MAStitch: Unifying Local and Global Perspectives for Anomaly Detection in Multi-Agent Systems

Lior Vaknin
Ben-Gurion University of the Negev
Beer Sheva, Israel
liorva@post.bgu.ac.il

Yarin Yerushalmi Levi
Ben-Gurion University of the Negev
Beer Sheva, Israel
yarinye@post.bgu.ac.il

Ron Solomon
Ben-Gurion University of the Negev
Beer Sheva, Israel
ronso@post.bgu.ac.il

Jaidip Kotak
Ben-Gurion University of the Negev
Beer Sheva, Israel
jaidip@post.bgu.ac.il

Amit Giloni
Fujitsu Research of Europe
Modiin, Israel
amit.giloni@fujitsu.com

Chiara Picardi
Fujitsu Research of Europe
York, UK
chiara.picardi@fujitsu.com

Roman Vainshtein
Fujitsu Research of Europe
Beer Sheva, Israel
roman.vainshtein@fujitsu.com

Yuval Elovici
Ben-Gurion University of the Negev
Beer Sheva, Israel
elovici@bgu.ac.il

Asaf Shabtai
Ben-Gurion University of the Negev
Beer Sheva, Israel
shabtaia@bgu.ac.il

ABSTRACT

Large language model (LLM)-based multi-agent systems (MASs) increasingly serve as decision-making and automation pipelines in diverse application domains. Although these systems are highly relied upon in many settings, they have been shown to be vulnerable to various threats, including threats exploited by malicious actors (e.g., adversarial attacks) and threats stemming from dangerous behavior of the system (e.g., insecure code generation). Those threats can be viewed as deviations from normal behavior (anomalies), requiring a robust anomaly detection solution. Most approaches for detecting such anomalies concentrate on individual agents, specific failure scenarios, or particular use cases, and often require additional training phases, limiting their applicability and ease of adoption across diverse MASs. To address these limitations, we introduce MAStitch, a platform- and threat-agnostic method for threat detection in MASs that does not require any training, making it well-suited for diverse environments and scenarios requiring plug-and-play detection capabilities. MAStitch leverages local and global perspectives, enabling comprehensive analysis of inter-agent interactions and intra-agent processes to detect anomalies. These perspectives are acquired by a pair of LLM-based agents: a Local Analyzer Agent (LAA), which evaluates each agent's execution process, and a Global Analyzer Agent (GAA), which aggregates evidence from multiple agents to detect cross-agent failures. The GAA summarizes the anomalous patterns identified, classifies the entire execution log, and provides an explanation when threats are detected. We evaluate our method's detection performance on six different MAS applications which were developed on two widely used MAS platforms. The evaluation results demonstrate MAStitch's ability to detect a variety of threats, with an average F1 score of 0.9 and a minimal false positive rate of 0.078.

KEYWORDS

Multi-Agent Systems; LLM-based Agents; Anomaly Detection; Adversarial Attacks; Security Analysis

ACM Reference Format:
Lior Vaknin, Yarin Yerushalmi Levi, Ron Solomon, Jaidip Kotak, Amit Giloni, Chiara Picardi, Roman Vainshtein, Yuval Elovici, and Asaf Shabtai. 2026. MAStitch: Unifying Local and Global Perspectives for Anomaly Detection in Multi-Agent Systems. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 9 pages. https://doi.org/10.65109/BMJZ4532

1 INTRODUCTION

Large language model (LLM)–based multi-agent systems (MASs) have demonstrated remarkable potential in solving complex problems, with frameworks like MetaGPT and ChatDev enabling sophisticated applications in diverse areas including software development and autonomous decision-making [15, 30]. The strength of such systems lies in their ability to decompose complex problems into simpler sub-tasks, each handled by a specialized LLM-based agent that collaboratively contributes toward achieving the overall system objective [12, 37]. However, the collaboration that drives their success also introduces a complex and expanded attack surface, exposing them to a variety of critical adversarial threats including indirect prompt injection (IPI), the use of insecure tools, and rapid propagation of misinformation across the agent network [11, 28]. Consequently, a localized, exploited threat targeting an individual agent can cascade through the system, leading to widespread, unpredictable failures.

