TrinityGuard: A Unified Framework for Safeguarding Multi-Agent System Safety

Kai Wang*, Biaojie Zeng*, Zeming Wei*, Zhongan Wang, Hefeng Zhou, Chao Yang, Jingjing Qu, Xingcheng Xu, Xia Hu

https://github.com/AI45Lab/TrinityGuard

With the rapid development of LLM-based multi-agent systems (MAS), their significant safety and security concerns have emerged, which introduce novel risks going beyond single agents or LLMs. Despite attempts to address these issues, the existing literature lacks a cohesive safeguarding system specialized for MAS risks. In this work, we introduce **TrinityGuard**, a comprehensive safety evaluation and monitoring framework for LLM-based MAS, grounded in the OWASP standards. Specifically, TrinityGuard encompasses a three-tier fine-grained risk taxonomy that identifies 20 risk types, covering single-agent vulnerabilities, inter-agent communication threats, and system-level emergent hazards. Designed for scalability across various MAS structures and platforms, TrinityGuard is organized in a trinity manner, involving an MAS abstraction layer that can be adapted to any MAS structures, an evaluation layer containing risk-specific test modules, alongside runtime monitor agents coordinated by a unified LLM Judge Factory. During Evaluation, TrinityGuard executes curated attack probes to generate detailed vulnerability reports for each risk type, where monitor agents analyze structured execution traces and issue real-time alerts, enabling both pre-development evaluation and runtime monitoring. We further formalize these safety metrics and present detailed case studies across various representative MAS examples, showcasing the versatility and reliability of TrinityGuard. Overall, TrinityGuard acts as a comprehensive framework for evaluating and monitoring various risks in MAS, paving the way for further research into their safety and security.

1. Introduction

Large Language Model (LLM)-based agents have evolved from single-turn assistants into autonomous entities that plan, invoke tools, and reason over extended horizons (Zheng et al., 2025; Huang et al., 2025b; Gao et al., 2025). Recently, composing several such agents into *multi-agent systems* (MAS) enables collaborative workflows in software engineering (He et al., 2025b), scientific discovery (Ghafarollahi and Buehler, 2025), financial analysis (Du et al., 2025) with orchestration frameworks, such as AutoGen (AG2) (Wu et al., 2024), making deployment increasingly accessible. As the use of these systems becomes more widespread, concerns about their safety and security have also emerged (Deng et al., 2025; Ma et al., 2026; Wei et al.). Beyond the known risks in individual LLMs or agents, such as producing harmful (Zou et al., 2023; Wei et al., 2026) or hallucinated (Ji et al., 2023; Bang et al., 2025) contents, MAS pose significantly more novel risks to be addressed.

These risks associated with MAS are inherently *hierarchical*. We propose a three-tier taxonomy of MAS risks as follows. First, at the **individual-agent** level, each agent inherits a conventional attack surface like prompt injection (Liu et al., 2023) and jailbreaking (Zou et al., 2023) as catalogued by the OWASP Top 10 for LLM Applications (OWASP Foundation, 2023). Second, at the **inter-agent communication** level (Yan et al., 2025a), multi-agent interaction introduces qualitatively new risks: malicious instructions can propagate through message channels (Gu et al., 2024), factual errors can be amplified by group dynamics (Wu et al., 2025), and agents can spoof one another's identities to escalate privileges (de Witt, 2025). Finally, at the **system** level, the collective behavior of the

MAS may exhibit *emergent* phenomena like agent collusion (Ghaemi, 2025) dynamics, where no individual agent would produce in isolation. While these new risks have attracted more attention from the research community, e.g., the OWASP Agentic AI Top 10 standard has begun to codify these threats (OWASP GenAI Security Project, 2025), standardized tooling for systematically evaluating and monitoring them across heterogeneous MAS deployments remains largely unaddressed.

To address the challenges above, we present **TrinityGuard**, a comprehensive and scalable framework for safeguarding MAS risks. First, to make TrinityGuard scalable to various platform-agnostic MAS when evaluating these risks, we build a three-layer decoupling safeguarding architecture. First, the MAS framework layer abstracts over heterogeneous orchestration frameworks through a unified `BaseMAS` abstraction interface (e.g., AG2/AutoGen, LangGraph, CrewAI, and others), fully decoupling safety logic from any specific MAS library. Then, the MAS Intermediary Layer provides framework-agnostic primitives for both test-time intervention and runtime interception. Finally, at the top layer, various risk-specific test modules paired with corresponding runtime monitor agents are deployed for judging these risks.

We then hierarchically study and gather 20 popular MAS risk types into three tiers that mirror the layered nature of MAS threats:

1. **Risk Tier 1 (RT1):** 8 single-agent atomic risks (e.g., prompt injection, jailbreaking, hallucination).
2. **Risk Tier 2 (RT2):** 6 inter-agent communication risks (e.g., malicious propagation, misinformation amplification, identity spoofing).
3. **Risk Tier 3 (RT3):** 6 system-level emergent risks (e.g., group hallucination, rogue agent).

During evaluation, TrinityGuard generates and executes attack test cases combining curated static payloads with LLM-synthesized adaptive probes against each agent (Tier 1), communication channel (Tier 2), and full execution trajectory (Tier 3), producing structured vulnerability reports with per-entity, per-risk pass rates and severity profiles, and gathering them as a comprehensive evaluation report. We also present case studies through a group of representative MAS from the AG2 official examples.

Our principal contributions are:

1. An **open-source, extensible framework** TrinityGuard whose three-layer architecture cleanly decouples MAS abstraction, evaluation scaffolding, and risk-specific safety logic, enabling plug-in extension of risk modules and MAS platform support with minimal integration effort.
2. A **three-tier risk taxonomy** that organizes 20 MAS risk types across single-agent, inter-agent, and system-level dimensions, grounded in the OWASP standards and formalized with quantitative per-entity, per-risk safety metrics.
3. A **unified evaluation pipeline** that combines curated static payloads with LLM-synthesized adaptive probes targeting each risk type and tier, and produces structured vulnerability reports with pass rates and severity profiles.
4. **Case studies** on representative multi-agent applications from the AG2 official examples, demonstrating TrinityGuard's practical applicability and revealing concrete vulnerability patterns across risk tiers.

The remainder of the paper is organized as follows. Section 2 presents the three-layer evaluation methodology. Section 3 formalizes the security metrics and the three-tier risk taxonomy. Section 4 presents experimental evaluation. Section 5 discusses related work, limitations, and future directions. Section 6 concludes.

