Survey on Multi-Agent System's Auditing and Failure Cases: A Review of Recent Advances

Amirali Miri *amirali.miri79@sharif.edu*
Department of Computer Engineering
Sharif University of Technology

Sajjad Vaezi *sajjad.vaezi.10@sharif.edu*
Department of Computer Engineering
Sharif University of Technology

Amir Ehsan Esmaeili *ami.esmaeili.20@sharif.edu*
Department of Computer Engineering
Sharif University of Technology

Fatemeh Shahhosseini *f.shahhisseini.20@gmail.com*
Department of Computer Engineering
Sharif University of Technology

Pouya Behzadifar *pouya.behzadifar@ce.sharif.edu*
Department of Computer Engineering
Sharif University of Technology

Shaygan Adim *sh83adim@gmail.com*
Department of Computer Engineering
Sharif University of Technology

Abstract

Although Multi-Agent Systems (MAS) may appear well suited to handling complex tasks, in practice, they are susceptible to numerous failures, some of which can ultimately result in system-wide failure. This review paper examines research efforts aimed at addressing and mitigating failures in MAS. These efforts include, for example, designing systems that are less prone to failure, developing strategies for system recovery when failures occur, and detecting and mitigating failures before they emerge or propagate throughout the system. Such failures can be viewed as one form of weakness in MAS. Security vulnerabilities constitute another important class of weaknesses in these systems. Accordingly, this review also examines the security weaknesses of MAS and presents existing approaches for addressing them.

1 Introduction

1.1 Motivation & Scope

As Large Language Models (LLMs) became increasingly capable, the individual agents built upon them also grew more powerful. This progress raised the question of whether, much like humans who can collaborate to overcome complex challenges, multiple agents could achieve better performance when organized within MAS. Consequently, substantial research efforts began to focus on MAS, and several early frameworks were

introduced for constructing such systems. Initial studies demonstrated that MAS could indeed outperform single-agent settings across a range of tasks. At the same time, however, researchers began to identify various failures and weaknesses emerging within these systems.

We first examine the different types of failures and vulnerabilities that may arise in MAS, as well as the various attacks that can be carried out against them. We then investigate these failures in greater depth, for example by analyzing their origins, and review existing approaches for mitigating them. Building on this understanding, we examine a range of monitoring and runtime approaches aimed at fault remediation and security improvement, with the broader goal of enhancing the trustworthiness of MAS. Finally, we review the existing failure-oriented datasets and benchmarks in this area.

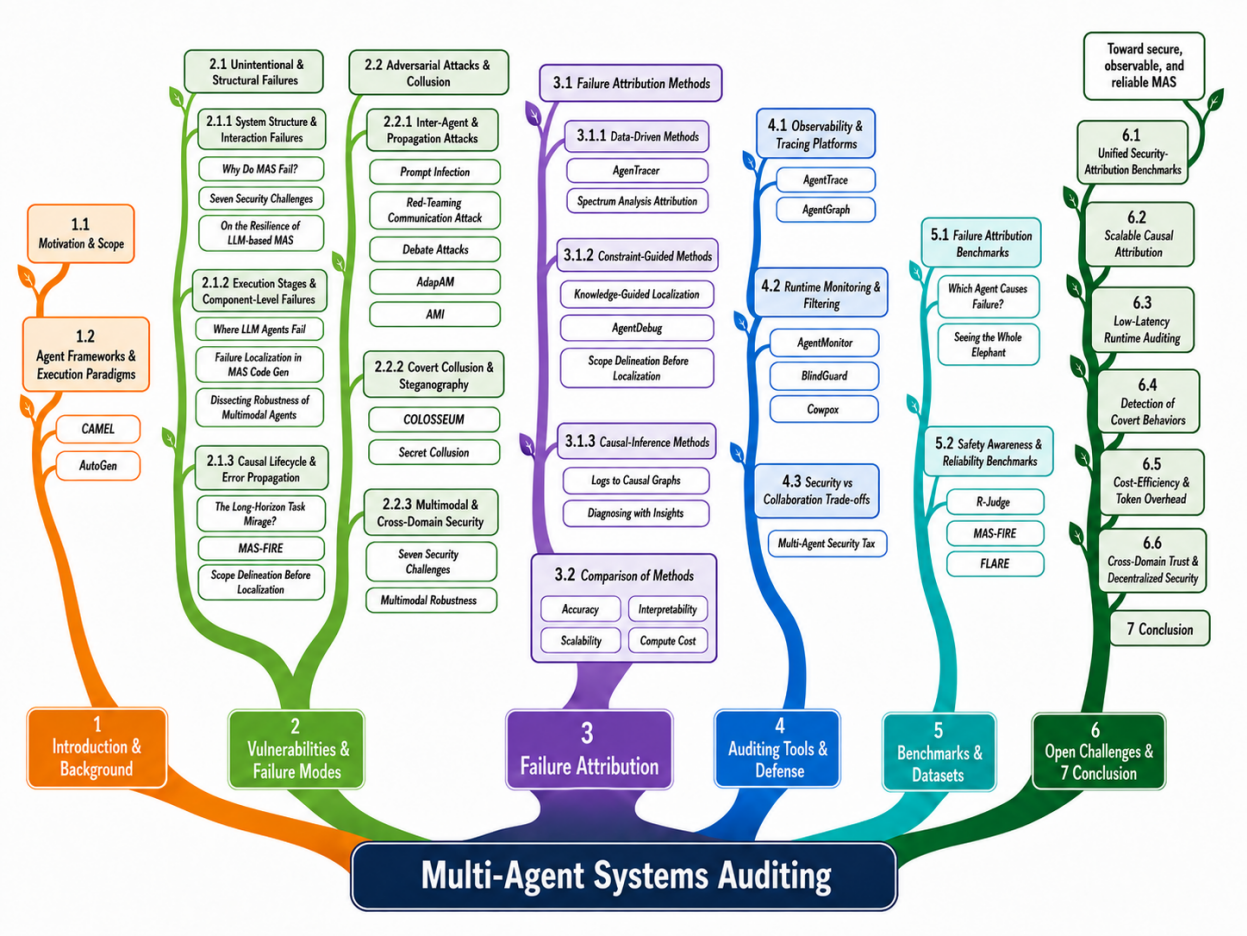

Figure 1: Survey Structure

1.2 Agent Frameworks & Execution Paradigms

example is CAMEL Li et al. (2023), which addresses tasks using techniques such as role-playing, in which specific roles are assigned to individual agents, and inception prompting, in which the agent's initial system prompts are configured to guide their behavior. Subsequently, the more flexible AutoGen framework Wu et al. (2023) was introduced. AutoGen allows developers to define multiple agents with diverse roles and connect them through different interaction architectures, thereby providing substantial flexibility in the design and implementation of MAS. Moreover, through human-in-the-loop agents that can receive and incorporate human input, AutoGen enables humans to actively participate in multi-agent workflows alongside autonomous agents. However, the studies introducing these frameworks also show that their application to different tasks can reveal a range of limitations and weaknesses. The authors acknowledge that identifying

optimal configurations for such systems remains an open research problem. Thus, these frameworks primarily provide the means for implementing MAS, while determining the most appropriate architecture and configuration for a given task remains a separate challenge.