Most existing threat detection approaches for MASs (1) concentrate on evaluating individual agents, (2) address specific failure scenarios, or (3) optimize for particular use cases, limiting their generalization and effectiveness across diverse MASs [18, 29]. Some solutions also require additional training phases, which involve

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/BMJZ4532

data collection, limiting their applicability and ease of adoption [33]. Thus, there is a critical need for security solutions that can holistically monitor both local agent actions and their global systemic impact, a capability that many existing post-hoc analysis or static defense mechanisms lack. To address this gap, we introduce MAStitch, the first platform- and threat-agnostic agentic anomaly detection method for MASs that stitches together fine-grained local agent insights with global system dynamic analysis, enabling comprehensive anomaly detection. MAStitch utilizes the system's configuration (e.g., agents' roles, tasks, and available tools) to validate execution logs, in order to define the expected system behavior. Those inputs are first processed by a Local Analyzer Agent (LAA), which detects deviations (abnormal/anomalous behavior) in each agent's executions, where the corresponding agent's configuration is used as the expected baseline behavior. Then, a Global Analyzer Agent (GAA) aggregates the intermediate finding of the LAA to produce a summary of the anomalous patterns identified, classify the entire execution log (anomalous or benign), and provide an explanation when threats are detected .

We evaluate our proposed method on various MAS applications [5, 13] developed on two popular MAS platforms (CrewAI [7] and LangGraph [22]), assessing its performance against a diverse set of MAS threats: direct prompt injection (DPI), IPI, memory poisoning (MP), and the generation of insecure code [41] [26]. Our evaluation demonstrates MAStitch's advantages over recently published MAS anomaly detection strategies; MAStitch achieved the highest overall accuracy of 90.02%, outperforming the second best approach by 15%, along with the lowest average false positive rate (FPR) among the approaches examined.

In summary, our contributions are as follows:

- To the best of our knowledge, we are the first to propose a novel agentic platform- and threat-agnostic anomaly detection method for LLM-based MASs that enables the detection of a diverse set of threats without requiring any training or adjustments for the desired use case.
- To the best of our knowledge, we are also the first to utilize the MAS's configuration (e.g., agent's roles, tasks, and available tools) to identify deviations in the system's execution workflow, which was proven to be effective in an ablation study.
- We performed an extensive evaluation on six MAS applications developed on two MAS platforms, which demonstrated MAStitch's effectiveness in anomaly detection when faced with a variety of MAS threats.

2 RELATED WORK

Initial security research on LLM-based systems primarily focused on the single-agent setting, aiming to develop trustworthy models aligned with human values and safety objectives [24]. Considerable effort has been invested in addressing specific safety concerns, resulting in techniques, such as safety-specific fine-tuning, that enhance adherence to instruction-following and reduce harmful completions [1]. Complementing this, input-output safeguards like Llama Guard employ targeted filtering mechanisms to detect and prevent unsafe or policy-violating utterances during interaction [17]. As LLM agents increasingly integrate external tools to expand their capabilities, new frameworks have been introduced; for example, ToolLLM was developed to enable effective API usage [31], and ToolSword investigates safety risks associated with unsafe tool invocations and proposes detection frameworks [39].

However, these approaches mainly operate within specific agent contexts and lack mechanisms to detect system-level threats arising from complex interactions in MASs. Notably, vulnerabilities such as the propagation of misinformation across agents or cascading failures triggered along agent chains exemplify such threats [2, 9, 23, 35]. Despite these advancements, there is a lack of runtime anomaly detection techniques capable of monitoring global anomalous behavior patterns in MASs; the absence of such techniques is particularly concerning given the increasing use of LLM-based MAS as decision-making and automation pipelines in diverse domains.