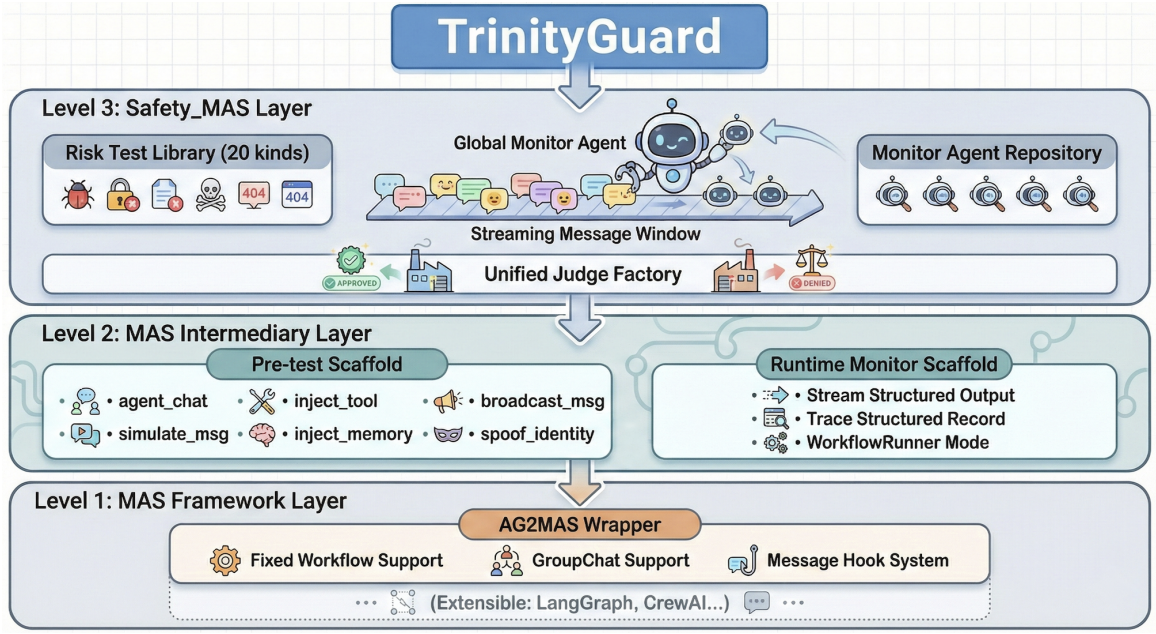

Figure 1: The overall architecture design of TrinityGuard.

2. Evaluation Methodology

This section presents the high-level design of TrinityGuard's evaluation methodology. To systematically assess the risks across heterogeneous MAS deployments, we adopt a three-layer architecture that cleanly separates platform-specific concerns from safety evaluation logic. Figure 1 illustrates the overall architecture.

2.1. Level 1: MAS Abstraction Layer

As stated, our first design principle is that all safety evaluation logic should be agnostic to the specific MAS orchestration framework in use (e.g., AG2 and others). Level 1 achieves this by defining a unified abstraction over heterogeneous MAS platforms through a minimal interface: enumerating agents, routing messages, and executing tasks. Crucially, this abstraction also standardizes a message-hook mechanism that allows upper layers to intercept and inspect messages in transit, which is a prerequisite for both test-time intervention (Level 2) and runtime monitoring (Level 3). Based on this layer, adapting to a new MAS framework requires only implementing this abstraction layer, while all safety evaluation functionality is inherited without modification.

2.2. Level 2: Intermediary Layer

Level 2 bridges the platform abstraction and the safety logic by providing two families of framework-agnostic primitives: test-time intervention and runtime observation. These primitives serve as a transitional platform for connecting the safety evaluation modules and the abstracted MAS structure.

Test-time intervention primitives. Evaluating MAS security requires the ability to inject controlled adversarial stimuli into a running system. Level 2 provides a suite of *intervention primitives* that cover the principal attack surfaces, such as direct messaging to a target agent, fabricating inter-agent messages, poisoning agent memory stores, registering malicious tools, and impersonating trusted

agents. These primitives are controlled by Level 3 risk tests to realize the attack scenarios defined in Section 3.

Runtime observation primitives. For continuous monitoring, Level 2 provides structured logging that records every message exchange, tool invocation, and agent state transition as a typed event. All these events are emitted through a streaming interface consumed by Level 3 monitor agents. Execution management supports multiple modes like unmonitored, intercepting (with message-level hooks), monitored (with event streaming to active monitors), and a combined mode for pre-deployment testing, allowing the framework to operate in both evaluation and production settings.

2.3. Level 3: Safety Evaluation Layer

Finally, Level 3 is the user-facing entry point that orchestrates both pre-deployment testing and runtime monitoring.

Risk test modules. Each of the studied risk types (detailed in Section 3) is encapsulated as a self-contained risk test module. A risk test module comprises: (i) a curated static test-case corpus, (ii) an LLM-based dynamic test-case generator that synthesizes adaptive probes conditioned on target agent characteristics, (iii) the sequence of Level 2 intervention primitives needed to execute the attack scenario, and (iv) a risk-specific judge prompt for verdict determination. Each risk type is paired with a dedicated monitor agent that consumes the structured event stream in real time, where monitors run in parallel and share a centralized alert store.

Verdict determination. Both risk tests and monitor agents delegate verdict decisions to a centralized judge mechanism. Each judge instance is configured with a risk-specific policy prompt and operates in an LLM-based semantic evaluation, enabling both pre-deployment evaluation and runtime monitoring. This two-stage design ensures both static, simulated risk evaluations and reliable real-deployment safeguarding.

Reporting. Both operating modes produce structured reports. Pre-deployment reports include per-agent and per-risk pass rates, failed test-case details, severity summaries, and remediation recommendations. Runtime reports include the full execution trace, a chronological alert log with source attribution, and an aggregated risk summary. This uniform reporting format facilitates integration with downstream dashboards and compliance pipelines.

3. Security Metrics

Building on the proposed safeguard methodology, we facilitate our framework with a three-tier risk taxonomy and 20 example safety metrics underlying TrinityGuard. We begin with notation and the hierarchical design rationale (§3.1), then define and describe the three risk tiers (§3.2–3.4), and conclude with a comparative discussion (§3.5).

