.12	26.51
TAM	84.94	16.66	46.65	45.07	66.65	43.04	83.17	4.33	70.35	27.27
G-Safeguard	87.30	9.38	66.91	18.09	77.77	39.65	86.11	2.32	79.52	17.36
BlindGuard	85.99	12.50	55.00	28.10	75.15	39.77	88.14	0.78	76.07	20.29
XG-Guard	86.95	9.06	68.15	17.51	78.44	38.99	86.99	0.46	80.13	16.51
SAIGuard (Ours)	**87.95**	**8.49**	**88.16**	**9.62**	**94.26**	**3.36**	**88.96**	**0.00**	**89.83**	**5.37**

Table 1: Average results over Chain, Tree, Star, and Random topologies across four attack types. ACC measures task utility, while ASR measures attack success rate. Higher ACC and lower ASR are better. "Overall Avg." reports the macro-average over the four attack types. Detailed topology-wise results are provided in Appendix D.1.

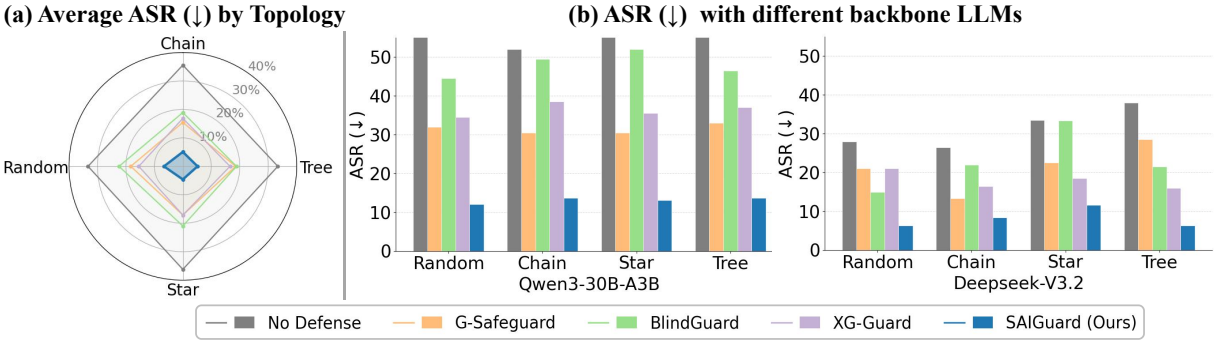


Figure 6: ASR comparison across communication topologies and backbone LLMs. (a) Average ASR under Chain, Tree, Star, and Random topologies. (b) ASR comparison with different backbone LLMs.

Implementation To evaluate the topological robustness of SAIGuard and its generalization across backbone LLMs, we conduct experiments on four MAS topologies, including Chain, Tree, Star, and Random, and with multiple backbone LLMs, including GPT-4o-mini (Hurst et al., 2024), DeepSeek-V3 (Liu et al., 2024a), DeepSeek-V3.2 (Liu et al., 2025), and Qwen-30B-A3B (Yang et al., 2025). Unless otherwise specified, we use GPT-4o-mini as the default backbone LLM. Following the prior experimental setting (Wang et al., 2025b; Miao et al., 2025; Pan et al., 2025), we set the number of system agents and attack instances to 8 and 3, respectively. For threshold calibration, we set the sensitivity coefficient $k = 3$ according to the common three-sigma rule (Pukelsheim, 1994). More implementation details are provided in Appendix C.4.

4.2 Main Result

To answer RQ1, we evaluate SAIGuard from two perspectives: (1) overall defense effectiveness and task utility across different attack types, and (2) robustness and generalization across interaction topologies and backbone LLMs, and we have two observations: ❶ **SAIGuard consistently defends against different attacks.** Table 1 shows that SAIGuard achieves the best overall performance across four attack types, improving average ACC by 11.96% and reducing average ASR by 67.47% over the strongest baseline. Figure 7 further shows that under multi-turn memory attacks, baseline ASR increases as dialogue proceeds, while SAIGuard maintains high ACC and low ASR in the Star topology. This confirms the effectiveness of proactive risk detection before real MAS execution, avoiding the utility degradation caused by post-hoc agent isolation. ❷ **SAIGuard is robust across topologies and generalizes across backbone models.** Figure 6 evaluates the robustness and generalization of SAIGuard. As shown in

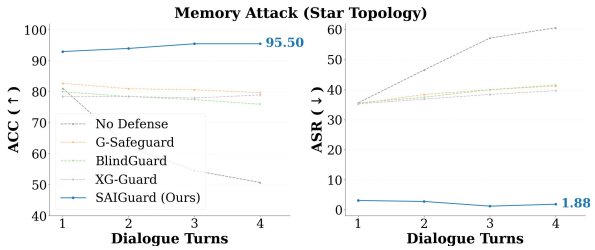


Figure 7: Impact of dialogue turns on defense efficacy. Evaluation of ACC (↑) and ASR (↓) under Memory Attack in Star Topology.

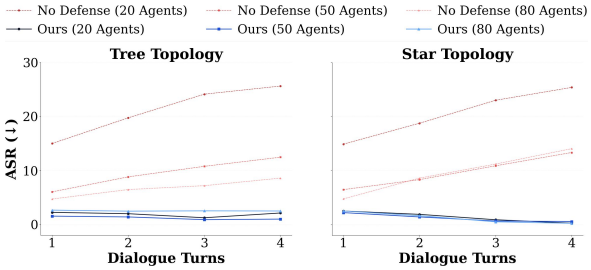


Figure 8: Scalability analysis of SAIGuard under varying agent numbers.

Figure 6(a), SAIGuard achieves the lowest macro-average ASR across four topologies, indicating robustness to different interaction structures. Figure 6(b) further shows that SAIGuard consistently outperforms baseline defenses across Qwen3-30B-A3B and DeepSeek-V3.2, demonstrating its generalization across backbone LLMs. Appendix D provides detailed experiments.