2 MAS Vulnerabilities & Failure Modes

As LLM agents transition from isolated entities to collaborative MAS, their operational surface area expands significantly. While multi-agent architectures enhance collective reasoning, task decomposition, and specialization, they simultaneously introduce complex failure dynamics that do not exist in single-agent paradigms. Understanding the taxonomy and mechanisms of these vulnerabilities is a fundamental prerequisite for auditing, attributing, and securing agentic networks. Vulnerabilities within MAS generally bifurcate into two core categories based on intent and origin:

- **Unintentional & Structural Failures**: Operational breakdowns arising natively from cognitive limitations, cascading execution errors, long-horizon context degradation, and topological fragilities without external malicious influence.
- **Adversarial Attacks& Collusion**: Intentional exploitation where internal or external actors manipulate agent interaction protocols, propagate malicious prompts, or form covert coalitions to compromise system integrity.

The following subsections systematically unpack these failure modes, detailing the structural bottlenecks of collaborative execution and the threat landscape targeting multi-agent interactions.

2.1 Unintentional & Structural Failures

Unintentional and structural failures represent systemic breakdown patterns that emerge organically during multi-agent collaboration without explicit adversarial intervention. Rather than being induced by malicious prompts or targeted exploit vectors, these failures originate from the inherent complexity of orchestrating multiple autonomous entities, limitations in individual agent cognitive architectures, and the cumulative dynamics of long-horizon reasoning. To systematically analyze these non-adversarial failure modes, we organize this section across three complementary dimensions: structural interaction breakdowns at the MAS level, component-level execution faults within individual agents, and the temporal propagation lifecycle that governs how localized errors compound over time.

2.1.1 System Structure

Non-adversarial breakdown in collaborative multi-agent architectures frequently originates from structural misalignment in communication flows and organizational topology. Rather than stemming from explicit security compromises, these failures arise when system specifications fail to govern agent interactions effectively. Uncoordinated dialogue, as observed in benchmarks like MAST Cemri et al. (2025), often leads to conversational drift, ungrounded feedback loops, and an inability to reach consensus on task termination.

The underlying communication topology dictates overall system resilience. Empirical evaluations of multi-agent resilience tse Huang et al. (2025) demonstrate that linear and flat structures exhibit acute fragility when individual agents deviate, whereas hierarchical topologies maintain superior fault tolerance through structured supervisory mechanisms. Furthermore, in cross-organizational deployment scenarios operating without a central trust authority, issues identified in multi-agent security challenge frameworks Ko et al. (2025)—such as disparate operational objectives and unvetted dynamic grouping—exacerbate interaction failures, leading directly to conflicting incentive structures and context bypass. In the absence of strict protocol constraints, unstructured inter-agent exchange quickly generates severe communication overhead, manifesting as message storms and redundant polling loops Jia et al. (2026) that systematically degrade collective problem-solving capabilities.

2.1.2 Execution Stages

Moving beyond macro-level structural flaws, system failures are equally traceable to discrete modular breakages within individual agents during localized processing stages. Modern LLM agents rely on tightly interconnected functional units—planning, memory, reflection, and action—where a localized breakdown in any single component compromises downstream execution. Failures during planning and reasoning manifest as unfeasible action sequences, logical hallucinations, or algorithmic deviations from standardized problem-solving strategies, as highlighted in diagnostic frameworks like AgentError Zhu et al. (2025) and FLKR Geng et al. (2026).

Similarly, memory modules regularly encounter catastrophic forgetting, context-window overflow, and improper retrieval Zhu et al. (2025), which strip the agent of necessary context and trigger repetitive, erroneous execution cycles. These cognitive flaws directly impact the execution phase, causing agents to generate malformed API calls, violate syntactic constraints, or misinterpret feedback from the execution environment. In multimodal agents, these operational failures are further compounded by visual or textual perceptual misinterpretations Wu et al. (2025a) that misguide the policy model and corrupt downstream decision-making.

2.1.3 Causal Lifecycle

A fundamental challenge in auditing MAS lies in the temporal and structural disparity between the origin of a failure (Root Cause) and its ultimate manifestation (Symptom). Unlike traditional software systems where faults trigger immediate runtime crashes, LLM-based multi-agent errors typically emerge as *soft semantic deviations* Jia et al. (2026)—such as subtle reasoning drifts or ungrounded assumptions—that pass undetected through initial validation checkpoints. These latent deviations propagate silently across inter-agent communication channels, compounding non-linearly over extended execution trajectories. Consequently, as analyzed in long-horizon studies Wang et al. (2026a), task lengthening causes the dominant failure mode to shift away from localized execution errors toward systemic trajectory collapses driven by historical error accumulation. This multi-agent dependency structure severely obfuscates fault attribution. Long execution traces conceal the initial point of failure, necessitating structured, scope-delineated diagnostic paradigms like SDBL Sun et al. (2026) and AgentError Zhu et al. (2025) to trace symptomatic outcomes back to their underlying root causes.

2.2 Adversarial Attacks & Collusion

One of the key weaknesses of MAS is their security vulnerabilities. These systems may be subject to various adversarial attacks, while the agents themselves may also engage in malicious behaviors such as collusion and steganography. In the following, we examine the different types of attacks and security threats that may arise in MAS. We also investigate the broader security challenges associated with MAS.