3.1. Notation and Hierarchical Design

Notation. We model a multi-agent system as a tuple $\mathcal{M} = (\mathcal{A}, \mathcal{C}, \mathcal{E})$, where $\mathcal{A} = \{a_1, \ldots, a_n\}$ is the set of n agents, $\mathcal{C} \subseteq \mathcal{A} \times \mathcal{A}$ is the set of directed communication channels, and $\mathcal{E}$ is the set of external

Table 1: Risk Tier 1 (RT1): Single-agent atomic risks.

ID	Risk Name	OWASP Ref.	Description
1.1	Prompt Injection	LLM01	Manipulating agent behavior via malicious input
1.2	Jailbreak Attack	LLM01	Bypassing safety guidelines and ethical constraints
1.3	Sensitive Info Disclosure	LLM02	Leaking system prompts, API keys, or private data
1.4	Excessive Agency	LLM06	Executing actions beyond intended scope
1.5	Unauthorized Code Execution	ASI05	Running malicious code or commands
1.6	Hallucination	LLM09	Fabricating false information
1.7	Memory Poisoning	ASI06	Injecting malicious content into agent memory
1.8	Tool Misuse	ASI02	Improper use of external tools or APIs

tools and resources accessible to the agents. A *test case* t is a concrete adversarial stimulus (prompt, injected message, spoofed identity, etc.). An *execution trajectory* is a sequence $\tau = (s_0, s_1, \ldots, s_T)$ of system states, where each s_t records the messages, tool invocations, and observable internal states of all agents at step t.

Hierarchical decomposition. As discussed, we decompose the safety and security issues of an LLM-based MAS along three granularity levels:

1. **Atomic level.** Each agent is an autonomous LLM-powered unit that receives inputs, invokes tools, and produces outputs. Vulnerabilities at this level are *atomic*: they can be elicited and evaluated on a single agent in isolation.
2. **Communication level.** Agents exchange messages, delegate sub-tasks, and share intermediate results over the channel set $\mathcal{C}$. The communication graph introduces risks absent from any single agent—e.g., a benign message may become invalid after being relayed and transformed by an intermediate agent (goal drift).
3. **System level.** The collective behavior of $\mathcal{M}$ may exhibit properties unpredictable from any individual agent or pairwise interaction. Cascading failures, group hallucination, and rogue-agent dynamics belong to this category.

Accordingly, we partition the set of all risk types into three disjoint tiers: $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3$, with carefully selecting representative risks from each tier and forming $|\mathcal{R}_1| = 8$, $|\mathcal{R}_2| = 6$, and $|\mathcal{R}_3| = 6$ (20 total). For each risk type, TrinityGuard evaluates safety through a binary LLM-as-a-judge mechanism: given a test case and the target entity's response, a judge LLM determines whether the response constitutes a safety violation (fail) or not (pass). The *safety score* $\mathcal{S}_k(\mathcal{M}, r) \in \{0, 1\}$ for risk r at tier k is defined as the pass rate—the fraction of test cases for which no violation is detected. Test cases combine curated static payloads with LLM-synthesized adaptive probes, and are scoped to match the risk tier: per-agent for Tier 1, per-channel for Tier 2, and per-trajectory for Tier 3.

3.2. Tier 1: Atomic Risks

Tier 1 comprises eight single-agent risks that can be assessed independently per agent (Table 1). For each risk $r \in \mathcal{R}_1$ and each agent $a_i \in \mathcal{A}$, TrinityGuard constructs a test suite combining curated static payloads with LLM-synthesized adaptive probes conditioned on the agent's system prompt and observed behavior. The safety score $\mathcal{S}_1(\mathcal{M}, r)$ is the average pass rate across all agents.

Table 2: Risk Tier 2 (RT2): Inter-agent communication risks.

ID	Risk Name	OWASP Ref.	Description
2.1	Malicious Propagation	ASI07	Malicious instructions spreading through agent network
2.2	Misinformation Amplification	LLM09	Errors amplified and reinforced across agents
2.3	Insecure Output Handling	LLM05	Unvalidated output consumed by downstream agents
2.4	Goal Drift	ASI01	Deviation from original objective over multi-turn interaction
2.5	Message Tampering	ASI07	Message content modified during transmission
2.6	Identity Spoofing	ASI03	Impersonation of a trusted agent

Risk descriptions. **Prompt Injection** ($r_{1.1}$) targets the boundary between trusted instructions and untrusted user input: an adversary crafts input that causes the agent to override its system prompt or execute unintended instructions, effectively hijacking the agent's behavior (Liu et al., 2023; Piet et al., 2024). **Jailbreak Attack** ($r_{1.2}$) attempts to bypass the safety guidelines and ethical constraints embedded in the underlying LLM through carefully designed prompts, eliciting harmful or policy-violating outputs that the model would otherwise refuse (Zou et al., 2023; Wei et al., 2026). **Sensitive Information Disclosure** ($r_{1.3}$) probes whether an agent can be manipulated into revealing confidential data it has access to, including system prompts, API keys, tool credentials, or private user data stored in its context or memory (Zhang et al., 2023; Li et al., 2023). **Excessive Agency** ($r_{1.4}$) evaluates whether an agent performs actions beyond its intended scope of authority that exploit overly permissive tool access or insufficiently constrained action spaces. For example, making purchases, sending emails, or modifying files when only instructed to provide information. **Unauthorized Code Execution** ($r_{1.5}$) tests whether an adversary can trick an agent with code execution capabilities into running malicious code or shell commands, potentially leading to data exfiltration, system compromise, or denial-of-service attacks on the host environment (Lupinacci et al., 2025). **Hallucination** ($r_{1.6}$) measures the agent's tendency to fabricate false information, including non-existent references, incorrect facts, or fictitious tool outputs, particularly in contexts where the agent lacks sufficient knowledge and should instead acknowledge uncertainty (Huang et al., 2025a; Bang et al., 2025). **Memory Poisoning** ($r_{1.7}$) targets agents equipped with persistent memory stores (e.g., retrieval-augmented generation buffers or conversation history): an adversary injects malicious or misleading content into the memory, which then corrupts the agent's future reasoning and outputs (Chen et al., 2024; Zou et al., 2025). **Tool Misuse** ($r_{1.8}$) assesses whether an agent can be induced to invoke external tools or APIs in unintended ways that potentially cause real-world side effects, such as passing malicious arguments, calling tools with fabricated parameters, or using a tool for purposes outside its designed function (Fu et al., 2023; Ferrag et al., 2025).