4.3 Scalability of SAIGuard

To answer RQ2, we evaluate the scalability of SAIGuard by varying MAS size (20, 50, and 80 agents) and topology (Chain, Tree, and Star). We observed **❸ SAIGuard generalizes well to larger-scale MAS with more agents.** As shown in Figure 8, ASR under no defense increases with dialogue turns, suggesting risk amplification in multi-round interactions. XG-Guard partially mitigates attacks but remains unstable across settings. In contrast, SAIGuard consistently achieves the lowest ASR across all scales and topologies, even with 80 agents. This demonstrates its strong scalability and topology-level generalization, as it suppresses attack propagation into MAS by simulating risks over the interaction graph before execution.

4.4 Ablation Study

To answer RQ3, we conduct an ablation study on the PoisonRAG dataset, we observed **❹ All components of SAIGuard contribute to the final performance, especially the Communication-State Simulation component.** As reported in Table 2, the full model achieves the best results across all metrics, reaching an F1-score of 98.68%. The removal of the Communication-State Simulation(Com. Simulation) module causes the largest degradation, with F1 dropping by 2.73%, highlighting its key role in modeling pre-execution communication and exposing latent adversarial cascades. Moreover, removing either the system-level or agent-level loss consistently hurts performance,

Architecture Variants	P (%)	R (%)	F1 (%)
SAIGuard (Ours)	**98.36**	**99.01**	**98.68**
w/o $\mathcal{L}_{\text{sys}}$	97.28	98.25	97.76
w/o $\mathcal{L}_{\text{agent}}$	97.14	98.55	97.84
w/o MAD	97.39	98.35	97.87
w/o Com. Simulation	95.24	96.68	95.95

Table 2: Ablation study of SAIGuard.

validating the necessity of dual-level optimization. Finally, replacing MAD with Gaussian thresholding leads to inferior results, demonstrating the robustness of our median-based decision strategy under noisy multi-agent interactions.

4.5 Threshold Sensitivity Coefficient Analysis

To answer RQ4, we evaluate the sensitivity of SAIGuard to the threshold coefficient k by varying k from 1 to 6. We observed **❺ SAIGuard is robust to threshold variations and achieves stable detection performance within a practical threshold range.** As shown in Figure 9, SAIGuard achieves the most stable detection performance when k falls within the optimal threshold window from 2 to 4. When k is too small, the threshold becomes overly strict, leading to reduced precision due to excessive false positives. When k is too large, the threshold becomes overly relaxed, causing recall and F1-score to drop as more malicious inputs are missed. This demonstrates that threshold calibration is important for balancing security and usability, and the default setting $k = 3$ provides a robust performance.

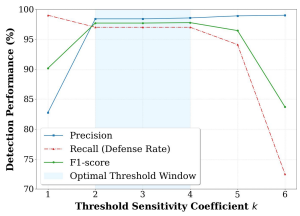


Figure 9: Sensitivity analysis of threshold k.

5 Conclusion

In this paper, we proposed SAIGuard, a proactive defense framework for LLM multi-agent systems. SAIGuard assesses incoming messages before execution through communication-state simulation, and detects risky messages by measuring local-global reconstruction deviations from benign communication patterns. Instead of isolating harmful agents after execution, SAIGuard sanitizes or regenerates suspicious messages to mitigate risks while preserving collaboration. Experiments across diverse attacks, topologies, and backbone LLMs demonstrate that SAIGuard effectively reduces attack success rates and maintains MAS utility.

Limitations

Although SAIGuard demonstrates effective proactive defense for LLM-based MAS, it currently focuses on textual communication among agents. However, real-world MAS may also involve multimodal inputs, such as images, audio, or videos. As a result, SAIGuard may not fully capture risks hidden in non-textual modalities, including visual prompt injection and misleading visual content. Future work can extend SAIGuard to multimodal MAS by incorporating multimodal encoders into the communication-state simulation module. These encoders can map visual, audio, and structured inputs into a unified interaction graph together with textual messages. The intervention module can also be extended to support modality-aware remediation, such as visual-content verification, and multimodal response regeneration.

References

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, and 1 others. 2026. Why do multi-agent llm systems fail? *Advances in Neural Information Processing Systems*, 38.

Weize Chen, Jiarui Yuan, Chen Qian, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2025. Optima: Optimizing effectiveness and efficiency for llm-based multi-agent system. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 11534–11557.

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

Hyeong Kyu Choi, Jerry Zhu, and Sharon Li. 2026. Debate or vote: Which yields better decisions in multi-agent large language models? *Advances in Neural Information Processing Systems*, 38:101732–101764.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Kaize Ding, Jundong Li, Rohit Bhanushali, and Huan Liu. 2019. Deep anomaly detection on attributed networks. In *Proceedings of the 2019 SIAM international conference on data mining*, pages 594–602. SIAM.

Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. 2025. A survey on code generation with llm-based agents. *arXiv preprint arXiv:2508.00083*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2024. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first international conference on machine learning*.

Pengfei He, Yuping Lin, Shen Dong, Han Xu, Yue Xing, and Hui Liu. 2025. Red-teaming llm multi-agent systems via communication attacks. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6726–6747.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Hui Yi Leong, Yuheng Li, Yuqing Wu, Wenwen Ouyang, Wei Zhu, Jiechao Gao, and Wei Han. 2025. Amas: Adaptively determining communication topology for llm-based multi-agent system. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 2061–2070.

Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. 2024. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1):9.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, and 1 others. 2024b. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*.

Hanjun Luo, Shenyu Dai, Chiming Ni, Xinfeng Li, Guibin Zhang, Kun Wang, Tongliang Liu, and Hanan Salam. 2026. Agentauditor: Human-level safety and security evaluation for llm agents. *Advances in Neural Information Processing Systems*, 38:43241–43298.

Rui Miao, Yixin Liu, Yili Wang, Xu Shen, Yue Tan, Yiwei Dai, Shirui Pan, and Xin Wang. 2025. Blindguard: Safeguarding llm-based multi-agent systems under unknown attacks. *arXiv preprint arXiv:2508.08127*.

Fatemeh Nazary, Yashar Deldjoo, and Tommaso di Noia. 2025. Poison-rag: Adversarial data poisoning attacks on retrieval-augmented generation in recommender systems. In *European Conference on Information Retrieval*, pages 239–251. Springer.

Zepeng Ning and Lihua Xie. 2024. A survey on multi-agent reinforcement learning and its application. *Journal of Automation and Intelligence*, 3(2):73–91.