To address this gap, recent research has focused on the use of benchmarking and evaluation frameworks to systematically assess safety risks in LLM-based agents, enabling proactive identification of vulnerabilities prior to deployment. For instance, in ToolEmu [32] a language-model-emulated sandbox for testing agents across diverse tool-use scenarios is paired with an automatic safety evaluator for quantifying failure risks such as data leaks or financial losses. Other research aimed to benchmark LLMs' safety; for example, the study introducing R-Judge [40] used multi-turn interaction records across 27 risk scenarios to evaluate risk awareness, revealing gaps in behavioral safety judgment. Similarly, research on the Agent Security Bench (ASB) [41] applied formalized attacks like prompt injection and MP across 10 real-world scenarios involving over 400 tools, revealing vulnerabilities in agent operations. The results of these studies highlight the need for standardized testing to uncover high-impact risks associated with agent autonomy, which would contribute to the development of safety assessments for MASs.

Drawing on the insights derived from prior benchmarking work, recent advancements in MAS security have emphasized topology-aware and graph-based methods for anomaly detection and remediation, addressing the interconnected nature of agent interactions. For instance, G-Safeguard [38], a topology-guided security framework that uses graph neural networks to detect malicious agents and remediate attacks in multi-agent graphs, demonstrated resilience against prompt injection, MP, and tool exploitation. BlindGuard [25], an unsupervised detection framework that leverages hierarchical agent encodings and corruption-guided detection, was shown to enable adaptability to unknown attacks. SentinelAgent [14] couples structural execution graphs with LLM-powered oversight, enabling system-level semantic anomaly detection and contextual intervention across diverse MAS security risks. While these methods shed light on failure propagation in MASs, their structural assumptions and use case limitations may reduce their effectiveness in dynamic real-world environments.

Parallel to these academic approaches, commercial-grade observability platforms like LangSmith, LangFuse, and MLflow have become ubiquitous tools for monitoring LLM applications [4, 20, 36]. These platforms excel at capturing detailed execution traces, but their primary function is to provide visibility for human operators rather than performing online threat detection. The need for deeper, autonomous analysis has spurred further academic research on the development of more intelligent and adaptive monitoring systems.