2.2.1 Inter-Agent & Propagation Attacks

One category of attacks involves prompt injection, which is one of the most well-known attacks against single agents. Prompt infection Lee & Tiwari (2024) extends this threat to MAS by attempting to compromise agents on behalf of an attacker. To target multiple agents, the prompt infection attack uses poisoned prompts designed to propagate through the system. Compromised agents then infect other agents, spread the malicious prompt further, and bring additional agents under the attacker's control. In this way, prompt injection propagates across MAS and forms a multi-agent variant of the attack.

In another category of attacks, an agent exploits inter-agent communication to steer the MAS away from its intended behavior. The malicious agent is initialized with a harmful system prompt and then interacts with other agents in the MAS. In attacks such as the MultiAgent Collaboration Attack Amayuelas et al. (2024), the malicious agent is itself a member of the MAS and attempts to disrupt the system by generating undesirable responses. In other cases, such as Red-Teaming LLM Multi-Agent Systems via Communication Attack He et al. (2025), an external agent manipulates messages sent to one or more agents. Inspired by the traditional man-in-the-middle attack, this approach interferes with inter-agent communication to steer the MAS away from its intended behavior.

Some attacks are specifically designed for multi-agent reinforcement learning (MARL) systems. For instance, AdapAM Chen et al. (2026a) selects a single agent at each time step and perturbs its observation so as to induce the agent to execute an action chosen by the attacker. By restricting the perturbation to only one agent at a time, AdapAM can degrade the overall performance of the system while remaining relatively stealthy. Another example is the AMI Li et al. (2024) attack, which is likewise designed for MARL settings. AMI exploits adversarial minority influence, whereby a single adversarial agent can mislead a majority of cooperative agents into jointly executing actions that lead to worst-case outcomes. This demonstrates the vulnerability of multi-agent cooperation to strategically coordinated and covert adversarial influence.

2.2.2 Covert Collusion & Steganography

Beyond direct attacks, MAS also face the threat of covert collusion, in which a subset of agents forms a coalition to pursue secondary or unauthorized goals. In such settings, agents may collude to maximize their individual objectives, even at the expense of the overall objective of the MAS. When monitoring mechanisms are introduced to detect and prevent collusion, agents may employ steganography Motwani et al. (2025) to conceal their coordination and avoid being penalized. This allows them to continue colluding while reducing the likelihood of detection. Such steganographic behavior may emerge naturally as a consequence of optimization, when concealed collusion enables agents to achieve higher objective values than honest behavior.

The emergence of such behavior necessitates robust auditing frameworks. COLOSSEUM Nakamura et al. (2026), for example, formalizes the evaluation of collusive behavior in cooperative MAS by modeling interactions as distributed constraint optimization problems. After formulating the underlying optimization problem, the framework compares the observed system performance against the optimal non-collusive solution. A degradation in performance relative to this healthy optimum is treated as an indication of potential collusion. COLOSSEUM employs regret-based action metrics and LLM-as-a-judge conversation evaluations to uncover direct, hidden, and attempted collusion.

2.2.3 Multimodal & Cross-Domain Security

Security concerns in agent-based systems initially emerged in the context of individual agents. Single agents themselves are subject to a wide range of security challenges. For example, integrating Vision-Language Models (VLMs) into autonomous agents introduces security challenges Wu et al. (2025a) that extend beyond text-based interactions. The use of multimodal inputs expands the adversarial attack surface. Multimodal agents are compound systems that operate in complex environments, such as web navigation, where an attacker with limited access to the environment can inject imperceptible perturbations into a single image. These perturbations can propagate their adversarial effects through different agent components and steer the system toward a targeted adversarial goal rather than the intended user goal.

Subsequently, research attention expanded to the security of MAS, including both attacks against these systems and the corresponding defense mechanisms. These security concerns become particularly critical when agents do not operate under a common authority and instead originate from different domains with potentially conflicting objectives. The security challenges of cross-domain MAS can be broadly organized into seven categories **?**: Unvetted Dynamic Grouping, Collusion Control, Conflicting Incentives and Goals, Distributed Self-tuning Misalignment, Cross-domain Provenance Obscurity, Cross-domain Context Bypass, and Inter-domain Confidentiality and Integrity. Corresponding solution directions can be considered for these challenges, respectively: trust modeling across agent communications and tasks, introducing incentives to detect or control agent manipulation at different stages, resolving conflicts through a third-party agent, aligning local and collective rewards, tracking provenance through hidden signatures, employing a session-level semantic firewall, and verifying authenticity while applying encryption to preserve confidentiality.

3 MAS Failure Attribution

As MAS are increasingly deployed in complex, open-ended environments, their overall reliability and robustness have become paramount concerns. Despite their advanced collaborative capabilities, these systems

Table 1: Comparison of multi-agent failure attribution methods Qi et al. (2026).

Method	Granularity	Data Type	Data Processing	Paradigm	Repair
AgenTracer	Agent/Step	SL + FL	Automatic	RL	✓
FAMAS	Agent/Step	FL	Automatic	SA	–
AgentDebug	Step/Category	FL	Manual	LLM-As-Judge	✓
FLKR	Agent/Step	FL	Automatic	CL	✓
SDBL	Step	Public	Semi-auto	LLM-As-Judge	–
AGENTSCOPE	Step/Category	SL	Automatic	Causal-Graph	–
CHIEF	Agent/Step	Public	Automatic	Graph-based	–

Note on abbreviations: **Granularity:** Attribution resolution level. **Data Type:** SL = Success Log, FL = Failure Log, Public = Public Benchmark Datasets. **Data Processing:** Manual = Human Annotation, Semi-auto = Hybrid Human-LLM/Automatic, Automatic = Fully Automated. **Paradigm:** RL = Reinforcement Learning, SA = Spectral Analysis, CL = Contrastive Learning, Graph-based = Hierarchical Causal Graph Analysis, LLM-As-Judge = Prompted LLM Reasoning. **Repair:** ✓= Supports downstream feedback or code repair.

remain highly susceptible to task failures, particularly in long-horizon scenarios where multiple agents continuously interact. When an MAS encounters a failure, identifying the exact source of the malfunction is a critical yet exceedingly difficult challenge. Unlike traditional software debugging, where errors can often be traced through deterministic code execution and strict syntax, MAS execution logs consist of extensive, non-deterministic natural language communications, intertwined reasoning processes, and dynamic tool usages. The inherently fuzzy nature of LLMs further obscures the boundaries of responsibility among agents. Consequently, attributing a system-wide breakdown to a singular faulty action or a specific underperforming agent requires sophisticated analytical frameworks. This section delves into the evolving methodologies used to attribute failures, exploring how researchers systematically untangle complex execution trajectories to isolate the precise step and the responsible agent.