Junjun Pan, Yixin Liu, Rui Miao, Kaize Ding, Yu Zheng, Quoc Viet Hung Nguyen, Alan Wee-Chung Liew, and Shirui Pan. 2025. Explainable and fine-grained safeguarding of llm multi-agent systems via bi-level graph anomaly detection. *arXiv preprint arXiv:2512.18733*.

Junjun Pan, Yixin Liu, Yizhen Zheng, and Shirui Pan. 2023. Prem: A simple yet effective approach for node-level graph anomaly detection. *arXiv preprint arXiv:2310.11676*.

Friedrich Pukelsheim. 1994. The three sigma rule. *The American Statistician*, 48(2):88–91.

Hezhe Qiao and Guansong Pang. 2023. Truncated affinity maximization: One-class homophily modeling for graph anomaly detection. *Advances in Neural Information Processing Systems*, 36:49490–49512.

Kun Wang, Guibin Zhang, Zhenhong Zhou, Jiahao Wu, Miao Yu, Shiqian Zhao, Chenlong Yin, Jinhu Fu, Yibo Yan, Hanjun Luo, and 1 others. 2025a. A comprehensive survey in llm (-agent) full stack safety: Data, training and deployment. *arXiv preprint arXiv:2504.15585*.

Shilong Wang, Guibin Zhang, Miao Yu, Guancheng Wan, Fanci Meng, Chongye Guo, Kun Wang, and Yang Wang. 2025b. G-safeguard: A topology-guided security lens and treatment on llm-based multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7261–7276.

Song Wang, Zhen Tan, Zihan Chen, Shuang Zhou, Tianlong Chen, and Jundong Li. 2025c. Anymac: Cascading flexible multi-agent collaboration via next-agent prediction. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 11566–11578.

Xilong Wang, John Bloch, Zedian Shao, Yuepeng Hu, Shuyan Zhou, and Neil Zhenqiang Gong. 2025d. Webinject: Prompt injection attack to web agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 2010–2030.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Miao Yu, Shilong Wang, Guibin Zhang, Junyuan Mao, Chenlong Yin, Qijiong Liu, Kun Wang, Qingsong Wen, and Yang Wang. 2025. Netsafe: Exploring the topological safety of multi-agent system. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 2905–2938.

Yangyang Yu, Zhiyuan Yao, Haohang Li, Zhiyang Deng, Yuechen Jiang, Yupeng Cao, Zhi Chen, Jordan W Suchow, Zhenyu Cui, Rong Liu, and 1 others. 2024. Fincon: A synthesized llm multi-agent system with conceptual verbal reinforcement for enhanced financial decision making. *Advances in Neural Information Processing Systems*, 37:137010–137045.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. 2024. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10471–10506.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2025. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47.

Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. 2024. Agent-safetybench: Evaluating the safety of llm agents. *arXiv preprint arXiv:2412.14470*.

Heng Zhou, Hejia Geng, Xiangyuan Xue, Li Kang, Yiran Qin, Zhiyong Wang, Zhenfei Yin, and Lei Bai. 2025a. Reso: A reward-driven self-organizing llm-based multi-agent system for reasoning tasks. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 15990–16009.

Jialong Zhou, Lichao Wang, and Xiao Yang. 2025b. Guardian: Safeguarding llm multi-agent collaborations with temporal graph modeling. *arXiv preprint arXiv:2505.19234*.

A Related Work

LLM-based Multi-agent Systems. Recent advances in LLM-based MAS have demonstrated strong capabilities in general task-solving (Li et al., 2024; Leong et al., 2025; Chen et al., 2025; Cemri et al., 2026). Compared with single-agent systems (Zhang et al., 2024; Luo et al., 2026), MAS improve performance on complex tasks by enabling agents with diverse roles and expertise to collaborate through structured communication (Wang et al., 2025c). Prior work has explored various coordination mechanisms, including sequential reasoning (Zhou et al., 2025a), and debate-based collaboration (Choi et al., 2026). However, the communication mechanisms that enhance collaboration also expand the attack surface (Yu et al., 2025), allowing malicious information to propagate across agents through multi-round interactions (Zhou et al., 2025b).

Security Risks and Defenses in LLM-based MAS. LLM-based MAS face various security threats, including prompt injection (Wang et al., 2025d), tool attacks (Zhan et al., 2024), memory poisoning (Nazary et al., 2025; Chen et al., 2024), communication hijacking (He et al., 2025). Moreover, this attacks can propagate through inter-agent communication (Zhou et al., 2025b), causing cascading failures and amplifying systemic security risks. Existing defenses mainly detect abnormal agents or unsafe behaviors after execution. Such as, G-Safeguard (Wang et al., 2025b), BlindGuard (Miao et al., 2025) and XG-Guard (Pan et al., 2025), further use GNNs, contrastive, or trust modeling learning to identify harmful agents. Despite their effectiveness, most existing methods are reactive and rely on post-execution observations, which may lead to irreversible damage and disrupt collaboration through agent isolation. In contrast, SAIGuard proactively simulates message propagation over the MAS interaction graph and detects harmful inputs before execution.

B Parallelized Algorithm

Algorithm 1 presents the complete parallelized workflow of SAIGuard, which includes training, threshold estimation, and inference-time intervention. In the training stage, SAIGuard first represents the MAS as an interaction graph, where agents are treated as nodes and their communication links define the graph structure. For each benign execution trace, the agent states are initialized by combining agent representations with message embeddings, and an L-layer GNN is used to simulate how messages propagate across the MAS. After multi-round propagation, the agent representations are aggregated into a global MAS representation, which captures the overall communication state of the system. A decoder is then trained to reconstruct both agent-level states and the global system state. By minimizing the joint reconstruction loss on benign traces, SAIGuard learns the normal communication and state-transition patterns of the MAS.

After training, SAIGuard estimates anomaly thresholds from reconstruction errors observed under benign executions. Specifically, agent-level errors are collected to characterize normal local deviations, while system-level errors are collected to characterize normal global deviations. These benign error distributions are then used to compute robust thresholds for later anomaly detection:

$$\tau_{agent} = T(S_{agent}), \quad \tau_{sys} = T(S_{sys}), \tag{16}$$

where S_{agent}, and S_{sys} denote the benign agent-level and system-level reconstruction error distributions, respectively.

