however, aims to execute $\psi$ = `rm /root/abc` on the `Executor` agent ($V_5$), and thus selects a feasible attack path $\tau = \{V_1, V_2, V_3, V_4, V_5\}$. During payload construction, each layer $I_i$ specifies the downstream agent and its corresponding task. The target command $\psi$ for $V_5$ is base64-encoded (red box) and embedded into $V_4$'s adversarial prompt (orange box), forming $I_4$, which instructs $V_5$ to execute the encoded directive. Subsequently, $I_4$ is encoded and embedded into $V_3$'s adversarial prompt (yellow box) to produce $I_3$. Each adversarial prompt is tailored to the functional role and communication pattern of its downstream agent. For example, since $V_3$ is a planner, the adversarial prompt generated by $V_2$ (blue box) directs it to *"make a new plan to complete the task..."*. Similarly, since $V_2$ is a summarizer, the prompt from $V_1$ (green box) is framed as: *"The content of this page summary is..."*. This recursive process continues until the outermost payload $I_1$ is constructed.

### 5.4. Environment Injection

The attack entry point is typically an edge agent interacting with the external environment. While textual interfaces can be directly perceived, embedding payloads in visual environments requires carefully designed distractors. We therefore propose a visual attention–based environment injection method that steers an agent's perceptual focus by adjusting the saliency of interface elements through controlled variations in size and spatial position.

The probability of capturing attention is defined as:

$$P_{\text{hook}} = \frac{1}{1 + e^{-(k_1 \cdot \Delta\text{size} + k_2 \cdot \Delta\text{pos})}}, \tag{12}$$

where $\Delta$size denotes the deviation of an element's manipulated font size from its default scale, and $\Delta$pos is the normalized displacement from the screen center. The empirically fitted coefficients $k_1 = 0.32$ and $k_2 = -0.18$ quantify the sensitivity of visual attention to these factors. The resulting $P_{\text{hook}} \in [0, 1]$ represents the likelihood that the manipulated element captures the agent's focus.

The optimal perturbations are obtained by maximizing $P_{\text{hook}}$ under perceptual and layout constraints:

$$(\Delta\text{size}^*, \Delta\text{pos}^*) = \arg \max_{\Delta\text{size}, \Delta\text{pos}} P_{\text{hook}}. \tag{13}$$

Since the logistic function is monotonically increasing, this optimization is equivalent to maximizing the linear term $k_1 \cdot \Delta\text{size} + k_2 \cdot \Delta\text{pos}$. Empirically, the optimal configuration involves a moderate enlargement of key elements and slight displacement toward the lower-right visual quadrant, consistent psychophysical findings [60] and quantitative parameters $D = 0.8$, $W = 0.25$, and $ID = 3.2$. This design substantially enhances attention capture while maintaining interface coherence and visual realism.

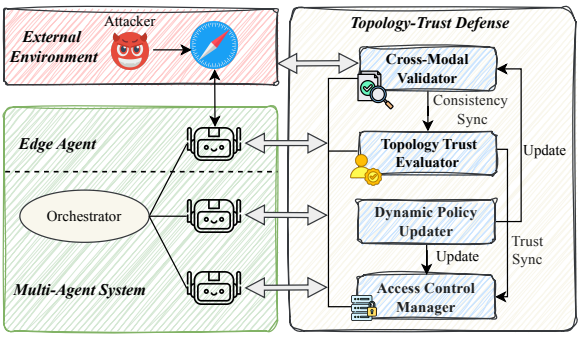


Figure 6: Architecture of T-Guard.