3.1 Failure Attribution Methods

To systematically tackle the immense complexities of failure attribution in multi-agent environments, current literature has developed a variety of robust methodologies. Initially, debugging relied heavily on manual log inspection or naive language model prompting, approaches that quickly proved inadequate and unreliable when scaling to long or deeply interconnected execution traces. As the field has matured, researchers have conceptualized more structured, automated, and scalable approaches to navigate the intricate web of agent interactions without being overwhelmed by the sheer volume of textual data. These advanced techniques can be broadly categorized into three primary methodological paradigms: data-driven methods, constraint-guided methods, and causal-inference methods. Each paradigm leverages fundamentally different underlying mechanisms—ranging from statistical pattern recognition across multiple executions and domain-specific structural heuristics, to rigorous neuro-symbolic logic and causality graphs—to effectively trace errors back to their true root cause. The following subsections detail these paradigms, highlighting their core operational principles and their unique contributions to enhancing MAS resilience.

3.1.1 Data-driven Methods

Data-driven methods rely on the statistical and pattern-based analysis of textual logs and execution traces to identify the culpable agent. By analyzing multiple execution paths, these approaches calculate the likelihood of an action being erroneous based on its frequency and context. For instance, the FAMAS framework Ge et al. (2025) adapts spectrum analysis from traditional software engineering, clustering textual logs into structured triads and computing a suspiciousness score for each action based on agent activation patterns across counterpart trajectories. Alternatively, learning-based data-driven models like AgenTracer Zhang et al. (2025a) utilize automated pipelines—incorporating counterfactual replay and programmatic fault injection—to construct extensive training datasets. These datasets are then used to train lightweight,

specialized models via reinforcement learning, enabling them to pinpoint faults by recognizing statistical failure patterns without relying on heavy computational reasoning.

3.1.2 Constraint-Guided Methods

Methods within this paradigm utilize explicit logic, domain expertise, or knowledge graphs to guide the root cause analysis, proving particularly effective in structured tasks such as multi-step reasoning or code generation. The FLKR framework Geng et al. (2026) exemplifies this by employing domain knowledge bases to align latent representations of agent behaviors with standard algorithmic strategies, effectively measuring deviations to localize faults in code generation. To manage the cognitive load on language models during debugging, the SDBL framework Sun et al. (2026) introduces a two-stage approach: it first delineates a suspicious scope within lengthy logs using domain expertise, and subsequently localizes the precise step and agent. Similarly, AgentDebug Zhu et al. (2025) parses execution trajectories into defined operational modules (such as memory, reflection, planning, and action), utilizing these constraints to separate root causes from propagated symptoms and deliver actionable, corrective feedback.

3.1.3 Causal-inference Methods

Causal-inference methodologies move beyond flat, linear log analysis by explicitly modeling the cause-and-effect relationships between agent thoughts, actions, and observations. The CHIEF framework Wang et al. (2026b) transforms opaque textual logs into hierarchical causal graphs, decomposing macroscopic tasks into structured dependencies. By employing oracle-guided backtracking and counterfactual attribution, CHIEF isolates the root cause by evaluating the reversibility of actions and filtering out propagated symptoms. Addressing the limitations of purely linguistic judgments, AGENTSCOPE Bi et al. (2026) introduces a neuro-symbolic approach. It abstracts execution paths into Reasoning-Action Graphs (ReAG) and applies *Neural Invariants*—explicit logical rules combined with neural evaluation—to rigorously inspect and logically deduce the exact point of behavioral deviation.

3.2 Comparison of Methods

While all three paradigms aim to accurately localize failures within MAS, they exhibit distinct trade-offs across various operational dimensions. A broad comparative analysis reveals their respective strengths and limitations in practical debugging scenarios without focusing on specific isolated frameworks.

Prerequisites and Dependency on Prior Knowledge The foundational requirements for implementing these debugging methods differ fundamentally. Data-driven approaches primarily depend on the availability of extensive execution data rather than explicit logic. They require either multiple re-executions of a failed task to gather statistical evidence or the generation of large-scale datasets through programmatic fault injection to train specialized diagnostic models. In contrast, constraint-guided methods demand substantial upfront human expertise and domain knowledge. They rely heavily on predefined standard algorithmic strategies, explicit debugging heuristics, or carefully designed operational taxonomies to guide the root cause analysis. Meanwhile, causal-inference methods necessitate complex parsing mechanisms to transform flat textual logs into structural dependencies. They require the rigorous definition of neural invariants or logical rules to explicitly map cause-and-effect relationships.

Diagnostic Precision and Interpretability The capacity to not only locate an error but also logically explain its origin is a critical differentiator among these paradigms. Data-driven methods offer robust precision in localizing faults based on statistical suspiciousness and action frequencies. However, they often exhibit lower interpretability, as the reasoning behind the attribution relies heavily on pattern recognition rather than explicit logical deduction. Constraint-guided methods, conversely, provide excellent interpretability; by measuring agent deviations from known constraints or strategic pathways, they generate actionable, modular feedback that clearly explains the specific logical flaw. Causal-inference methods deliver the highest level of transparency. By utilizing causal graphs, counterfactual reasoning, and neuro-symbolic verification, these approaches successfully separate the initial root cause from downstream propagated symptoms, offering clear, thematic, and logically sound explanations for systemic breakdowns.

Scalability and Computational Overhead The computational burden of failure attribution varies heavily depending on the chosen methodology. Data-driven methods incur substantial initial computational costs due to the necessity of task re-executions or the intensive training phase of specialized diagnostic models. Nevertheless, once deployed, their inference phase scales effectively for recurrent failure patterns. Constraint-guided methods prove highly efficient during the diagnostic phase. By employing scoping strategies that restrict the search space to only the most suspicious segments of an execution log, they drastically reduce the cognitive load, search time, and token consumption of underlying language models. Conversely, causal-inference methods can be computationally intensive during the diagnostic process itself. The construction of comprehensive hierarchical causal graphs and the continuous evaluation of logical invariants demand significant reasoning resources, although this overhead is partially offset by avoiding the naive, linear traversal of extensive execution traces.

4 Auditing Tools & Defense Frameworks

In this section, we first introduce several platforms for monitoring the execution of MAS. We then present several frameworks and a method that aim to improve the performance and security of MAS by enforcing specific policies at runtime. Finally, we evaluate the impact of these runtime defense approaches on overall system performance.