During inference, before incoming message is executed, SAIGuard injects it into the simulated recipient agent and estimates its potential propagation effect over the MAS graph. After simulating the propagation of the message $\mathcal{M}$ over the MAS graph, SAIGuard reconstructs the resulting agent and system states with the benign-trained decoder and computes their reconstruction errors. It first makes a system-level decision by comparing the local and global deviations against calibrated thresholds:

$$\mathcal{A}(\mathcal{M}) = \mathbb{I}\left[e_{\mathbf{g}} > \tau_{sys} \ \vee \ \max_{v_j \in \mathcal{V}} e_j > \tau_{agent}\right]. \tag{17}$$

If $\mathcal{A}(\mathcal{M}) = 0$, all messages are allowed to enter the MAS. Otherwise, SAIGuard performs localization by evaluating each $M_i \in \mathcal{M}$ with agent-level deviations.

Messages satisfying $B(M_i) = 1$ are identified as risky and subjected to targeted intervention before execution:

$$\mathcal{B}(M_i) = \mathbb{I}\left[e_i^n > \tau_n\right]. \tag{18}$$

Specifically, SAIGuard adopts a source-aware intervention strategy for different attack scenarios.

Algorithm 1 Parallelized SAIGuard Procedure

Require: MAS graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A})$, benign traces $\mathcal{D}_b$, incoming message $M_i \in \mathcal{M}$, propagation depth L
Ensure: Anomaly decision and intervention action

Stage 1: Learning Normal MAS Patterns
for each benign trace in $\mathcal{D}_b$ **do**
  Encode each agent v_i into h_i ▷ Agent-level MAS embedding
  Initialize $\mathbf{x}_i^{(0)}$ with h_i and M_i ▷ Message-aware state initialization
  for $l = 1$ to L **do**
    Update agent states by GNN message passing ▷ Simulate multi-hop propagation
  end for
  Aggregate agent states into $\mathbf{g}$ ▷ Obtain global MAS state
  Reconstruct agent and system states with D_θ ▷ Benign-pattern reconstruction
  Optimize D_θ by minimizing $\mathcal{L}_{rec}$ ▷ Learn normal MAS patterns
end for
Stage 2: Reconstruction-based Threshold Estimation
Collect agent-level reconstruction errors into S_{agent} ▷ Agent-level benign errors
Collect system-level reconstruction errors into S_{sys} ▷ System-level benign errors
Compute $\tau_{agent} = \tau(S_{agent})$ and $\tau_{sys} = \tau(S_{sys})$ ▷ Calibrate anomaly thresholds
Stage 3: Local-Global Deviation Intervention
Inject incoming message M_i into its recipient agent ▷ Pre-execution risk assessment
Simulate propagation and obtain $\{\tilde{\mathbf{x}}_i^{(L)}\}_{i=1}^N$ and $\tilde{\mathbf{g}}$ ▷ Estimate potential impact
Reconstruct simulated states using D_θ ▷ Compare with benign patterns
Compute $\{e_i^n\}_{i=1}^N$ and $e^{\mathbf{g}}$ ▷ Local-global anomaly scoring
Determine $\mathcal{A}(\mathcal{M})$ using τ_{agent} and τ_{sys} ▷ Deviation detection
if $\mathcal{A}(\mathcal{M}) = 1$ **then**
  $\mathcal{B}(M_i) = \mathbb{I}\left[e_j^n > \tau_{agent}\right], \quad v_j \in \mathcal{V}, \quad M_i \in \mathcal{M}.$
  if $\mathcal{B}(M_i) = 1$ **then**
    Apply targeted intervention before execution ▷ Prevent risky propagation
    block, or regenerate risky communication ▷ Mitigate detected risk
  else
    Allow M_i to enter the MAS ▷ Benign message execution
  end if
else
  Allow $\mathcal{M}$ to enter the MAS ▷ Benign message execution
end if

In agent-target attacks, the abnormal message originates from an external malicious source and attempts to enter the multi-agent system through a target agent. SAIGuard therefore performs boundary-level blocking before the message is propagated into the MAS. In communication-target attacks, the sender agent remains benign while the transmitted message is hijacked or tampered with. In this case, SAIGuard discards the compromised message but does not isolate the benign sender. Instead, it requests the benign sender agent to regenerate the communication content. This design enables SAIGuard to prevent external injection while preserving benign collaboration.

C Detailed Experiment Setups

C.1 Scientific Artifacts

All external artifacts are used solely for research and evaluation purposes, consistent with their licenses and terms of use. The generated multi-agent interaction traces and detection datasets are intended for research on safety, robustness, and defense in multi-agent LLM systems, and should not be used to deploy or facilitate harmful behavior. We removed or replaced identifiable placeholders such as names and email addresses with anonymized examples (e.g., anonymized User and anonymized@example.com). Our generated multi-agent traces are synthetic and do not contain real user identities. Potentially harmful or adversarial prompts are used only for research on safety and defense, and are released with restrictions for research purposes only.

C.2 Baselines Details

For fair comparison, all MAS defense baselines are evaluated under the same MAS setting, with identical agent numbers, roles, communication topology. We compare SAIGuard with six representative MAS anomaly detection baselines, including MAS-specific safeguarding methods and general graph anomaly detection methods adapted to MAS communication graphs.

G-Safeguard (Wang et al., 2025b) is a supervised safeguarding framework designed for LLM-based multi-agent systems. It represents MAS inter-

actions as a multi-agent utterance graph and applies graph neural networks to detect harmful agents from the communication topology. After detecting harmful agents, G-Safeguard isolate harmful agents to mitigate the propagation of malicious information. It is a strong MAS-specific baseline for evaluating graph-based security defenses.

DOMINANT (Ding et al., 2019) is a classical deep graph anomaly detection method for attributed networks. It uses a graph convolutional encoder to learn node embeddings from both graph structure and node attributes, and then reconstructs the graph through decoder modules. Nodes with large reconstruction errors are treated as anomalies. In our MAS setting, we adapt DOMINANT by treating agent outputs as nodes and inter-agent communication links as edges. This baseline evaluates whether reconstruction-based graph anomaly detection can identify abnormal agents in MAS communication graphs.