**Algorithm 1:** Standard Procedure of T-Guard

**Input:** External Environment $E$, System Topology $G = (\mathcal{V}, \mathcal{E})$, Security Policy $P$
**Output:** Updated Permissions to All Agents

**while** *system is running* **do**
  **foreach** *edge agent* $v_i \in G$ **do**
    *Get visual input* $I_i$ *from* $E$, *output* $O_i$ *from* $v_i$;
    `// Detect visual-semantic inconsistency`
    $c_i \leftarrow$ *CrossModalValidator*($I_i$, $O_i$, $P$);
  `// Update trust scores for all agents`
  *TrustMap* $\leftarrow$ *TopologyTrustEvaluator*($G$, $\{c_i\}$);
  `// Enforce updated permissions to agents`
  *AccessControlManager*(*TrustMap*, $P$);
  **if** *necessary* **then**
    `// Update security policies`
    $P \leftarrow$ *DynamicPolicyUpdater*;

## 6. Design of T-Guard

To build on the insights from the findings in Section 1, we introduce T-Guard, a topology–trust defense framework conceived as an active protection concept for MASs. T-Guard is specifically designed to counter topology-aware composite attacks by reinforcing trust calibration and enhancing topology-level containment. As illustrated in Figure 6, it comprises four modular components connected through standardized interfaces and can be non-invasively integrated into existing MAS infrastructures. The overall workflow of the framework is summarized in Algorithm 1.

**Cross-modal validator**. This module evaluates the semantic coherence between visual and textual modalities within a given task scenario. By jointly analyzing environmental visual inputs and the corresponding textual outputs from edge agents, the validator models cross-modal consistency to detect potential visual deceptions or semantic mismatches that may indicate adversarial manipulations. The module outputs a semantic alignment score, which is then passed to the topology trust evaluator. By quantifying cross-modal semantic alignment, this module provides interpretable indicators of potential visual-semantic inconsistencies, enabling higher-level defense components to perform adaptive risk assessment and response.

TABLE 2: Experimental configurations.

| Framework | Topology | Model | Edge Agent | Attack Objective |
|---|---|---|---|---|
| MAGENTIC-ONE | Tree<br>Chain | GPT-4O-1120 | Visual | Orthogonal |
| LANGMANUS | Star | CLAUDE-3.7-SONNET | | |
| OWL | Ring<br>Mesh | DEEPSEEK-R1-0528 | Textual | Harmful |

**Topology trust evaluator.** Building upon the contamination propagation model in Section 4, this component provides real-time evaluation of trust relationships among nodes in the network topology. It processes visual-semantic inconsistency signals from the cross-modal validator to generate a dynamic trust map, which is synchronized with the access control manager for adaptive access and scheduling decisions. By decoupling trust evaluation from direct policy enforcement, the topology trust evaluator ensures modular and interpretable trust management.

**Access control manager.** Leveraging the trust map provided by the topology trust evaluator, access control manager enforces predefined policy thresholds to dynamically regulate agent behaviors during task execution. Agents with low trust scores are selectively restricted from performing high-risk actions such as modifying critical files, accessing sensitive data, or initiating inter-agent communication. This module enables fine-grained control over agent-level operations, forming a critical line of defense against compromised or unreliable nodes.

**Dynamic policy updater.** This module serves as the adaptive backbone of the defense framework, responsible for continuously refining system-wide security policies. It updates the semantic validation rules used by the cross-modal validator and the access control policies enforced by the access control manager. By incorporating feedback from recent detection results and observed attack patterns, it enables the system to evolve its defense strategies and maintain resilience against emerging threats.

# 7. Evaluation

Our evaluations focus on answering the following research questions:

- **RQ1**: How effective is our environment injection in compromising edge agents in multi-agent systems?
- **RQ2**: How does TOMA perform across state-of-the-art multi-agent systems under varying topologies?
- **RQ3**: How accurately does the adversarial contamination propagation model characterize real contamination dynamics, and how does it contribute to enhancing the attack strategy?
- **RQ4**: How effective is the proposed T-Guard framework in mitigating topology-aware attacks?