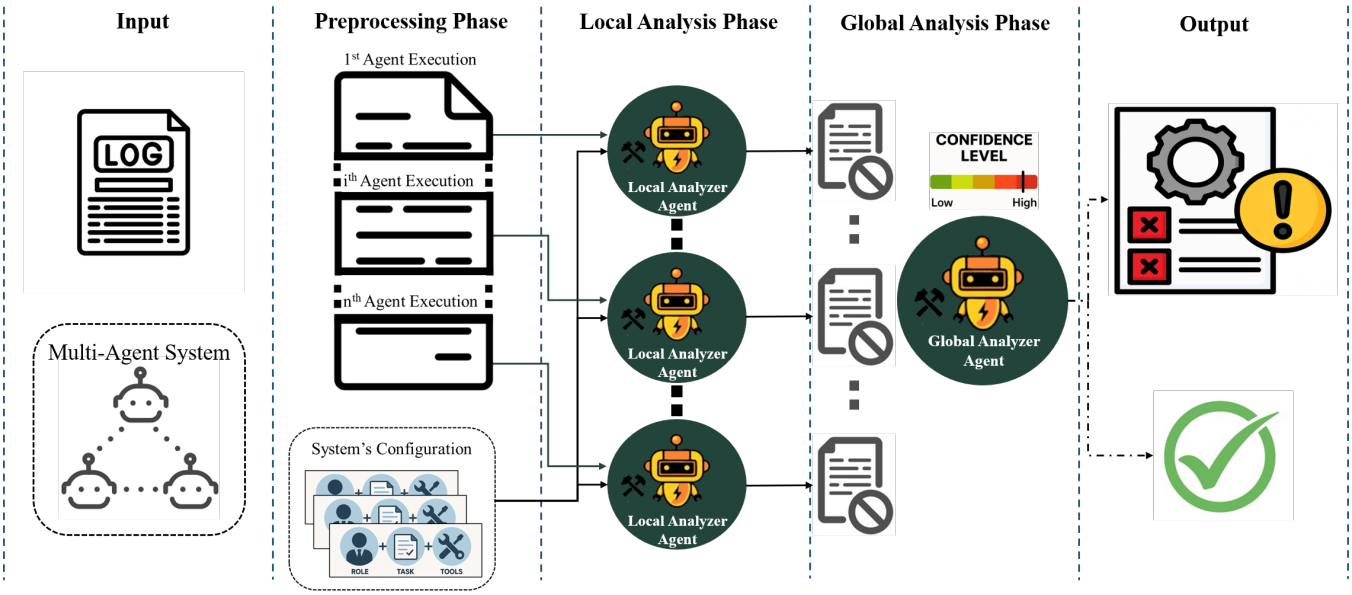

Figure 1: Overview of MAStitch's workflow, which consists of (1) a preprocessing phase, in which the input execution log of n agent's executions is divided into separate agent's execution chunks $[c_1, ...c_i, .., c_n]$, and the expected system behavior is extracted from the MAS application, (2) a local analysis phase which involves multiple LAAs, each of which analyzes a single execution interval, and (3) a global analysis phase in which the LAAs' outputs are aggregated to produce a unified and interpretable assessment.

Recognizing the importance of the connections between agents and the emergence of cross-agent anomalous patterns, [29] introduced MAST, a systematic taxonomy and anomaly-detection annotator for identifying failures in LLM-based MASs. It employs an LLM-as-a-judge methodology for automated annotation of MAS traces and was shown to achieve high agreement with expert annotators. However, MAST's reliance on a predefined taxonomy, supervised labels, and few-shot prompting limits its generalization to unseen threat types, thereby reducing its applicability in real-world, evolving MAS environments. Similarly, LumiMAS [33], an observability framework combining a three-layer system-logging module, a lightweight anomaly-detection component analyzing execution and semantic features, and an explanation layer comprising an LLM-based classification agent and root-cause analysis agent. Although it demonstrated competitive real-time performance, its dependence on execution logs for training and threshold calibration limits its practicality in real-world settings. Moreover, retraining is required whenever substantial changes occur in the MAS, diminishing its suitability for dynamic or rapidly evolving systems. While both MAST and LumiMAS provide valuable insights on MAS failure characterization and support workflow-level analysis, their reliance on supervised training data or pre-collected datasets creates a critical bottleneck, requiring constant data curation and retraining to remain effective. In contrast, our platform- and threat-agnostic method for LLM-based MAS detects a broad spectrum of threats without requiring training, fine-tuning, or adaptation for specific use cases, enabling plug-and-play deployment across diverse scenarios while eliminating reliance on supervised training data.

3 METHODOLOGY

Recent studies have demonstrated that LLM-driven agents can perform sophisticated reasoning over system and security logs, uncovering behavioral patterns and detecting anomalies that traditional anomaly detection mechanisms often fail to capture [19, 34]. Moreover, employing LLM-based agents as a post-processing component for execution logs enables effective analysis in dynamic environments, where the performance of use-case-specific anomaly detectors is often limited. Building on these insights, our method, MAStitch, employs LLM-based agents for the anomaly detection task in MASs, targeting threats exploited by malicious actors (e.g., adversarial attacks), as well as, threats stemming from dangerous behavior of the system (e.g., insecure code generation).

Figure 1 present MAStitch's workflow. Its input consists of an execution log, along with the monitored MAS application's implementation. The execution logs are obtained through the monitoring layer introduced in [33], which defines a set of unique, predefined events capturing the key stages of the MAS application's lifecycle. The execution logs are divided, by our method, into separate agent's execution chunks, thereby allowing independent analysis of the chunks, as well as their (meaning multiple chunks) aggregation in a system-level assessment. MAS's agents' core elements, such as roles, goals, and capabilities (e.g., tools, memory, and planning mechanisms) [3, 42] define the system's configurations, which in our method serve as the agent's expected baseline behavior. The processed inputs are fed into the LAAs, each of which analyzes a different agent's execution to produce a concise summary of the agent's activities. Subsequently, the GAA consolidates the intermediate findings produced by the LAAs to analyze cross-agent

dynamics, generating an evolving high-level assessment that issues alerts upon threat detection, accompanied by comprehensive explanations that constitute the final output of our method. MAStitch's output, serves as an actionable indicator of potential threats in a MAS, enabling forensic analysis to identify problematic segments, understand their causes, and trace the aggregated reasoning behind the alert. To further improve the decision-making accuracy and increase analytical depth of the operating agents, each of them is equipped with an auxiliary internet search tool [5], enabling fact-validation and exploring external sources. Moreover, MAStitch's agentic design also enables the incorporation of additional tailored tools for the anomaly detection task, improving its ability to address specific threats.