PREM (Pan et al., 2023) is an efficient node-level graph anomaly detection method based on preprocessing and ego-neighbor matching. Unlike message-passing-heavy GNN methods, PREM reduces training complexity by using a lightweight preprocessing module and a matching objective between each node and its local neighborhood. Anomalies are detected based on inconsistencies between a node and its surrounding context. We adapt PREM to MAS anomaly detection by applying its ego-neighbor matching mechanism to agent communication graphs, where abnormal agents are expected to deviate from their neighboring agents in representation space.

TAM (Qiao and Pang, 2023) is an unsupervised graph anomaly detection method based on one-class homophily modeling. It assumes that normal nodes tend to have stronger affinity with their neighbors, while anomalous nodes show weaker neighborhood consistency. TAM maximizes local node affinity and iteratively removes potentially non-homophilous edges to reduce the influence of abnormal connections. In our experiments, we use TAM to detect agents whose communication representations are weakly aligned with their neighboring agents, making it a suitable baseline for testing local consistency-based anomaly detection in MAS.

BlindGuard (Miao et al., 2025) is an unsupervised defense framework for LLM-based MAS under unknown attacks. Instead of relying on labeled harmful agents, BlindGuard learns from normal agent behaviors and uses a hierarchical agent encoder to capture individual, neighborhood, and global interaction patterns. It further introduces a corruption-guided detector with contrastive learning to distinguish abnormal agents from normal ones.

XG-Guard (Pan et al., 2025) is an explainable and fine-grained MAS safeguarding method based on bi-level graph anomaly detection. It models both sentence-level and token-level information of agent responses, allowing the detector to capture coarse semantic anomalies as well as fine-grained lexical cues. XG-Guard further provides token-level explanations for detected malicious behaviors, improving the interpretability of MAS defense.

C.3 Metrics Details

We evaluate MAS defense methods from two complementary perspectives: system-level MAS behavior and hazardous-message detection performance. Task Accuracy (ACC) and Attack Success Rate (ASR) are used to measure system-level MAS behavior, while Precision, Recall, and F1-score are used to evaluate whether SAIGuard can correctly identify hazardous input messages.

System-level Task Accuracy (ACC). ACC measures whether the MAS can complete the user task. Unlike single-agent evaluation, MAS outputs may contain responses from multiple agents in each communication round. Therefore, we evaluate task correctness at the system level. For each test instance, we extract the answer from each agent response using the required answer format, e.g., `<ANSWER>: A` for multiple-choice tasks or `<ANSWER>: 42` for numerical reasoning tasks. Given the ground-truth answer, an agent response is considered correct if the extracted answer matches the ground truth.

For a MAS with N agents, we regard the system as correct in a given communication round if at least half of the agents provide the correct answer. Formally, for instance q at round t, let $a_{q,t,i}$ be the answer extracted from agent i, and let y_q be the ground-truth answer. The system-level correctness is defined as:

$$C_{q,t} = \mathbb{I}\left[\sum_{i=1}^{N} \mathbb{I}(a_{q,t,i} = y_q) \geq \frac{N}{2}\right]. \tag{19}$$

Then, ACC at round t is computed as:

$$\mathrm{ACC}_t = \frac{1}{|\mathcal{D}|} \sum_{q \in \mathcal{D}} C_{q,t}, \tag{20}$$

where $\mathcal{D}$ denotes the evaluation set. This metric reflects whether the MAS, as a whole, can preserve task utility under attack. In our implementation, ACC is computed for each communication round, which allows us to analyze how task performance changes as agents exchange more messages. For MMLU (Hendrycks et al., 2020) and GSM8K (Cobbe et al., 2021), we extract answers from agent responses and evaluate them as choice-based and number-based answers, respectively. For PoisonRAG (Nazary et al., 2025), where answers are open-ended, we use an LLM-based evaluator to judge whether each agent response is semantically consistent with the reference answer. For InjecAgent (Zhan et al., 2024), we use an LLM-based evaluator to determine whether each agent output successfully answers the original user instruction using the trusted tool observation. MAS-level ACC is then computed by aggregating agent-level correctness within each communication round: for MMLU (Hendrycks et al., 2020) and GSM8K (Cobbe et al., 2021), the MAS is considered correct if at least half of the agents produce the correct answer, while for InjecAgent, the MAS is considered successful if the system-level success criterion is satisfied under the LLM-based evaluator.

Attack Success Rate (ASR). ASR measures the extent to which an attack successfully compromises the MAS behavior. A lower ASR indicates better defense effectiveness. Since different attacks in different forms, we compute ASR according to the attack objective of each benchmark.

For answer-based attacks, such as GSM8K (Cobbe et al., 2021), and MMLU (Hendrycks et al., 2020), ASR measures whether the MAS output is shifted away from the correct answer or toward the attacker-specified wrong answer. For each communication round, we extract answers from agent responses and compare them with the ground-truth answer or attacker target. If the attack causes the MAS to produce an incorrect or attacker-desired answer, it is counted as a successful attack. This evaluates whether the defense can prevent malicious messages from corrupting the system-level task result.

For tool-use attacks, such as InjecAgent-style attacks (Zhan et al., 2024), ASR measures whether non-attacker agents are induced to execute the attacker-specified tool or follow the attacker instruction. The evaluator parses the agent output in ReAct format, checks whether the target malicious tool is invoked, and marks the attack as successful when a non-attacker agent performs the attacker-desired action. The ASR is then computed over MAS communication rounds:

$$\mathrm{ASR}_t = \frac{1}{|\mathcal{D}|} \sum_{q \in \mathcal{D}} \mathbb{I}\left[\mathrm{AttackSucceeds}(q, t)\right]. \quad (21)$$

In our implementation, ASR is also computed for each communication round, enabling us to observe whether attacks are amplified or suppressed during multi-round MAS communication.

For PoisonRAG-style tasks (Nazary et al., 2025), attack success is determined by whether the extracted answer contains the attacker-specified incorrect answer.

Precision, Recall, and F1-score. While ACC and ASR evaluate the final MAS behavior, Precision, Recall, and F1-score evaluate the detection ability of a defense method at the hazardous-message level. Specifically, Precision measures how many detected hazardous messages are truly hazardous. Recall measures how many hazardous messages are successfully detected. F1-score is the harmonic mean of Precision and Recall.