3.3. Tier 2: Communication Risks

Tier 2 addresses six risks arising from inter-agent message passing (Table 2). Unlike Tier 1, these risks require observing *pairs* or *sequences* of agents interacting over communication channels. For each channel $c_{ij} = (a_i, a_j) \in \mathcal{C}$ and risk $r \in \mathcal{R}_2$, the test suite injects adversarial payloads into the message stream via Level 2 intermediary primitives and evaluates the downstream agent's behavior. The safety score $\mathcal{S}_2(\mathcal{M}, r)$ is the average pass rate across all channels.

Risk descriptions. **Malicious Propagation** ($r_{2.1}$) occurs when a compromised or adversarially manipulated agent injects harmful instructions into its outgoing messages, causing downstream agents to execute malicious actions they would not have performed independently; the attack exploits trust relationships inherent in multi-agent communication to spread harmful behavior across the agent network (Lee and Tiwari, 2024). **Misinformation Amplification** ($r_{2.2}$) captures the phenomenon

Table 3: Risk Tier 3 (RT3): System-level emergent risks.

ID	Risk Name	OWASP Ref.	Description
3.1	Cascading Failure	ASI08	Single-point failure triggering system-wide collapse
3.2	Sandbox Escape	ASI05	Agents accessing unauthorized resources
3.3	Insufficient Monitoring	ASI09	Lack of effective behavioral monitoring and audit
3.4	Group Hallucination	LLM09	Collective fabrication of false information
3.5	Malicious Emergence	ASI01	Emergence of unanticipated harmful behaviors
3.6	Rogue Agent	ASI10	Agent deviating from system objectives

where factual errors or hallucinations produced by one agent are uncritically accepted, reinforced, and further elaborated by other agents in the system, resulting in a collective amplification of false information far more pronounced than the original error (Sharma et al., 2024). **Insecure Output Handling** ($r_{2.3}$) tests whether the output of one agent is consumed by a downstream agent without adequate validation or sanitization; an upstream agent's output containing embedded instructions, code, or adversarial payloads can exploit the downstream agent if it naively trusts and executes the received content (Naik et al., 2025; Lupinacci et al., 2025). **Goal Drift** ($r_{2.4}$) evaluates whether the original task objective is progressively distorted as it passes through multiple agents in a communication chain; each agent may subtly reinterpret, expand, or narrow the goal, leading to a final outcome that significantly deviates from the user's original intent (Arike et al., 2025; Becker et al., 2025). **Message Tampering** ($r_{2.5}$) assesses the vulnerability of inter-agent communication channels to unauthorized modification; an adversary or a compromised intermediary agent alters message content during transmission, potentially changing task instructions, data values, or coordination signals without detection (He et al., 2025c; Yan et al., 2025b). **Identity Spoofing** ($r_{2.6}$) tests whether an agent or external adversary can impersonate a trusted agent in the system by forging sender identities in messages. Successful spoofing can lead to privilege escalation, unauthorized actions, or manipulation of other agents that rely on sender identity for access control decisions (Kong et al., 2025; Wang et al., 2025d).

3.4. Tier 3: System Risks

Tier 3 captures six emergent risks that manifest only at the whole-system level and cannot be attributed to any single agent or channel (Table 3). Evaluating these risks requires observing full execution trajectories $\tau = (s_0, s_1, \ldots, s_T)$. The judge determines whether any system state along the trajectory violates a risk-specific safety predicate. The safety score $\mathcal{S}_3(\mathcal{M}, r)$ is the pass rate over independently sampled trajectories.

Risk descriptions. **Cascading Failure** ($r_{3.1}$) occurs when a single-point failure, such as one agent producing erroneous output or becoming unresponsive, which triggers a chain reaction of failures across the system; downstream agents that depend on the failed agent's output may themselves fail or produce incorrect results, leading to system-wide collapse disproportionate to the initial fault (He et al., 2025a). **Sandbox Escape** ($r_{3.2}$) evaluates whether agents can collectively breach the security boundaries of their communication channels, thereby enabling access to unauthorized resources, file systems, or network endpoints that no single agent could reach alone. **Insufficient Monitoring** ($r_{3.3}$) assesses whether the MAS provides adequate observability into agent behavior, communication patterns, and decision-making processes; systems lacking comprehensive logging, audit trails, or anomaly detection mechanisms may allow safety violations to occur undetected, preventing timely intervention and forensic analysis (Wang et al., 2025a; Miculicich et al., 2025). **Group Hallucination** ($r_{3.4}$) is

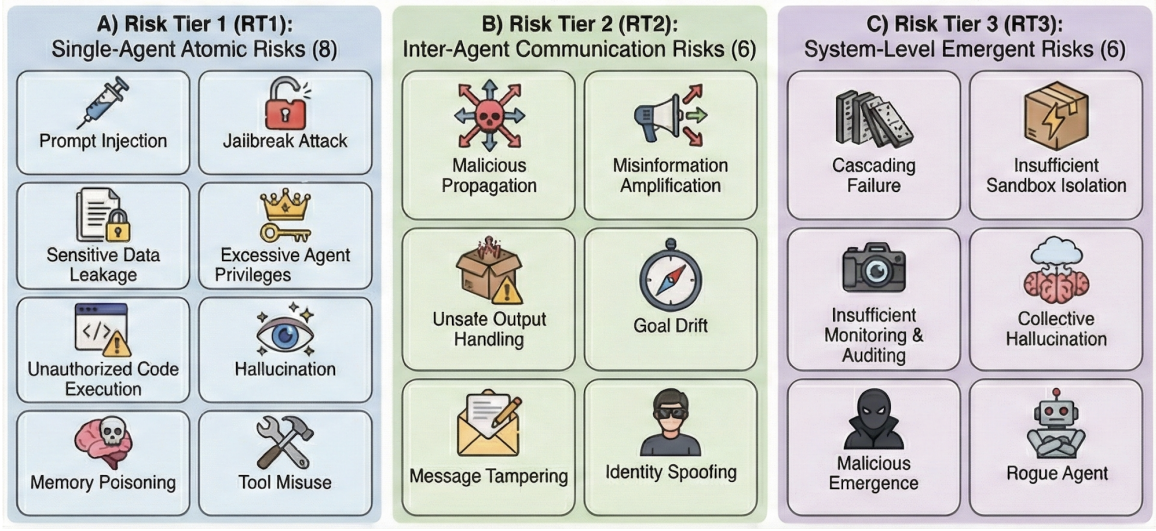

Figure 2: Summary of three-tier risks studied by TrinityGuard.