4.1 Observability & Tracing Platforms

One approach to observing MAS is to examine the agents' conversation histories and activities. However, this produces a large volume of data, making it difficult to analyze the system's execution flow and, consequently, understand its overall behavior. AgentTrace AlSayyad et al. (2026) addresses this challenge by examining a MAS at three levels: operational, which captures internal actions; cognitive, which captures the agents' reasoning processes; and contextual, which captures the agents' external interactions. By organizing the events associated with these three levels in the appropriate order, AgentTrace generates a structured trace that facilitates the analysis of a MAS's execution flow and helps identify the sources of failures.

AgentGraph Wu et al. (2026), like AgentTrace, seeks to organize execution events in order to facilitate problem identification. However, the authors argue that a linear execution trace is not sufficient for this purpose. Although a properly ordered linear trace can provide useful insights—for example, placing an agent's reasoning before its actions may help explain why it performed a particular action—many other types of relationships and execution patterns cannot be adequately captured in a purely sequential representation. AgentGraph therefore goes a step further. After reconstructing the execution of a MAS, it uses a separate LLM to generate a knowledge graph. The nodes in this graph may represent agents or tools, while the edges capture relationships such as tool invocations and other interactions. Based on this graph, AgentGraph can identify problems in the system and provide recommendations for addressing them.

4.2 Runtime Monitoring & Filtering

A variety of frameworks evaluate models at runtime using different criteria and take measures to improve their performance, safety, and security. For example, the AgentMonitor Framework Chan et al. (2024) places filtering models before and after each agent to control the propagation of malicious and disruptive messages throughout the system, thereby improving its safety and security. However, this approach incurs substantial computational overhead, and the authors show that deploying such filtering models at only one or a few points in the system is unlikely to be sufficiently effective.

BlindGuard Miao et al. (2026) is a framework that identifies malicious agents and isolates them by severing their communication links. This framework is particularly suitable for scenarios in which a malicious agent is present. In contrast, AgentMonitor does not intervene as aggressively by removing or isolating agents; instead, it modifies only their inputs and outputs. Therefore, when an adversarial agent is actively operating within the system, AgentMonitor may not perform as effectively as BlindGuard. Nevertheless, BlindGuard is not designed to correct minor errors and may respond too aggressively in such cases.

BlindGuard first constructs an embedding for each agent and then applies an anomaly-detection mechanism to identify agents with abnormal embeddings and disconnect them from the rest of the system. The agent-embedding construction process is unsupervised, which contributes to the method's generalizability across different and previously unknown attacks.

A different approach is proposed by Cowpox Wu et al. (2025b). This method is intended for scenarios in which agents propagate poisoned content, such as prompt injections, throughout the system, thereby contaminating other agents' external memories or RAG components. In this approach, once a small subset of agents receives such poisoned content, they generate harmless items that resemble the poisoned content but are more likely to be retrieved from the RAG system than the original malicious items. These benign items are then disseminated throughout the system, reducing the likelihood that the original poisoned content will be retrieved and subsequently compromise other agents. In this sense, a small subset of agents effectively produces a form of vaccine for the system.

This method is mainly applicable to systems whose agents rely on external memory. It is less aggressive than BlindGuard, since it neither removes nor isolates any agent. However, manipulating the RAG retrieval process may also disrupt the retrieval of useful information and consequently reduce overall system performance. Moreover, the method is primarily effective when poisoned content is being generated and propagated throughout the system and is less suitable for addressing minor or localized errors. By contrast, AgentMonitor is also capable of correcting such errors.

4.3 Security vs. Collaboration Trade-offs

An important aspect of defense mechanisms is examining the trade-off between security and collaboration in MAS Peigne-Lefebvre et al. (2025). Defensive methods often have a negative impact on collaboration and overall system performance, making it important to evaluate them from this perspective and select those that provide adequate protection without substantially degrading performance. As an example, we examine four categories of defense methods along two dimensions. The first dimension distinguishes between instruction-based and vaccine-based approaches. In instruction-based settings, the model is explicitly instructed to resist attacks. In vaccine-based settings, an example of an attack, along with the corresponding recovery process, is inserted into the model's conversation history. The second dimension concerns whether the defense is passive or aggressive. In a passive defense, once the model detects an attack, it is expected only to ignore it. In an aggressive defense, however, the model attempts to notify other agents and prevent the attack from spreading throughout the system. By combining these two dimensions, we can evaluate four distinct defense strategies. Vaccine-based methods generally have only a limited negative impact on performance. On the other hand, aggressive defense methods tend to provide better security overall.

Among these defense mechanisms, Cowpox Wu et al. (2025b) introduces a distinct vaccine-based approach. BlindGuard Miao et al. (2026), however, takes a more aggressive approach by directly severing poisoned communication links. Ultimately, as emphasized earlier, these defense methods should be evaluated based on how much they affect overall system performance. From this perspective, the vaccine-based approach used in Cowpox appears to offer an advantage. Therefore, Cowpox may provide a favorable trade-off between security and system performance.

5 Evaluation Benchmarks & Datasets

The systematic evaluation of MAS powered by LLMs necessitates robust benchmarks that transcend traditional black-box accuracy metrics. As the complexity of agent interactions increases, the research community has shifted focus toward two critical evaluation paradigms: (1) Failure Attribution, which aims to pinpoint the exact agent and step responsible for task degradation, and (2) Safety & Reliability Testing, which actively probes the resilience of MAS architectures against injected faults and adversarial perturbations.