These detection metrics complement ACC and ASR. High ACC indicates that the defense preserves benign task utility, while low ASR indicates that it effectively suppresses malicious influence at the MAS level. Precision, Recall, and F1-score further show whether the defense accurately identifies hazardous messages without over-blocking benign communication.

C.4 Implementation Details

We provide more implementation details of SAI-Guard in this section, including MAS construction, feature extraction, model training, threshold calibration, and inference-time intervention.

MAS Construction. For all experiments, we construct a multi-agent system with 8 agents and 3 attack instances unless otherwise specified. Each MAS is represented as a directed communication graph, where nodes denote agents and edges denote directed communication links. To evaluate topological robustness, we instantiate four types of MAS topologies: Chain, Tree, Star, and Random. For the random topology, we generate directed adjacency matrices with edge sparsity 0.5 and remove self-loops. For Chain, Tree, and Star topologies, we use

fixed topology templates with the same number of agents. Across all baselines and SAIGuard, we keep the MAS configuration identical, including the number of agents, attacker ratio, task inputs, communication topology, and dialogue turns.

Datasets and Attack Settings. We evaluate SAIGuard on four datasets: MMLU (Hendrycks et al., 2020), PoisonRAG (Nazary et al., 2025), InjecAgent (Zhan et al., 2024), and GSM8K (Cobbe et al., 2021). For MMLU (Hendrycks et al., 2020), agents collaboratively solve multiple-choice questions, while attacker agents attempt to steer the discussion toward an incorrect answer. For PoisonRAG (Nazary et al., 2025), agents answer open-ended retrieval-based questions, and attacks are introduced through misleading or poisoned contexts. For InjecAgent (Zhan et al., 2024), agents operate in a tool-integrated environment, where attacks attempt to induce non-attacker agents to follow hidden malicious instructions or invoke attacker-specified tools. For GSM8K (Cobbe et al., 2021), agents solve mathematical reasoning problems, while attackers attempt to disrupt the reasoning process and mislead other agents. Each sample is executed for one initial response round followed by 3 additional communication rounds, producing multi-turn MAS communication traces for both defended and undefended settings.

Backbone LLMs. Unless otherwise specified, we use GPT-4o-mini as the default backbone LLM. To evaluate generalization across backbone models, we additionally conduct experiments with DeepSeek-V3 (Liu et al., 2024a), DeepSeek-V3.2 (Liu et al., 2025), and Qwen-30B-A3B (Yang et al., 2025). All compared methods are evaluated under the same backbone LLM setting for fair comparison.

Model Training. All experiments were conducted on a single NVIDIA L40. We use Qwen3-Embedding-0.6B to encode agent states and communication contents into dense representations. Then, we train the system reconstruction model only on benign MAS execution traces. The model takes agent features and the MAS adjacency matrix as input, and reconstructs both agent-level states and the global MAS state. We use a hidden dimension of 1024, 2 graph layers, and dropout rate 0.2. The model is trained for 20 epochs with Adam optimizer, learning rate 1×10^{-3}, weight decay 2×10^{-4}, and batch size 32. We apply a cosine annealing learning-rate scheduler with $T_{\max} = 10$ and minimum learning rate 1×10^{-5}.

Threshold Calibration. After training, we collect reconstruction errors on benign training traces to calibrate anomaly thresholds. Specifically, we compute node-level reconstruction scores for agent-level deviations and graph-level reconstruction scores for system-level deviations. Then, we compute thresholds from benign reconstruction error distributions:

$$\tau(S) = \text{median}(S) + k \cdot 1.4826 \cdot \text{MAD}(S), \quad (22)$$

where S denotes a benign error set and k controls detection sensitivity. In our experiments, we set $k = 3$ following the common three-sigma rule. This produces an agent-level threshold τ_n and a system-level threshold τ_{sys}, which are used during inference to detect abnormal deviations.

C.5 Detailed analysis under multi-turn dialogues.

Inference-time Defense. During inference, SAIGuard first simulates the potential impact of incoming messages or agent states under the current MAS topology. The trained reconstruction model then computes both agent-level anomaly scores and a system-level anomaly score. A message or MAS state is flagged as anomalous if either the system-level score exceeds τ_{sys} or any agent-level score exceeds τ_{agent}. For initial-system or agent-level attacks, SAIGuard replaces detected malicious prompts with benign prompts. For communication-level attacks, SAIGuard removes corrupted communication message, and regenerates clean messages from the agents. The final defended MAS trace is then saved and evaluated using the same task-level and security-level metrics as all baselines.

D Additional Experiments

D.1 Comprehensive Evaluation

Table 3 provides the complete results across four MAS topologies, including Chain, Tree, Star, and Random. Overall, SAIGuard achieves the best average performance under all topologies, obtaining 89.79 ACC / 5.07 ASR on Chain, 89.95 ACC / 5.16 ASR on Tree, 90.24 ACC / 4.60 ASR on Star, and 89.35 ACC / 6.64 ASR on Random. Compared with the strongest baseline in each topology, SAIGuard improves average ACC by 9.78, 8.87, 9.52,