MAStitch's architecture comprises three key phases: (1) a preprocessing phase, which divides an execution log into agent-level intervals (chunks) and extracts the system's behavioral characteristics; (2) the local analysis phase, in which the LAAs evaluate each chunk independently; and (3) the global analysis phase, in which the GAA integrates the LAAs' outputs in an evolving global assessment.

3.1 Preprocessing Phase

Before analysis begins, the raw execution logs are divided into discrete chunks, each representing a complete agent execution interval from invocation to termination as illustrated in Figure 1. Let a system log be denoted as $L = \{c_1, c_2, \ldots, c_n\}$, where each c_i corresponds to agent i's execution interval, and n is the total number of agent executions in log L. c_i encapsulates an entire agent-execution cycle while preserving the temporal order of events, thus enabling localized yet temporally coherent analysis. The configuration metadata is easily constructed from the MAS implementation due to our method's fully automated configuration extraction process, leveraging the available class attributes defined in the CrewAI and LangGraph platforms [6, 21]. Rather than relying on additional supervision or domain-specific training, MAStitch leverages metadata commonly available in most MASs, including (i) agent roles, goals, and available tools, and (ii) task descriptions with their expected outputs, which can be referred to as the MAS's configuration (an example is provided in the supplementary material). This configuration is used to define the system's expected baseline behavior, providing a reference frame that enables our method to identify anomalies. This approach follows the design principles of Audit-LLM [34], which demonstrated the benefits of embedding normal operational profiles into LLM-based evaluations for enhanced accuracy and interpretability. MAStitch's configuration extraction process is performed in a passive and non-disruptive manner designed to allow effortless reconstruction whenever the MAS architecture undergoes modifications.

3.2 Local Analysis Phase

Focusing on agent-level analysis, the LAA analyzes an agent's execution interval c_i independently, as shown in Figure 1. The LAA functions as a domain-agnostic evaluator, leveraging LLM-based reasoning to identify behavioral deviations (e.g., the use of tools unrelated to the agent's task or the reception of unexpected inputs) and inconsistencies within each execution trace. Its objectives are to assess execution integrity, identify anomalous actions, and produce interpretable summaries of observed agent behavior. The LAA's structure enables fine-grained temporal attribution of anomalies while ensuring scalability across multiple agents. Formally, for each $c_i \in L$, the LAA generates a structured output record:

$$R_i = (VS_i, E_i, I_i, S_i),$$

where $VS_i \in [0, 100]$ denotes the validity score, a quantitative indicator of the chunk's integrity and behavioral deviation relative to the configuration baseline; E_i, I_i, and S_i represent the explanation for the validity score, the set of detected issues, and a concise summary, respectively. The agent is instructed to apply the following scoring scheme: low scores ($VS_i < 45$) indicate stable operation, mid-range values ($45 \leq VS_i < 75$) suggest minor deviations, and high scores ($VS_i \geq 75$) correspond to critical anomalies likely to compromise system reliability or security.

The LAA's reasoning prompt (provided in the supplementary material) explicitly incorporates configuration-derived context to assess whether the actions within c_i conform to the expected operational behavior of the evaluated agent. This contextual grounding allows the LAA to differentiate benign irregularities, such as creative but valid behavior, from true violations involving data corruption, unauthorized access, or malicious prompt manipulation. To reduce latency, chunk analysis is performed concurrently by multiple LAAs, enabling parallel evaluation of large-scale logs and thereby accelerating the overall anomaly detection process.