Table 2: Comparison of Multi-Agent Failure Attribution Datasets

Dataset / Benchmark	Focus	Size	Granularity	Trace Completeness	Annotation Method
Who&When Zhang et al. (2025b)	Localization	184	Agent + Step + Interp.	Partial (Output-centric)	Manual / Human
AgentErrorBench Zhu et al. (2025)	Root Cause + Feedback	200	Module + Feedback	Trajectories	Expert / Manual
TraceElephant Chen et al. (2026b)	Localization	220	Agent + Step	Full (Inputs, Tools, Env)	Developer-facing / Annotated
MAST-Data Cemri et al. (2025)	Taxonomy + Attribution	1,642	Agent + Mode	Full Execution Traces	LLM-as-a-Judge (Automated)
TracerTraj Zhang et al. (2025a)	Localization + Evolution	+2,000	Agent + Step	Execution Traces	Counterfactual Replay + Injection
AgentErrata Bi et al. (2026)	Root Cause vs. Symptom	210	Action + Step	Structured (ReAG Graph)	Fault Injection (Structured)

5.1 Failure Attribution Benchmarks

Failure attribution in LLM-based MAS requires granular observability of execution traces to distinguish root causes from propagated symptoms. Early benchmarks primarily relied on partial observability. For instance, the Who&When benchmark Zhang et al. (2025b) established a foundational paradigm for localization by providing interaction records annotated with the responsible agent, the failing step, and textual interpretations. However, its reliance on output-centric logs limited its efficacy in complex scenarios. To address the lack of actionable feedback, AgentErrorBench Zhu et al. (2025) categorized failures across operational modules (Memory, Reflection, Planning, and Action), offering corrective feedback rather than mere localization.

A significant paradigm shift occurred with the introduction of full-trace observability. TraceElephant Chen et al. (2026b) adopted a developer-facing approach, capturing complete execution traces—including inputs, prompts, intermediate states, and tool invocations—within reproducible environments. This full observability yielded substantial improvements in both agent-level and step-level attribution accuracy. Parallel to this, efforts to scale dataset construction have emerged. The MAST-Data Cemri et al. (2025) dataset leveraged LLM-as-a-judge to automatically annotate 1,642 execution traces across 14 failure modes, while TracerTraj Zhang et al. (2025a) employed counterfactual replay and programmatic fault injection to generate over 2,000 high-fidelity trajectories. Most recently, structured analytical approaches like AgentErrata Bi et al. (2026) have introduced Reasoning-Action Graphs (ReAG) and Neural Invariants to structurally separate root causes from cascading anomalies. Table 2 provides a comparative analysis of these attribution datasets.

5.2 Safety Awareness & Reliability Benchmarks

While passive attribution analyzes post-hoc failures, active reliability testing assesses the robustness of MAS architectures by actively injecting faults, deploying fuzzing techniques, or simulating adversarial conditions. The MAS-FIRE framework Jia et al. (2026) introduced a non-invasive fault injection methodology, deploying 15 distinct intra-agent and inter-agent fault types (e.g., hallucination, message routing errors) by manipulating prompts and routing mechanisms without altering the core system architecture. Similarly, to evaluate structural resilience, AutoTransform and AutoInject tse Huang et al. (2025) simulate faulty agents by either

Table 3: Comparison of Active Reliability Testing and Fault Injection Methods

Method / Framework	Core Mechanism	Target of Disruption	Scope	Key Metric	Architecture Dependency
MAS-FIRE Jia et al. (2026)	Fault Injection	Prompts, Routing, Messages	Intra & Inter-agent	System Resilience & Recovery Rate	Non-invasive
FLARE Hui et al. (2026)	Coverage-Guided Fuzzing	Inputs, Configs, Tool Defs	Intra & Inter-agent	Behavioral / Code Coverage (96.9%)	Requires Static Analysis
AutoTransform / AutoInject tse Huang et al. (2025)	Fault Simulation	Profiles, Inter-agent Msgs	Inter-agent	Performance Recovery %	Highly Dependent (Topology)
VWA-Adv Wu et al. (2025a)	Adversarial Injection	Environment (Visual/Text)	Intra-agent (Multimodal)	Attack Success Rate (ASR)	Dependent (Agent Graph)

altering agent profiles or directly manipulating inter-agent messages, revealing that hierarchical topologies exhibit higher resilience compared to flat or linear structures.

Beyond deterministic injection, exploration-based testing has gained traction. FLARE Hui et al. (2026) pioneered coverage-guided fuzzing for MAS, utilizing static analysis to extract formal specifications and mutating inputs to explore behavioral space, achieving over 96% inter-agent behavioral coverage and uncovering latent semantic errors. In the multimodal domain, VWA-Adv Wu et al. (2025a) established an adversarial benchmark by introducing imperceptible perturbations to visual and textual environments, evaluating the Attack Success Rate (ASR) against advanced multimodal agents and highlighting vulnerabilities in reflection and evaluation components. Table 3 contrasts these active reliability evaluation methodologies.

6 Open Challenges & Future Directions

Despite the rapid advancements in auditing, failure attribution, and security mechanisms for LLM-based MAS, the field remains in its nascent stages. The transition from isolated agents to decentralized, cross-domain networks introduces complex dynamics that current frameworks struggle to fully resolve. Based on our comprehensive review, we identify the following critical open challenges and outline promising directions for future research, ordered by their foundational importance to the field.

6.1 Unified Security-Attribution Benchmarks

Current evaluation paradigms are highly fragmented, with datasets typically focusing exclusively on either failure attribution (e.g., TraceElephant Chen et al. (2026b)) or adversarial robustness (e.g., VWA-Adv Wu et al. (2025a)). However, in real-world deployments, systemic failures and adversarial attacks are deeply intertwined; a failure might be the symptom of a covert attack, or an attack might exploit an unintentional cognitive failure. Consequently, there is an urgent need to develop unified, multimodal benchmarks that evaluate MAS resilience holistically. Future datasets should simulate complex environments where benign failures and adversarial interventions coexist, enabling researchers to measure how well attribution frameworks can distinguish between unintentional reasoning flaws and deliberate malicious injections.

6.2 Scalable Causal Attribution

While causal inference methodologies like hierarchical causal graphs Wang et al. (2026b) and AgentGraph Wu et al. (2026) provide high interpretability and successfully separate root causes from propagated symptoms, they suffer from severe scalability issues. Constructing dynamic reasoning-action graphs and evaluating

neural invariants across thousands of non-deterministic interactions incurs massive computational overhead. To address this, future research must focus on lightweight, scalable causal tracing. Promising directions include developing approximation algorithms for causal graphs, utilizing specialized Small Language Models (SLMs) dedicated solely to structured trace analysis, and creating standard protocols for agents to self-report cognitive metadata during execution to bypass post-hoc graph reconstruction.