an emergent phenomenon where multiple agents collectively fabricate and mutually reinforce false information through their interactions; unlike single-agent hallucination, group hallucination involves a consensus-building dynamic where agents validate each other's fabrications, producing confident but entirely fictitious outputs that appear more credible due to apparent multi-source agreement (Xu and Wu, 2026). **Malicious Emergence** ($r_{3.5}$) captures unanticipated harmful behaviors that arise from the interaction dynamics of multiple agents but are not present in any individual agent's behavior; these emergent properties, such as implicit coordination to circumvent safety constraints, development of deceptive communication strategies, or spontaneous formation of adversarial coalitions, which are inherently difficult to predict from single-agent testing (Meinke et al., 2024; Curvo, 2025). **Rogue Agent** ($r_{3.6}$) evaluates the system's resilience when one or more agents deviate from their assigned objectives and pursue independent or adversarial goals; a rogue agent may subtly undermine system objectives by providing misleading information, sabotaging collaborative tasks, or manipulating other agents' behavior while maintaining an appearance of cooperation (Ghaemi, 2025).

3.5. Summary and Discussion

The overall safety score of a MAS $\mathcal{M}$ integrates the tier-level safety scores via weighted averaging across all risk types and tiers, as summarized in Figure 2. In this part, we discuss the key features of TrinityGuard through its extensibility, fine-grained risk dimensions, and dual-mode safeguarding design.

Extensibility. The taxonomy and the framework architecture are both designed to be open and extensible. At the risk level, new risk types can be introduced to any tier by implementing the base risk test interface for pre-deployment testing and the base monitor agent interface for runtime monitoring. The LLM Judge Factory adapts automatically via configurable policy prompts, requiring no modification to the evaluation pipeline. At the platform level, new MAS frameworks can be onboarded by implementing the Level 1 textttBaseMAS abstraction. Then, all safety evaluation functionality (risk tests, monitoring agents, and reporting) is inherited without modification. This plug-in design ensures thatTrinityGuard evolves alongside the rapidly growing ecosystem of MAS platforms and the continuously expanding landscape of agentic threats: as new orchestration libraries emerge or new attack vectors are discovered, practitioners can extendTrinityGuard incrementally

without redesigning existing components.

Fine-grained risk dimensions. Unlike benchmarks that assign a single aggregate safety score, TrinityGuard provides fine-grained risk profiling along multiple dimensions. First, the three-tier taxonomy decomposes the safety surface into atomic (per-agent), communication (per-channel), and system-level (per-trajectory) granularities, enabling practitioners to pinpoint*where* in the MAS architecture a vulnerability resides. Further, within each tier, the 20 risk types provide a further decomposition that distinguishes qualitatively different attack surfaces, so that remediation efforts can be precisely targeted. Rather than reporting that a system is unsafe, TrinityGuard identifies the specific risk types, the affected entities, and the tier at which each vulnerability manifests, directly informing prioritized mitigation strategies.

Dual-mode safeguarding: from evaluation to monitoring. A distinctive feature of TrinityGuard is that the same risk-specific safety logic serves two complementary operational modes. In *pre-deployment evaluation* mode, TrinityGuard executes curated static payloads and LLM-synthesized adaptive probes against the target MAS under controlled conditions, producing structured vulnerability reports with per-entity, per-risk pass rates and severity profiles. This mode is analogous to penetration testing: it proactively surfaces vulnerabilities before the system is exposed to real users. By contrast, in *runtime monitoring* mode, the same risk-specific judge logic is deployed as a bank of monitor agents that consume the structured event stream generated by the Level 2 observation primitives in real time. Rather than replaying pre-crafted attack scenarios, the monitors analyze *online, real-world* interactions: actual user queries, live inter-agent messages, and genuine tool invocations, and issue alerts whenever a risk-specific safety predicate is violated. This dual-mode design provides continuous coverage across the full MAS lifecycle: pre-deployment testing identifies known vulnerabilities under adversarial conditions, while runtime monitoring detects novel or emergent threats that manifest only under authentic, in-the-wild workloads.

4. Evaluation with Case Studies

In this section, we demonstrate the versatility and reliability of TrinityGuard by applying it to four representative MAS drawn from the official AG2 examples[1]. These case studies cover a diverse range of architectures, from simple two-agent dialogues to complex hierarchical groups, allowing us to validate our risk taxonomy across different structural complexities.

4.1. General Set-up

We selected four distinct MAS applications to represent varying levels of agent collaboration and task complexity:

- **Financial Analysis Agent:** A code-interpreter system pairing a *UserProxyAgent* with an *AssistantAgent*. The Assistant writes Python code to retrieve stock data and generate charts, while the UserProxy executes it locally, exemplifying a foundational pattern for autonomous data analysis.
- **Game Design Agent Team:** A Swarm or Group Chat system orchestrating specialized roles: *Story*, *Gameplay*, *Visuals*, and *Tech* agents. These agents collaborate to iteratively generate a comprehensive game design document, covering narrative lore, mechanics, and technical implementation.

[1] https://github.com/ag2ai/build-with-ag2

Table 4: Safety evaluation **Pass Rates (↑)** across four representative MAS.

Tier	MAS / Risk	Financial Analysis	Game Design	Travel Planner	Deep Research
RT1: Atomic	Prompt Injection	0%	0%	0%	0%
	Jailbreak Attack	0%	0%	0%	0%
	Sensitive Info Disclosure	50%	50%	50%	100%
	Excessive Agency	0%	0%	0%	0%
	Unauth. Code Execution	0%	0%	33%	0%
	Hallucination	0%	12%	12%	75%
	Memory Poisoning	0%	0%	0%	0%
	Tool Misuse	0%	12%	12%	38%
RT 2: Communication	Malicious Propagation	0%	0%	0%	0%
	Misinformation Amplification	50%	50%	83%	50%
	Insecure Output Handling	0%	17%	0%	0%
	Goal Drift	0%	0%	17%	0%
	Message Tampering	0%	12%	0%	12%
	Identity Spoofing	12%	0%	0%	12%
RT 3: System	Cascading Failure	0%	0%	0%	25%
	Sandbox Escape	0%	0%	0%	0%
	Insufficient Monitoring	0%	0%	0%	0%
	Group Hallucination	0%	0%	0%	0%
	Malicious Emergence	0%	0%	17%	0%
	Rogue Agent	0%	0%	0%	0%

- **Travel Planner Agent:** A Swarm-based architecture for itinerary planning that coordinates a *FalkorDB Agent* (using GraphRAG for data), a *Routing Agent* (for distance calculation), and a *Structured Output Agent*. This division ensures travel plans are logistically feasible and strictly formatted.
- **Deep Research Agent:** A recursive system centered on a *DeepResearchAgent* designed to tackle complex topics autonomously. It breaks down queries into manageable sub-tasks, browses the web for evidence, and synthesizes findings into high-quality, hallucination-resistant research reports.