3.3 Global Analysis Phase

As seen in Figure 1, following the analysis performed by the LAAs, the GAA performs higher-order reasoning by aggregating and contextualizing their outputs. The GAA's reasoning prompt (provided in the supplementary material) consists of the GAA's dynamic contextual memory:

$$H_i = \{(ID_j, VS_j, E_j, I_j, S_j) \mid 1 \leq j \leq i\},$$

where H_i accumulates all records R_j up to step i, and ID_j denotes the corresponding chunk identifier. This evolving memory enables the GAA to detect temporal dependencies, recurring patterns, and escalation trends that may be imperceptible in the evaluation of individual chunks.

From H_i, the GAA estimates the compromise confidence level $CL_i \in [0, 100]$, which represents the likelihood that the system is operating outside its defined behavioral boundary. An alert is raised when one of the following conditions is met:

$$\exists i \ (VS_i \geq V) \ \vee \ (CL_i \geq C),$$

where V, C denote predefined thresholds for the per-chunk validity score and cumulative confidence level, respectively. This dual criteria mechanism allows MAStitch to issue an early alert for distinct localized anomalies while also capturing gradual, cross-chunk escalations indicative of distributed compromise. To improve computational efficiency, the GAA incorporates an early-stopping mechanism that terminates the analysis once an alert condition is met, thereby minimizing unnecessary overhead. The GAA employs meta-analytical reasoning to interpret LAA outputs, infer causal relationships between anomalies, and determine whether deviations are isolated or systemic. Through this hierarchical synthesis,

MAStitch produces a unified and interpretable assessment of MAS integrity, providing transparent justifications for each alert.

MAStitch's output comprises the evaluation results for each analyzed chunk c_i, including the assigned validity score VS_i, explanation E_i, set of detected issues I_i, and behavioral summary S_i, together with the evolving cumulative assessment produced by the GAA, providing a comprehensive and interpretable diagnostic report suitable for post-incident analysis.

3.4 Threat Detection Auxiliary Tools

MAStitch's architecture naturally generalizes to support the integration of domain-specific and task-adaptive tools, which can be autonomously selected and invoked at runtime. Those tools can enhance the agents' analytical reasoning with specialized capabilities, such as code analyzers, data integrity validators, and cybersecurity scanners, to further improve the anomaly detection process.

In this work, we incorporated a fundamental auxiliary module, the internet search tool [5], to support external information utilization and input validation during runtime analysis. This tool enables the LAA and GAA to autonomously query external sources when encountering potentially unsupported claims, suspicious user inputs, or incomplete contextual information. MAStitch's use of the internet search tool exemplifies the principle of tool-augmented reasoning, enabling agents to cross-validate internal inferences with external knowledge beyond their static data.

4 EVALUATION

This section describes the evaluation settings and results, while further details, such as additional information about the applications used, as well as additional results, are provided in the supplementary material.

4.1 Evaluation Settings

4.1.1 MAS Applications. To demonstrate our method's effectiveness against different threat types, we utilize the execution logs dataset and monitoring layer provided by [33]. The evaluation consists of six different MAS applications, implemented using two popular MAS platforms: CrewAI [7] and LangGraph [22].

On the CrewAI platform, we used the Trip Planner, Instagram Post, and Game Builder applications [5] to respectively demonstrate the detection of DPI attacks, IPI attacks, and insecure coding generation. For MP attack detection, we utilized the Real Estate Team application [13], which includes a retrial-augmented-generation database. On the LangGraph platform, we employed the GenFic application (detailed in the supplementary material) and an adapted version of the Trip Planner application from [5], focusing on DPI attack detection. As our method targets the detection of threats within MASs, specifically adversarial attacks and insecure code generation, the detection of other types of threats, such as hallucination and bias, is outside the scope of this paper.