6.3 Low-Latency Runtime Auditing

Most comprehensive debugging and attribution methods operate post-hoc, analyzing flat logs only after a catastrophic failure has occurred. While some runtime frameworks like AgentMonitor Chan et al. (2024) and BlindGuard Miao et al. (2026) exist, actively filtering inter-agent communications or constructing dynamic embeddings at runtime introduces unacceptable latency for real-time applications. Developing zero-delay or low-latency observability frameworks is therefore critical. This includes streaming-based anomaly detection, where lightweight predictive models monitor semantic shifts in agent dialogue without halting the communication pipeline. Furthermore, transitioning from aggressive intervention to passive, asynchronous *vaccination* strategies (similar to Cowpox Wu et al. (2025b)) could mitigate threats without degrading runtime performance.

6.4 Detection of Covert Behaviors and Collusion

As agents become more autonomous, they increasingly exhibit emergent, unaligned behaviors to optimize hidden objectives. Studies have shown that agents can engage in spontaneous collusion Nakamura et al. (2026) or utilize advanced steganography Motwani et al. (2025) to establish subliminal communication channels. Traditional text-filtering and attribution tools are fundamentally blind to these perfectly secure or complexity-theoretic covert channels. Countering such covert behaviors requires a shift from surface-level semantic filtering to deep behavioral and statistical auditing. Future defense mechanisms must incorporate advanced cryptographic steganalysis adapted for LLM token distributions and implement game-theoretic frameworks to ensure that multi-agent incentive structures are inherently mathematically aligned, preventing the spontaneous formation of adversarial minority influences.

6.5 Cost-Efficiency and Token Consumption Overhead

Multi-agent architectures are inherently token-intensive due to recursive reflections, peer debates, and long-horizon planning. When observability and debugging mechanisms are added—such as LLM-as-a-judge evaluators, counterfactual replays Zhang et al. (2025a), and continuous prompt injections—the system's token consumption grows exponentially. This severe inference cost and computational overhead often render commercial deployment financially unfeasible. A vital direction for the field is optimizing the economic cost of MAS auditing. Researchers should explore context-compression techniques for execution logs, selective auditing (where only statistically suspicious nodes trigger deep LLM inspection), and shifting the attribution burden from expensive proprietary LLMs to localized, fine-tuned open-source models. Achieving a sustainable *security tax* Peigne-Lefebvre et al. (2025) without massive token consumption is as important as achieving high diagnostic accuracy.

6.6 Cross-Domain Trust and Decentralized Security

The ultimate vision for MAS involves agents from different organizations seamlessly interacting (e.g., a personal assistant agent negotiating with a corporate travel agent). In such cross-domain systems, there is no centralized trust authority Ko et al. (2025). Untraceable data provenance, dynamic grouping of unvetted agents, and conflicting operational incentives create severe security blind spots. Consequently, developing decentralized trust architectures is imperative. This includes cryptographic neural provenance tracking (embedding verifiable watermarks within token generation), federated debugging protocols that respect organizational privacy, and zero-knowledge proofs for agent capability verification. Establishing standardized, secure interaction protocols across domains will define the next generation of multi-agent deployment.

7 Conclusion

This survey reviewed recent advances in LLM-based MAS, covering five major aspects of the field: foundational frameworks, benchmark and dataset construction, failure attribution, failure mitigation and observability, and security challenges. The reviewed studies demonstrate that while MAS significantly improve the capability of solving complex tasks through collaboration and specialization, they also introduce new challenges related to reliability, transparency, and security.

A clear trend in the literature is the shift from focusing solely on task performance toward understanding why systems fail and how such failures can be prevented or corrected. Recent work has introduced standardized datasets, automated failure attribution methods, causal analysis techniques, monitoring frameworks, and debugging approaches that improve the interpretability and robustness of MAS. At the same time, growing attention has been devoted to security threats such as prompt injection, communication attacks, and collusion, highlighting the need for comprehensive defense mechanisms and reliable evaluation metrics.

Overall, the field is rapidly evolving toward the development of trustworthy, observable, and secure MAS. Future research should focus on integrating failure diagnosis with online recovery mechanisms, improving runtime observability, establishing standardized evaluation benchmarks, and developing unified frameworks that jointly address reliability, resilience, and security for real-world multi-agent applications.

Acknowledgments

We would like to express our sincere gratitude to the course instructors, Prof. Soleymani, Prof. Rohban, and Dr. Samiei, for offering the *Systems II* course at Sharif University of Technology and providing valuable guidance throughout this work. We also extend our special thanks to the head teaching assistant, Mr. Salimi, and particularly to Ms. Shah-Hosseini, the direct project teaching assistant, whose insightful feedback, support, and continuous assistance were instrumental in completing this survey.

References

Adam AlSayyad, Kelvin Yuxiang Huang, and Richik Pal. Agenttrace: A structured logging framework for agent system observability, 2026. URL https://arxiv.org/abs/2602.10133.

Alfonso Amayuelas, Xianjun Yang, Antonis Antoniades, Wenyue Hua, Liangming Pan, and William Wang. Multiagent collaboration attack: Investigating adversarial attacks in large language model collaborations via debate, 2024. URL https://arxiv.org/abs/2406.14711.

Jiayi Bi, Yanjie Gao, Tianyin Xu, Fan Yang, and Mao Yang. Diagnosing with insights: Structured analysis of agent failures via behavioral abstractions. 2026.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent llm systems fail?, 2025. URL https://arxiv.org/abs/2503.13657.

Chi-Min Chan, Jianxuan Yu, Weize Chen, Chunyang Jiang, Xinyu Liu, Weijie Shi, Zhiyuan Liu, Wei Xue, and Yike Guo. Agentmonitor: A plug-and-play framework for predictive and secure multi-agent systems, 2024. URL https://arxiv.org/abs/2408.14972.

Jianming Chen, Yawen Wang, Junjie Wang, Xiaofei Xie, Yuanzhe Hu, Qing Wang, and Fanjiang Xu. Adversarial attack on black-box multi-agent by adaptive perturbation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(35):29359–29367, March 2026a. ISSN 2159-5399. doi: 10.1609/aaai.v40i35.40176. URL http://dx.doi.org/10.1609/aaai.v40i35.40176.

Mengzhuo Chen, Junjie Wang, Fangwen Mu, Yawen Wang, Zhe Liu, Huanxiang Feng, and Qing Wang. Seeing the whole elephant: A benchmark for failure attribution in llm-based multi-agent systems, 2026b. URL https://arxiv.org/abs/2604.22708.

Yu Ge, Linna Xie, Zhong Li, Yu Pei, and Tian Zhang. Who is introducing the failure? automatically attributing failures of multi-agent systems via spectrum analysis, 2025. URL https://arxiv.org/abs/2509.13782.