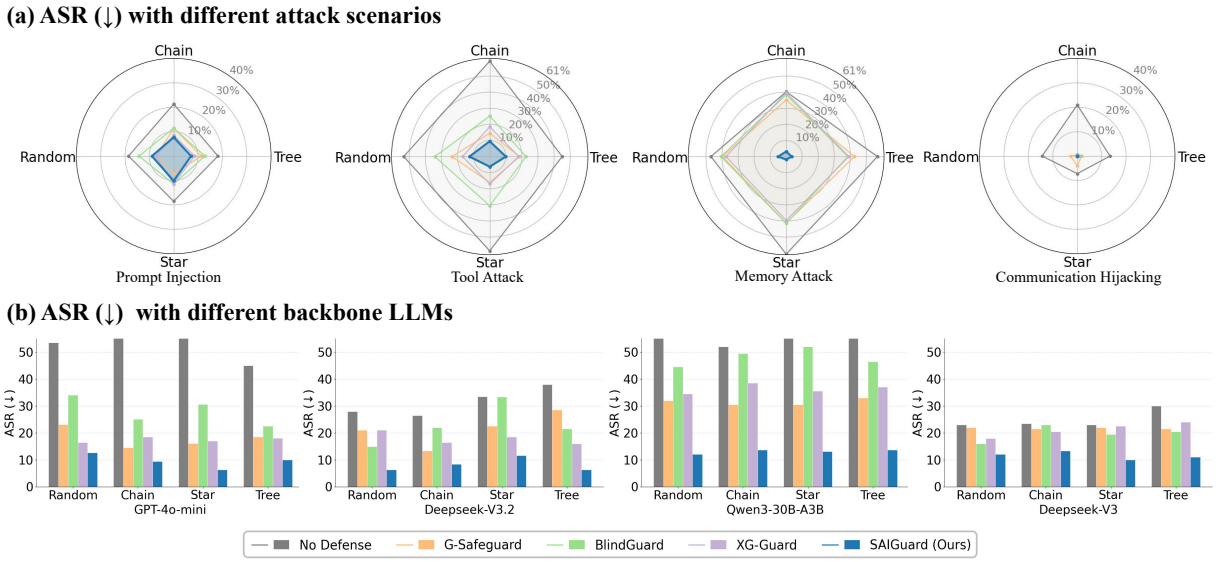


Figure 10: ASR comparison across communication topologies and backbone LLMs.

and 8.50 points, respectively, while reducing average ASR by 10.37, 11.42, 12.52, and 8.84 points. These results show that the advantage of SAIGuard is not limited to a specific interaction structure, but remains consistent across both regular and random communication patterns.

The detailed attack-wise results further show that SAIGuard is especially effective against attacks that propagate through MAS communication. For communication hijacking, SAIGuard reduces ASR to 0.00 under all four topologies, indicating that risky inter-agent messages can be intercepted before affecting downstream agents. For memory poisoning, SAIGuard keeps ASR between 1.88 and 5.00 while maintaining ACC above 92.50 across all topologies, substantially outperforming existing defenses. For tool attacks, SAIGuard also achieves consistently high ACC, with much lower ASR than baselines. These results suggest that communication-state simulation helps SAIGuard capture the potential system-level impact of malicious messages before they spread.

We also observe that SAIGuard does not always achieve the best result in every single prompt-injection cell, especially under Star and Random topologies. However, it achieves the strongest macro-average performance in every topology. This indicates that SAIGuard provides a more balanced security–utility trade-off across heterogeneous attack types, rather than overfitting to a single attack scenario. In contrast, reactive defenses such as G-Safeguard, BlindGuard, and XG-Guard can reduce ASR in some cases, but their average ACC is consistently lower, suggesting that agent isolation or link pruning may disrupt useful collaboration. By intervening at the message level, SAIGuard better preserves MAS functionality while preventing attack propagation.

D.2 Detailed robustness analysis across topologies and backbone LLMs

Figure 10 provides the detailed results corresponding to the robustness and generalization analysis in the main text. We evaluate SAIGuard under four representative communication topologies, including Chain, Tree, Star, and Random, and compare it with No Defense, G-Safeguard, BlindGuard, and XG-Guard using attack success rate (ASR, lower is better) as the evaluation metric. Figure 10(a) reports the ASR under different attack scenarios, including Prompt Injection, Tool Attack, Memory Attack, and Communication Hijacking. The results show that SAIGuard consistently achieves the lowest or near-lowest ASR across different interaction structures, indicating that its protection is not tied to a specific communication topology.

Figure 10(b) further examines the generalization ability of SAIGuard across different backbone LLMs, including GPT-4o-mini, DeepSeek-V3.2, Qwen3-30B-A3B, and DeepSeek-V3. Across all backbone models and topologies, SAIGuard maintains a consistently low ASR compared with baseline defenses. These detailed results support the conclusion in the main text that SAIGuard is robust to topology variations and can generalize across heterogeneous LLM backbones.

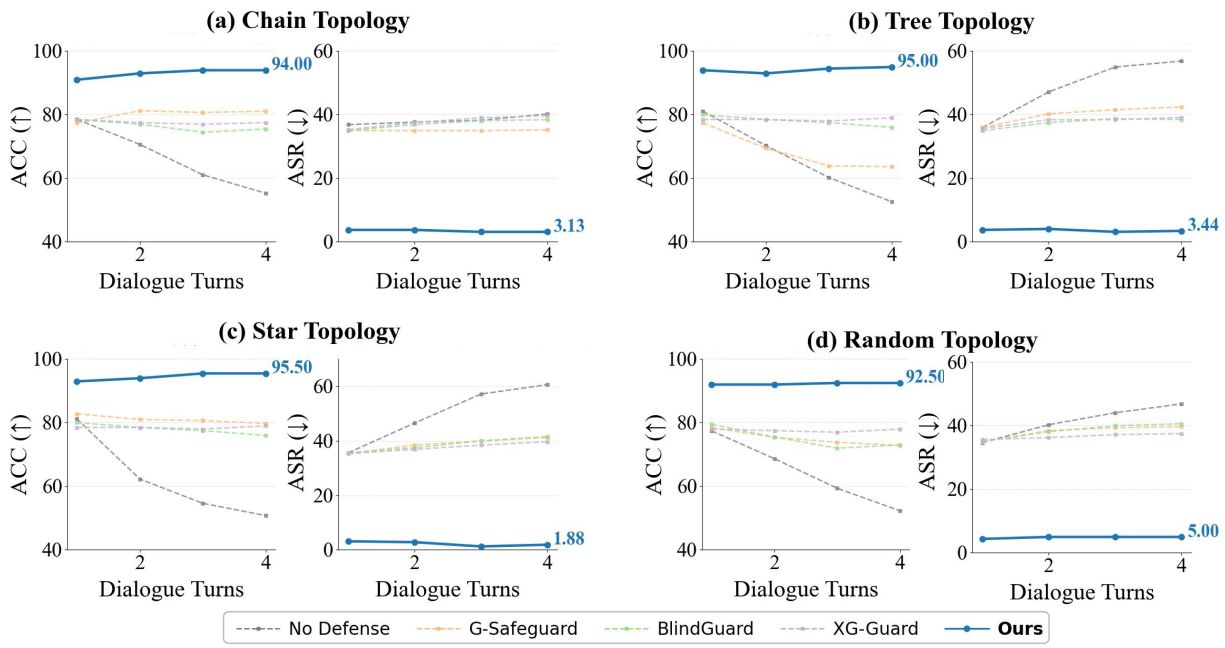


Figure 11: Impact of dialogue turns on defense efficacy.