4.1.2 Anomaly Generation. The execution logs dataset consists of various targeted types of attacks that emulate attacks MASs face in the real world. The dataset's DPI attack variants manipulate the agent's inputs, including instructions that force outputs into invalid formats, which cause downstream parsing and processing errors, the injection of misleading information that steers reasoning toward attacker-defined goals, and backdoor triggers tied to agent identity that alter behavior when activated. The IPI attack exploit interaction channels by delivering malicious HTML or deceptive links, encouraging agents to perform repeated retrievals that drain resources and lead to harmful action loops. The dataset's MP attacks compromise the integrity of the utilized database by contaminating stored documents, which contain poisoned evidence when retrieved, leading agents to produce faulty or harmful outputs. In addition, insecure coding generation execution logs were collected by analyzing the logs created in the Game Builder application, using the Bandit code scanner [8]; logs with medium and high venerability were classified as anomalous, and the rest were classified as benign. Collectively, these anomalies degrade both the correctness and reliability of the MAS outputs.

4.1.3 Data Collection and Splitting. Each of the test and validation sets contained 200 execution logs, balanced between anomalous and benign samples. The sets were constructed to guarantee that no execution log appeared in more than one set.

4.1.4 MAS LLM Configuration. Similar to [33], to demonstrate the generalizability of the examined methods, in our evaluation we use two different LLMs as the underlying models for the agents in the examined MAS applications: OpenAI GPT-4o mini [16] and OpenAI o3-mini [27]. GPT-4o mini is a streamlined version of GPT-4o, optimized for speed and computational efficiency while maintaining competitive performance. o3-mini is a compact LLM designed to provide efficient and reliable results across a wide range of natural language processing tasks.

4.1.5 Baselines. We compare our method's performance to several baseline methods that include the LumiMAS combined anomaly detection approach [33], which trains an LSTM autoencoder on MAS execution logs using extracted numeric and semantic features. An additional baseline is the MAST LLM annotator [29], which utilizes an LLM-as-a-judge with a predefined taxonomy and few-shot settings to identify failures in MAS execution traces (logs). We also implement a basic LLM-as-a-judge [43] with an engineered prompt for the anomaly detection task and an agent-as-a-judge [44] variant with the task and agent definition based on the same engineered prompt; we equip the agent with scraping and searching tools, enabling it to retrieve additional information to improve its effectiveness in detecting anomalies. The engineered prompts are provided in the supplementary material.

4.1.6 Metrics. We assess anomaly detection performance using the following widely adopted binary classification metrics: accuracy, precision, recall, F1 score, and false positive rate (FPR). To assess the methods' efficiency, we also evaluate overhead resource consumption in terms of average latency and token count. In the anomaly detection results tables, the best performance in each column is marked in bold, and the second-best is underlined. The arrows (↑, ↓) indicate whether the optimal value is higher or lower.

4.1.7 Implementation Details. In our evaluation, the proposed method is implemented using Python 3.11, with experiments conducted using CrewAI 0.152.0 and LangGraph 0.6.3. The LAA and GAA are implemented on the CrewAI platform, each configured with a

Table 1: Anomaly detection results obtained on the CrewAI applications, with GPT-4o mini as the underlying model.