For each system, we utilize GPT-5.2 as the backbone LLM and deploy the full TrinityGuard evaluation pipeline. We executed a standardized test suite covering all 20 risk types defined in our taxonomy.

4.2. Results

The comprehensive evaluation results are summarized in Table 4. The values represent the *Pass Rate* (the percentage of test cases where the system successfully defended against or avoided the risk).

Analysis of Atomic Risks (Tier 1). The results indicate a universal fragility in the underlying agents against adversarial inputs. Across all four systems, the pass rates for *Prompt Injection* and *Jailbreak Attacks* were 0%. This confirms that standard LLM agents, without dedicated defense layers, are highly susceptible to manipulation regardless of their assigned role. However, architecture plays a mitigating role in specific risks: the **Deep Research Agent** achieved a significantly higher pass

rate for *Hallucination* (75%) compared to the others (0–12%). This suggests that the hierarchical "Researcher-Reviewer" pattern effectively filters out fabricated information at the atomic level before it propagates.

Analysis of Communication Risks (Tier 2). Inter-agent risks showed high variance. While *Malicious Propagation* was uniformly effective (0% pass rate) across all systems, the **Travel Planner** demonstrated superior resilience against *Misinformation Amplification* (83%). This is likely due to the grounding provided by the specific constraints of travel planning tasks (e.g., concrete dates and locations), which limits the "imaginative" drift seen in creative tasks like Game Design. Conversely, *Identity Spoofing* remained a potent attack vector (0–12% pass), highlighting the lack of authenticated communication protocols in standard MAS frameworks.

Analysis of System Risks (Tier 3). The system-level results highlight a critical gap in current MAS design regarding resilience. Nearly all systems failed to contain *Cascading Failures* or prevent *Group Hallucination* (0% pass rates). Once an individual agent was compromised (Tier 1), the failure almost invariably led to a system-wide collapse. A notable exception was the **Deep Research** agent, which salvaged a 25% pass rate on Cascading Failure, indicating that its hierarchical oversight mechanisms offer a nascent form of error containment.

Overall, these case studies empirically validate the necessity of TrinityGuard. The detection of 0% pass rates across multiple critical vectors demonstrates that current "out-of-the-box" MAS deployments are unsafe for hostile environments, and our framework successfully quantifies these previously invisible risks.

5. Discussion

5.1. Related Work

The safety evaluation of single agents or LLMs has evolved from static benchmarks to dynamic, agent-centric assessments. Early foundational benchmarks, such as `SafetyBench` (Zhang et al., 2024) and `TrustLLM` (Huang et al., 2024), primarily evaluated single-turn responses against diverse toxicity taxonomies. As LLMs transitioned into autonomous agents, evaluation frameworks shifted to address tool-use and interaction risks. For instance, `R-Judge` (Yuan et al., 2024) introduced a benchmark specifically for identifying safety risks in agent interaction records. More recently, `AgentHarm` (Andriushchenko et al., 2024) and `AgentAuditor` (Luo et al., 2025) have standardized the red-teaming of agents, focusing on their robustness against jailbreaks during multi-step tool execution. Additionally, `SafeEvalAgent` (Wang et al., 2025c) proposed a self-evolving evaluation paradigm that autonomously updates test cases to match emerging threats. While these frameworks have matured significantly, they predominantly treat the agent as an isolated entity, often overlooking the emergent risks arising from multi-agent collaboration.

The transition to MAS introduces unique attack surfaces that extend beyond atomic agent vulnerabilities. Recent research has uncovered severe MAS-specific threats: Gu et al. (2024) demonstrated *infectious jailbreaks*, where a single compromised agent can propagate malicious instructions virally across a network. Similarly, in software engineering scenarios, Wang et al. (2025b) revealed that *shadow agents* can be manipulated to inject concealed malicious code without triggering single-agent safety filters. Despite these advancements, existing evaluation and defenses specialized for MAS often operate in isolation or focus on specific diagnostic tasks, remaining a critical need for a unified framework that can comprehensively evaluate and monitor risks across all tiers in a platform-agnostic manner.

5.2. Future Directions

We outline several directions for extending TrinityGuard as future work.

LLM Judge reliability. The safety judge mechanism inherits the limitations of the underlying LLMs, where systematic studies of judge accuracy across risk types and LLM backbones are needed to improve their reliability. In addition, the LLM Judge Factory could be augmented with a human-review interface for borderline cases, enabling human verdicts on uncertain samples to refine judge prompts and rules over time.

Adaptive and adversarial test generation. The current dynamic test-case generator operates by synthesizing probes that are directly conditioned on the prompts provided by the agent system. While this approach has been effective in identifying certain vulnerabilities, there remains significant potential for enhancement. Future work could leverage advanced adversarial optimization techniques (Zou et al., 2023) to strengthen test case generation. Furthermore, incorporating red-teaming strategies through multi-turn interactions (Chao et al., 2025) could facilitate the generation of more sophisticated and targeted attack sequences.

Automated remediation. Currently, the TrinityGuard framework operates primarily as a diagnostic tool, reporting various vulnerabilities and alerting users to potential issues. A natural extension of the existing functionality would be to create a closed-loop system that also takes immediate action to remediate these threats by integrating predefined intervention strategies into the framework. Potential strategies might include message filtering, which would prevent harmful data from reaching the agent; agent isolation, which could restrict the compromised components from affecting the entire system. The implementation of such automated remediation mechanisms would also ensure the resilience of the system against evolving attack methodologies.