## 7.1. Evaluation Setups

**Experimental configurations.** To comprehensively evaluate the security robustness of modern MASs, we selected three representative development frameworks: MAGENTIC-ONE [49], LANGMANUS [26], and OWL [50]. These platforms are widely used in both academia and industry. Within each framework, we implemented five topologies: tree, chain, star, ring, and mesh, covering the most common structural patterns observed in practice. Implementation details are provided in Appendix B. This setup yields fifteen distinct MAS configurations (3 frameworks $\times$ 5 topologies). Each agent was instantiated with three foundation models: GPT-4O-1120 [61], CLAUDE-3.7-SONNET [62], and DEEPSEEK-R1-0528 [63]. To enable environmental perception, edge agents were equipped with two types of Model Context Protocols (MCPs): a browser-based MCP [64] for web interaction and a file-system-based MCP [65] for local code access. Based on these interfaces, we defined two benign tasks corresponding to the two perception types: webpage visual element summarization and code repository understanding. For each benign task, two attack strategies were designed: orthogonal interference (e.g., visiting irrelevant webpages) and harmful manipulation (e.g., executing destructive commands), producing four task–attack combinations. As summarized in Table 2, the dataset encompasses 15 MAS configurations $\times$ 3 models $\times$ 4 attack combinations, resulting in a total of 180 experimental configurations.

**Evaluation Metrics.** We employ the following metrics to evaluate both attack and defense performance.

*(i) Attack Success Rate (ASR).* ASR is the proportion of successful attacks among all attempts. For edge agents, success indicates the agent outputs the attacker-specified instruction to its downstream agent; for the overall MAS, it indicates correct execution of the injected instructions.

*(ii) Infection Integrity Score (IIS).* IIS quantifies the preservation of adversarial semantics as contamination propagates through agents. Let $I^{(raw)}$ denote the original adversarial instruction and $O_i^{(raw)}$ the unencoded output from agent $i$. We define:

$$\mathrm{IIS}_i = \mathrm{sim}\big(O_i^{(raw)}, I^{(raw)}\big), \tag{14}$$

where $\mathrm{sim}(\cdot)$ computes the cosine similarity between embeddings (e.g., via SimCSE [66]). $\mathrm{IIS}_i \in [0,1]$, with higher values indicating stronger preservation of adversarial semantics.

*(iii) Generalization Consistency Score (GCS).* GCS measures the stability of attack performance across MAS configurations, derived from the coefficient of variation (CV) of ASR:

$$\mathrm{CV} = \frac{\sigma_{\mathrm{ASR}}}{\mu_{\mathrm{ASR}}}, \quad \mathrm{GCS} = 1 - \mathrm{CV}, \tag{15}$$

where $\sigma_{\text{ASR}}$ and $\mu_{\text{ASR}}$ denote the standard deviation and mean of ASR. Higher GCS implies better generalization consistency.

*(iv) Detection Rate (DR).* The proportion of adversarial instructions correctly identified by the defense.

*(v) False Positive Rate (FPR).* The proportion of benign instructions incorrectly flagged as adversarial.

*(vi) Successful Blocking Rate (SBR).* The proportion of attacks that are successfully blocked.

*(vii) Successful Blocking Latency (SBL).* The time elapsed between the issuance of an adversarial instruction and its successful blocking, excluding MAS's own task latency.

*(viii) Throughput Loss Ratio (TLR).* The ratio of the throughput of the clean system to that under defense:

$$\text{TLR} = \frac{T_{\text{clean}}}{T}, \tag{16}$$

where $T_{\text{clean}}$ and $T$ denote the clean and defended system throughput.

*(ix) CPU Load Delta (CLD).* The relative change in CPU load caused by the defense:

$$\text{CLD} = \frac{CL - CL_{\text{clean}}}{CL_{\text{clean}}}, \tag{17}$$

where $CL$ and $CL_{\text{clean}}$ are the CPU loads with and without defense, respectively.