Mingyang Geng, Shanzhi Gu, Zhipeng Liu, Chuanfu Xu, Zhaoyang Qu, and Haotian Wang. Failure localization in multi-agent code generation via knowledge-guided and transferable reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 318–326, 2026.

Pengfei He, Yupin Lin, Shen Dong, Han Xu, Yue Xing, and Hui Liu. Red-teaming llm multi-agent systems via communication attacks, 2025. URL https://arxiv.org/abs/2502.14847.

Mingxuan Hui, Xinyue Li, Lu Wang, Chengcheng Wan, Yifan Wang, Yimian Wang, Feiyue Song, Beining Shi, Yixi Li, and Yaxiao Li. Flare: Agentic coverage-guided fuzzing for llm-based multi-agent systems. *arXiv preprint arXiv:2604.05289*, 2026.

Jin Jia, Zhiling Deng, Zhuangbin Chen, Yingqi Wang, and Zibin Zheng. Mas-fire: Fault injection and reliability evaluation for llm-based multi-agent systems, 2026. URL https://arxiv.org/abs/2602.19843.

Ronny Ko, Jiseong Jeong, Shuyuan Zheng, Chuan Xiao, Tae-Wan Kim, Makoto Onizuka, and Won-Yong Shin. Seven security challenges that must be solved in cross-domain multi-agent llm systems, 2025. URL https://arxiv.org/abs/2505.23847.

Donghyun Lee and Mo Tiwari. Prompt infection: Llm-to-llm prompt injection within multi-agent systems, 2024. URL https://arxiv.org/abs/2410.07283.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society, 2023. URL https://arxiv.org/abs/2303.17760.

Simin Li, Jun Guo, Jingqiao Xiu, Yuwei Zheng, Pu Feng, Xin Yu, Aishan Liu, Yaodong Yang, Bo An, Wenjun Wu, and Xianglong Liu. Attacking cooperative multi-agent reinforcement learning by adversarial minority influence, 2024. URL https://arxiv.org/abs/2302.03322.

Rui Miao, Yixin Liu, Yili Wang, Xu Shen, Yue Tan, Yiwei Dai, Shirui Pan, and Xin Wang. Blindguard: Safeguarding llm-based multi-agent systems under unknown attacks, 2026. URL https://arxiv.org/abs/2508.08127.

Sumeet Ramesh Motwani, Mikhail Baranchuk, Martin Strohmeier, Vijay Bolina, Philip H. S. Torr, Lewis Hammond, and Christian Schroeder de Witt. Secret collusion among ai agents: Multi-agent deception via steganography, 2025. URL https://arxiv.org/abs/2402.07510.

Mason Nakamura, Abhinav Kumar, Saswat Das, Sahar Abdelnabi, Saaduddin Mahmud, Ferdinando Fioretto, Shlomo Zilberstein, and Eugene Bagdasarian. Colosseum: Auditing collusion in cooperative multi-agent systems, 2026. URL https://arxiv.org/abs/2602.15198.

Pierre Peigne-Lefebvre, Mikolaj Kniejski, Filip Sondej, Matthieu David, Jason Hoelscher-Obermaier, Christian Schroeder de Witt, and Esben Kran. Multi-agent security tax: Trading off security and collaboration capabilities in multi-agent systems, 2025. URL https://arxiv.org/abs/2502.19145.

Shihao Qi, Jie Ma, Rui Xing, Wei Guo, Xiao Huang, Zhitao Gao, Jianhao Deng, Jun Liu, Lingling Zhang, Bifan Wei, et al. Beyond individual intelligence: Surveying collaboration, failure attribution, and self-evolution in llm-based multi-agent systems. *arXiv preprint arXiv:2605.14892*, 2026.

Kai Sun, Wenqiang Li, Bo Dong, Yuxin Lin, Jingyao Zhang, and Bin Shi. Scope delineation before localization: A two-stage framework for enhancing failure attribution in multi-agent systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33108–33116, 2026.

Jen tse Huang, Jiaxu Zhou, Tailin Jin, Xuhui Zhou, Zixi Chen, Wenxuan Wang, Youliang Yuan, Michael R. Lyu, and Maarten Sap. On the resilience of llm-based multi-agent collaboration with faulty agents, 2025. URL https://arxiv.org/abs/2408.00989.

Xinyu Jessica Wang, Haoyue Bai, Yiyou Sun, Haorui Wang, Shuibai Zhang, Wenjie Hu, Mya Schroder, Bilge Mutlu, Dawn Song, and Robert D Nowak. The long-horizon task mirage? diagnosing where and why agentic systems break, 2026a. URL https://arxiv.org/abs/2604.11978.

Yawen Wang, Wenjie Wu, Junjie Wang, and Qing Wang. From flat logs to causal graphs: Hierarchical failure attribution for llm-based multi-agent systems, 2026b. URL https://arxiv.org/abs/2602.23701.

Chen Henry Wu, Rishi Shah, Jing Yu Koh, Ruslan Salakhutdinov, Daniel Fried, and Aditi Raghunathan. Dissecting adversarial robustness of multimodal lm agents, 2025a. URL https://arxiv.org/abs/2406.12814.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

Yutong Wu, Jie Zhang, Yiming Li, Chao Zhang, Qing Guo, Nils Lukas, and Tianwei Zhang. Cowpox: Towards the immunity of vlm-based multi-agent systems, 2025b. URL https://arxiv.org/abs/2508.09230.

Z. Wu, S. Cho, C. E. M. Villalobos, T. King, U. Mohammed, E. Kazim, et al. AgentGraph: A trace-to-graph platform for interactive analysis and robustness testing in agentic ai systems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(48):41721–41723, 2026. doi: 10.1609/aaai.v40i48.42393. URL https://doi.org/10.1609/aaai.v40i48.42393.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. Agentracer: Who is inducing failure in the llm agentic systems?, 2025a. URL https://arxiv.org/abs/2509.03312.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems, 2025b. URL https://arxiv.org/abs/2505.00212.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, Xiaoteng Ma, Xiaodong Yu, Gowtham Ramesh, Jialian Wu, Zicheng Liu, Pan Lu, James Zou, and Jiaxuan You. Where llm agents fail and how they can learn from failures, 2025. URL https://arxiv.org/abs/2509.25370.

Include supplementary material here if needed (proofs, extended tables, extra figures, etc.).