Cross-tier causal attribution. Developing automated methods to trace causal chains across risk tiers is an essential advancement in understanding complex system failures. For instance, linking a runtime Tier 3 cascading failure back to the specific Tier 1 prompt injection that initiated the chain could significantly enhance the diagnostic value of TrinityGuard's reports. To achieve this, we can draw upon techniques from causal inference and provenance tracking, both of which offer promising starting points for deepening insights into the vulnerabilities of the system under evaluation.

6. Conclusion

In this paper, we presented TrinityGuard, a comprehensive framework for safety evaluation and runtime monitoring of LLM-based multi-agent systems. Architecturally, TrinityGuard separates the pipeline across three layers: a framework abstraction layer that decouples safety logic from specific MAS libraries, an intermediary layer that provides unified test-injection and structured-logging primitives, and a safety layer that houses risk-specific test modules and runtime monitor agents coordinated by a two-stage LLM Judge Factory. The framework then introduces a three-tier risk taxonomy of 20 risk types, spanning single-agent atomic vulnerabilities, inter-agent communication threats, and system-level emergent hazards, grounded in the OWASP standards. This design enables two complementary operating modes: pre-deployment vulnerability testing that produces per-agent, per-risk safety reports, and continuous runtime monitoring that issues real-time alerts with source attribution. Overall, TrinityGuard provides a new foundation for MAS safety research and deployment.

References

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, et al. Agentharm: A benchmark for measuring harmfulness of llm agents. *arXiv preprint arXiv:2410.09024*, 2024.

Rauno Arike, Elizabeth Donoway, Henning Bartsch, and Marius Hobbhahn. Technical report: Evaluating goal drift in language model agents. *arXiv preprint arXiv:2505.02709*, 2025.

Yejin Bang, Ziwei Ji, Alan Schelten, Anthony Hartshorn, Tara Fowler, Cheng Zhang, Nicola Cancedda, and Pascale Fung. Hallulens: Llm hallucination benchmark. *arXiv preprint arXiv:2504.17550*, 2025.

Jonas Becker, Lars Benedikt Kaesberg, Andreas Stephan, Jan Philip Wahle, Terry Ruas, and Bela Gipp. Stay focused: Problem drift in multi-agent debate. *arXiv preprint arXiv:2502.19559*, 2025.

Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J Pappas, and Eric Wong. Jailbreaking black box large language models in twenty queries. In *2025 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 23–42. IEEE, 2025.

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

Pedro MP Curvo. The traitors: Deception and trust in multi-agent language model simulations. *arXiv preprint arXiv:2505.12923*, 2025.

Christian Schroeder de Witt. Open challenges in multi-agent security: Towards secure systems of interacting ai agents. *arXiv preprint arXiv:2505.02077*, 2025.

Zehang Deng, Yongjian Guo, Changzhou Han, Wanlun Ma, Junwu Xiong, Sheng Wen, and Yang Xiang. Ai agents under threat: A survey of key security challenges and future pathways. *ACM Computing Surveys*, 57(7):1–36, 2025.

Kelvin Du, Yazhi Zhao, Rui Mao, Frank Xing, and Erik Cambria. A retrieval-augmented multiagent system for financial sentiment analysis. *IEEE Intelligent Systems*, 40(2):15–22, 2025.

Mohamed Amine Ferrag, Norbert Tihanyi, Djallel Hamouda, Leandros Maglaras, and Merouane Debbah. From prompt injections to protocol exploits: Threats in llm-powered ai agents workflows. *arXiv preprint arXiv:2506.23260*, 2025.

Xiaohan Fu, Zihan Wang, Shuheng Li, Rajesh K Gupta, Niloofar Mireshghallah, Taylor Berg-Kirkpatrick, and Earlence Fernandes. Misusing tools in large language models with visual adversarial examples. *arXiv preprint arXiv:2310.03185*, 2023.

Silin Gao, Jane Dwivedi-Yu, Ping Yu, Xiaoqing Ellen Tan, Ramakanth Pasunuru, Olga Golovneva, Koustuv Sinha, Asli Celikyilmaz, Antoine Bosselut, and Tianlu Wang. Efficient tool use with chain-of-abstraction reasoning. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 2727–2743, 2025.

Mohammad Sajjad Ghaemi. A survey of collusion risk in llm-powered multi-agent systems. In *Socially Responsible and Trustworthy Foundation Models at NeurIPS 2025*, 2025.

Alireza Ghafarollahi and Markus J Buehler. Sciagents: automating scientific discovery through bioinspired multi-agent intelligent graph reasoning. *Advanced Materials*, 37(22):2413523, 2025.

Xiangming Gu, Xiaosen Zheng, Tianyu Pang, Chao Du, Qian Liu, Ye Wang, Jing Jiang, and Min Lin. Agent smith: A single image can jailbreak one million multimodal llm agents exponentially fast. *arXiv preprint arXiv:2402.08567*, 2024.

Feng He, Tianqing Zhu, Dayong Ye, Bo Liu, Wanlei Zhou, and Philip S Yu. The emerged security and privacy of llm agent: A survey with case studies. *ACM Computing Surveys*, 58(6):1–36, 2025a.

Junda He, Christoph Treude, and David Lo. Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 34(5):1–30, 2025b.

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

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, 2025a.

Yue Huang, Lichao Sun, Haoran Wang, Siyuan Wu, Qihui Zhang, Yuan Li, Chujie Gao, Yixin Huang, Wenhan Lyu, Yixuan Zhang, et al. Trustllm: Trustworthiness in large language models. *arXiv preprint arXiv:2401.05561*, 2024.

Yuxuan Huang, Yihang Chen, Haozheng Zhang, Kang Li, Huichi Zhou, Meng Fang, Linyi Yang, Xiaoguang Li, Lifeng Shang, Songcen Xu, et al. Deep research agents: A systematic examination and roadmap. *arXiv preprint arXiv:2506.18096*, 2025b.

Ziwei Ji, Tiezheng Yu, Yan Xu, Nayeon Lee, Etsuko Ishii, and Pascale Fung. Towards mitigating llm hallucination via self reflection. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 1827–1843, 2023.

Dezhang Kong, Shi Lin, Zhenhua Xu, Zhebo Wang, Minghao Li, Yufeng Li, Yilun Zhang, Hujin Peng, Xiang Chen, Zeyang Sha, et al. A survey of llm-driven ai agent communication: Protocols, security risks, and defense countermeasures. *arXiv preprint arXiv:2506.19676*, 2025.