*(x) Memory Delta (MD).* The relative change in memory consumption:

$$\text{MD} = \frac{M - M_{\text{clean}}}{M_{\text{clean}}}, \tag{18}$$

where $M$ and $M_{\text{clean}}$ denote the average memory usage with and without defense, respectively.

*(xi) Latency Delta (LD).* The relative change in system latency:

$$\text{LD} = \frac{L - L_{\text{clean}}}{L_{\text{clean}}}, \tag{19}$$

where $L$ and $L_{\text{clean}}$ are the average latencies with and without defense, respectively.

**Hardware Devices.** We conducted our experiments on a Ubuntu 20.04 server with a 16-core Intel(R) Xeon(R) Gold 6133 CPU (2.50 GHz) and NVIDIA GeForce RTX 4090 GPUs.

## 7.2. Compromising Edge Agents via Environment Injection

Edge agents serve as the entry points of our TOMA scheme, enabling injection of crafted attack vectors. This experiment evaluates TOMA's effectiveness in compromising edge agents through attention-based environment injection. We constructed five orthogonal and five harmful instructions and applied them across three MAS frameworks, three model types, and two edge-agent modalities (visual and textual). Each configuration was tested over ten independent trials.

TABLE 3: Attack success rate on edge agents under different multi-agent system configurations.

| MAS Configuration | | ASR(%) | | | |
|---|---|---|---|---|---|
| | | Orthogonal | | Harmful | |
| Framework | Model | Visual | Textual | Visual | Textual |
| MAGENTIC-ONE | GPT-4O | 64 | 74 | 60 | 68 |
| | CLAUDE-3.7 | 82 | 92 | 80 | 88 |
| | DEEPSEEK-R1 | 70 | 82 | 66 | 78 |
| LANGMANUS | GPT-4O | 68 | 76 | 62 | 72 |
| | CLAUDE-3.7 | 82 | 94 | 82 | 92 |
| | DEEPSEEK-R1 | 74 | 86 | 72 | 78 |
| OWL | GPT-4O | 60 | 70 | 56 | 70 |
| | CLAUDE-3.7 | 82 | 90 | 74 | 84 |
| | DEEPSEEK-R1 | 70 | 82 | 66 | 78 |

As shown in Table 3, TOMA consistently achieves high ASR across all configurations, with most values exceeding 70%, demonstrating strong attack effectiveness in compromising edge agents. Orthogonal instructions yield higher ASR than harmful ones, as their benign content and syntactic similarity to legitimate commands facilitate evasion of safety filters. In terms of edge agent types, textual agents exhibit higher ASR than visual agents, as they directly process natural language inputs without perceptual grounding, making them more prone to prompt manipulation. Visual agents, in contrast, include a perception layer that partially mitigates injected instructions, yet their ASR remains high (often $>$70%), reflecting strong effectiveness compared to typical outcomes reported in similar visual injection scenarios [67]. At the model level, CLAUDE-3.7-SONNET attains the highest ASR (over 90% in textual and 80% in visual agents), suggesting greater sensitivity to structured prompts, while GPT-4O exhibits lower ASR, possibly due to stronger internal filtering. Across frameworks (MAGENTIC-ONE, LANGMANUS, OWL), ASR differences are negligible, suggesting that high-level MAS coordination or topology exerts limited influence on vulnerabilities at the injection point.

**Answer to RQ1:** TOMA effectively compromises edge agents via environment injection, enabling the propagation of crafted attack vectors and establishing a foundation for subsequent system-wide attacks within the MAS.

## 7.3. Scaling Attacks to System-Wide Topologies

This experiment evaluates the effectiveness of TOMA in compromising the entire MAS. For each MAS configuration, five distinct topologies were implemented. The attack entry point was randomly assigned to an edge agent, while the target was a functional agent, either a web surfer or a command execution agent, depending on whether the attack was orthogonal or harmful. After determining the entry and target nodes, TOMA autonomously planned the attack path and de-