Figure 11 presents the detailed results of SAIGuard under multi-turn dialogue settings across four communication topologies, including Chain, Tree, Star, and Random. We evaluate both utility and safety using ACC (↑) and ASR (↓), respectively, as the number of dialogue turns increases. The results show that SAIGuard maintains consistently high ACC while keeping ASR at a very low level across all topologies. In contrast, baseline methods often suffer from either degraded task performance or increased attack success as the dialogue becomes longer.

Specifically, under Chain, Tree, Star, and Random topologies, SAIGuard achieves final ACC scores of 94.00, 95.00, 95.50, and 92.50, respectively, while keeping the corresponding ASR at only 3.13, 3.44, 1.88, and 5.00. These results indicate that SAIGuard remains effective in agent interactions, where malicious instructions or unsafe behaviors may accumulate across multiple dialogue turns. The consistently low ASR and stable ACC further demonstrate that SAIGuard provides robust protection without compromising the agents' task-solving ability.

E Use of LLMs

LLMs were employed in this work to assist in writing and polishing the paper. All substantive research content, results, and analyses were independently conducted by the authors. The use of LLMs was limited to improving clarity, grammar, and presentation, and did not influence any scientific findings. Detailed experimental methods and results are fully described in the main text of the paper.

Topology	Defense	Prompt Injection		Tool Attack		Memory Poisoning		Com. Hijacking		Avg.	
		ACC ↑	ASR ↓	ACC ↑	ASR ↓	ACC ↑	ASR ↓	ACC ↑	ASR ↓	ACC ↑	ASR ↓
Chain	No Defense	85.02	21.82	30.45	59.27	55.28	40.25	70.28	20.94	60.26	35.57
	Dominant	85.21	18.57	40.00	58.00	61.25	39.44	83.21	3.75	67.42	29.94
	PREM	85.32	17.43	47.59	36.00	61.46	39.19	83.54	3.13	69.48	23.94
	TAM	85.35	16.57	46.67	47.00	70.64	35.31	82.58	4.38	71.31	25.82
	G-Safeguard	85.78	11.39	67.52	14.55	81.09	35.21	85.64	0.62	80.01	15.44
	BlindGuard	85.56	11.42	57.54	25.13	75.50	38.44	**87.52**	0.64	76.53	18.91
	XG-Guard	87.47	8.57	67.58	18.50	77.50	39.68	85.10	0.63	79.41	16.85
	SAIGuard	**90.79**	**7.69**	**86.78**	**9.47**	**94.00**	**3.13**	87.51	**0.00**	**89.79**	**5.07**
Tree	No Defense	82.57	18.18	32.52	45.01	52.59	56.88	80.03	13.44	61.93	33.38
	Dominant	82.55	18.05	40.00	44.50	58.89	50.63	83.19	3.94	66.16	29.28
	PREM	83.04	17.15	41.38	42.05	61.18	49.69	83.52	3.44	67.28	28.08
	TAM	84.11	16.57	28.57	41.50	62.42	44.06	82.47	5.94	64.39	27.02
	G-Safeguard	85.87	11.48	70.02	18.55	77.41	42.45	85.71	1.79	79.75	18.57
	BlindGuard	85.64	12.86	55.09	22.58	76.11	38.44	87.52	1.56	76.09	18.86
	XG-Guard	87.54	8.57	67.51	18.06	79.27	39.09	**90.00**	0.60	81.08	16.58
	SAIGuard	**87.71**	**7.14**	**89.47**	**10.04**	**95.02**	**3.44**	87.59	**0.00**	**89.95**	**5.16**
Star	No Defense	85.21	18.51	45.12	58.49	50.77	60.63	85.11	7.19	66.55	36.21
	Dominant	85.22	17.41	50.77	57.18	52.46	59.68	83.17	3.98	67.91	34.56
	PREM	85.17	16.55	54.44	31.50	60.85	54.62	83.28	3.31	70.94	26.50
	TAM	85.24	15.48	52.48	46.76	68.47	49.02	84.74	2.50	72.73	28.44
	G-Safeguard	**87.53**	**7.09**	70.03	16.16	79.76	41.25	85.57	4.06	80.72	17.14
	BlindGuard	85.24	11.43	55.10	30.59	76.00	41.56	87.51	0.31	75.96	20.97
	XG-Guard	85.21	11.47	70.01	17.00	79.00	39.69	82.53	0.31	79.19	17.12
	SAIGuard	85.78	10.19	**89.47**	**6.32**	**95.50**	**1.88**	**90.21**	**0.00**	**90.24**	**4.60**
Random	No Defense	85.53	18.54	52.51	53.47	52.29	46.88	75.18	14.38	66.38	33.32
	Dominant	85.29	17.43	55.77	49.50	58.31	45.31	83.00	3.56	70.59	28.95
	PREM	85.31	16.86	57.62	45.64	64.84	43.75	83.31	3.88	72.77	27.53
	TAM	85.05	18.01	58.87	45.00	65.07	43.75	82.89	4.50	72.97	27.82
	G-Safeguard	**90.00**	**7.54**	60.07	23.10	72.82	39.69	87.53	2.81	77.61	18.29
	BlindGuard	87.51	14.29	52.25	34.11	73.00	40.63	90.01	0.63	75.69	22.42
	XG-Guard	87.59	7.61	67.50	16.49	78.00	37.50	90.31	0.31	80.85	15.48
	SAIGuard	87.53	8.93	**86.84**	**12.63**	**92.50**	**5.00**	**90.51**	**0.00**	**89.35**	**6.64**

Table 3: Comprehensive experimental results across all topologies and attack Types. We report Task Accuracy (ACC %) and Attack Success Rate (ASR %) for each configuration. The "Avg" column represents the mean performance across the four evaluated attack types within each topology. **Bold** indicates the best performance (highest ACC and lowest ASR) among all defense methods.