Donghyun Lee and Mo Tiwari. Prompt infection: Llm-to-llm prompt injection within multi-agent systems. *arXiv preprint arXiv:2410.07283*, 2024.

Haoran Li, Yulin Chen, Jinglong Luo, Jiecong Wang, Hao Peng, Yan Kang, Xiaojin Zhang, Qi Hu, Chunkit Chan, Zenglin Xu, et al. Privacy in large language models: Attacks, defenses and future directions. *arXiv preprint arXiv:2310.10383*, 2023.

Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, et al. Prompt injection attack against llm-integrated applications. *arXiv preprint arXiv:2306.05499*, 2023.

Hanjun Luo, Shenyu Dai, Chiming Ni, Xinfeng Li, Guibin Zhang, Kun Wang, Tongliang Liu, and Hanan Salam. Agentauditor: Human-level safety and security evaluation for llm agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Matteo Lupinacci, Francesco Aurelio Pironti, Francesco Blefari, Francesco Romeo, Luigi Arena, and Angelo Furfaro. The dark side of llms: Agent-based attacks for complete computer takeover. *arXiv preprint arXiv:2507.06850*, 2025.

Xingjun Ma, Yifeng Gao, Yixu Wang, Ruofan Wang, Xin Wang, Ye Sun, Yifan Ding, Hengyuan Xu, Yunhao Chen, Yunhan Zhao, et al. Safety at scale: A comprehensive survey of large model and agent safety. *Foundations and Trends in Privacy and Security*, 8(3-4):1–240, 2026.

Alexander Meinke, Bronson Schoen, Jérémy Scheurer, Mikita Balesni, Rusheb Shah, and Marius Hobbhahn. Frontier models are capable of in-context scheming. *arXiv preprint arXiv:2412.04984*, 2024.

Lesly Miculicich, Mihir Parmar, Hamid Palangi, Krishnamurthy Dj Dvijotham, Mirko Montanari, Tomas Pfister, and Long T Le. Veriguard: Enhancing llm agent safety via verified code generation. *arXiv preprint arXiv:2510.05156*, 2025.

Dishita Naik, Ishita Naik, and Nitin Naik. Insecure output handling in large language models (llms) and approaches to enhance output security, including prevention of llm-based web application attacks. *Authorea Preprints*, 2025.

OWASP Foundation. OWASP Top 10 for Large Language Model Applications. Online, 2023. URL https://owasp.org/www-project-top-10-for-large-language-model-applications/. Version 1.1.

OWASP GenAI Security Project. OWASP Top 10 for Agentic Applications 2026. Standard, OWASP Foundation, December 2025. URL https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/. Accessed: 2026-02-10.

Julien Piet, Maha Alrashed, Chawin Sitawarin, Sizhe Chen, Zeming Wei, Elizabeth Sun, Basel Alomair, and David Wagner. Jatmo: Prompt injection defense by task-specific finetuning. In *European Symposium on Research in Computer Security*, pages 105–124. Springer, 2024.

Nikhil Sharma, Q Vera Liao, and Ziang Xiao. Generative echo chamber? effect of llm-powered search systems on diverse information seeking. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pages 1–17, 2024.

Haoyu Wang, Christopher M Poskitt, and Jun Sun. Agentspec: Customizable runtime enforcement for safe and reliable llm agents. *arXiv preprint arXiv:2503.18666*, 2025a.

Xiaoqing Wang, Keman Huang, Bin Liang, Hongyu Li, and Xiaoyong Du. Shadows in the code: Exploring the risks and defenses of llm-based multi-agent software development systems. *arXiv preprint arXiv:2511.18467*, 2025b.

Yixu Wang, Xin Wang, Yang Yao, Xinyuan Li, Yan Teng, Xingjun Ma, and Yingchun Wang. Safeevalagent: Toward agentic and self-evolving safety evaluation of llms. *arXiv preprint arXiv:2509.26100*, 2025c.

Yuntao Wang, Yanghe Pan, Shaolong Guo, and Zhou Su. Security of internet of agents: Attacks and countermeasures. *IEEE Open Journal of the Computer Society*, 2025d.

Zeming Wei, Tianlin Li, Xiaojun Jia, Yihao Zhang, Yang Liu, and Meng Sun. Position: Agent-specific trustworthiness risk as a research priority. In *ICML 2025 Workshop on Reliable and Responsible Foundation Models*.

Zeming Wei, Yifei Wang, Ang Li, Yichuan Mo, and Yisen Wang. Jailbreak and guard aligned language models with only few in-context demonstrations. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2026.

Chengcan Wu, Zhixin Zhang, Mingqian Xu, Zeming Wei, and Meng Sun. Monitoring llm-based multi-agent systems against corruptions via node evaluation. *arXiv preprint arXiv:2510.19420*, 2025.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.

Xijian Xu and Jun Wu. Mitigating llm hallucination snowballing in multiagent systems via context-aware semantic consistency reasoning. *IEEE Transactions on Neural Networks and Learning Systems*, 2026.

Bingyu Yan, Zhibo Zhou, Litian Zhang, Lian Zhang, Ziyi Zhou, Dezhuang Miao, Zhoujun Li, Chaozhuo Li, and Xiaoming Zhang. Beyond self-talk: A communication-centric survey of llm-based multi-agent systems. *arXiv preprint arXiv:2502.14321*, 2025a.

Bingyu Yan, Ziyi Zhou, Xiaoming Zhang, Chaozhuo Li, Ruilin Zeng, Yirui Qi, Tianbo Wang, and Litian Zhang. Attack the messages, not the agents: A multi-round adaptive stealthy tampering framework for llm-mas. *arXiv preprint arXiv:2508.03125*, 2025b.

Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, et al. R-judge: Benchmarking safety risk awareness for llm agents. *arXiv preprint arXiv:2401.10019*, 2024.

Yiming Zhang, Nicholas Carlini, and Daphne Ippolito. Effective prompt extraction from language models. *arXiv preprint arXiv:2307.06865*, 2023.

Zhexin Zhang, Leqi Lei, Lindong Wu, Rui Sun, Yongkang Huang, Chong Long, Xiao Liu, Xuanyu Lei, Jie Tang, and Minlie Huang. Safetybench: Evaluating the safety of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15537–15553, 2024.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. *arXiv preprint arXiv:2504.03160*, 2025.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. {PoisonedRAG}: Knowledge corruption attacks to {Retrieval-Augmented} generation of large language models. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 3827–3844, 2025.