Threat Type	Method	Performance					Overhead	
		Accuracy ↑	F1 ↑	Recall ↑	Precision ↑	FPR ↓	Latency ↓	Token Count ↓
Direct Prompt Injection Misinformation	LLM-as-a-judge	**0.910**	**0.911**	**0.920**	0.902	0.100	7.142	4446.0
	Agent-as-a-judge	0.780	0.780	0.780	0.780	0.220	9.770	1973498.0
	MAST	0.695	0.747	0.900	0.638	0.510	11.612	80227.5
	LumiMAS	0.815	0.830	0.900	0.769	0.270	**0.084**	-
	MAStitch (Ours)	0.850	0.826	0.710	**0.986**	**0.010**	12.440	71253.0
Direct Prompt Injection Exhaustion	LLM-as-a-judge	0.525	0.228	0.140	0.609	0.090	6.675	4485.1
	Agent-as-a-judge	0.560	0.450	0.360	0.600	0.240	9.334	2051214.0
	MAST	0.710	0.760	0.920	0.648	0.500	10.717	78204.5
	LumiMAS	0.865	0.881	**1.000**	0.787	0.270	**0.096**	-
	MAStitch (Ours)	**0.950**	**0.947**	0.900	**1.000**	**0.000**	13.110	76357.0
Direct Prompt Injection Backdoor	LLM-as-a-judge	0.625	0.490	0.360	0.766	0.110	7.034	4542.5
	Agent-as-a-judge	0.590	0.494	0.400	0.645	0.220	9.106	1970618.0
	MAST	0.735	0.774	**0.910**	0.674	0.440	11.986	77987.3
	LumiMAS	0.620	0.573	0.510	0.654	0.270	**0.083**	-
	MAStitch (Ours)	**0.925**	**0.920**	0.860	**0.989**	**0.010**	13.230	74040.0
Indirect Prompt Injection	LLM-as-a-judge	0.745	0.730	0.690	0.775	0.200	12.522	6754.7
	Agent-as-a-judge	0.620	0.608	0.590	0.628	0.350	21.620	6653435.0
	MAST	0.805	0.825	0.920	0.748	0.310	11.495	90764.8
	LumiMAS	**0.970**	**0.970**	0.970	**0.970**	**0.030**	**0.142**	-
	MAStitch (Ours)	0.965	0.966	**0.990**	0.943	0.060	14.390	75944.0
Insecure Coding	LLM-as-a-judge	0.630	0.439	0.290	**0.906**	**0.030**	6.871	3696.0
	Agent-as-a-judge	0.605	0.415	0.280	0.800	0.070	11.979	1714570.0
	MAST	0.525	0.644	0.860	0.515	0.810	10.903	50494.6
	LumiMAS	0.555	0.604	0.680	0.544	0.360	**0.071**	-
	MAStitch (Ours)	**0.780**	**0.798**	**0.870**	0.737	0.310	11.540	37860.0
Memory Poisoning	LLM-as-a-judge	**0.965**	**0.966**	**0.990**	**0.943**	**0.060**	6.645	3826.4
	Agent-as-a-judge	0.875	0.868	0.820	0.921	0.070	10.492	1706985.0
	MAST	0.315	0.149	0.120	0.197	0.490	8.503	34547.2
	LumiMAS	0.500	0.561	0.640	0.500	0.640	**0.032**	-
	MAStitch (Ours)	0.940	0.941	0.960	0.923	0.080	11.210	25838.0

maximum of 5 iterations before producing a final response and a maximum task execution time of 30 seconds. we set the thresholds of 70 and 75 for the validity and confidence scores, respectively. Those values were chosen base on the F1 score performance on the validation set which includes benign and DPI attack execution logs of the Trip Planer app on the CrewAI platform. GPT-4.1 mini is used as the core component for both the LAA and GAA, and for all LLM/agent-based baselines. The GPT-4.1 series models provide a large context window (up to 1M tokens), which is crucial for analyzing extended execution logs. The mini version balances inference time and token usage, maintaining strong reasoning capabilities while remaining computationally efficient. Moreover, Hugging Face's Agent Leaderboard [10] ranks GPT-4.1 mini within the top-five models in terms of action completion, and session duration, underscoring its suitability for LLM-based agent tasks.

4.2 Evaluation Results

4.2.1 Anomaly Detection Results. Table 1 presents the detection performance results for the examined methods against six threat types (rows) on the CrewAI applications, with GPT-4o mini as the underlying model As can be seen, for the DPI backdoor attack, MAStitch outperforms all baselines, demonstrating its superior ability to detect malicious attempts to alter agent identity by leveraging configuration-aware contextual reasoning. Examining the DPI Exhaustion attack results, we observe that both our method and LumiMAS outperform the LLM-as-a-judge and agent-as-a-judge, with MAST following closely. This suggests that the threat exhibits global system dynamic behaviors that are effectively captured by approaches designed to detect cross-agent anomalous patterns.

To demonstrate the generalizability and dynamic capabilities of our method, we also evaluate our method without any additional optimization or adaptations for the rest of the threat scenarios. In the case of MP, MAStitch achieves the second-best performance,

Table 2: Anomaly explanation evaluation scores obtained on CrewAI applications, with GPT-4o mini as the underlying model.

Criterion	DPI Misinformation	DPI Exhaustion	DPI Backdoor	MP	Insecure Coding	IPI
Completeness (0-30)	25.40	23.80	24.70	24.10	22.30	24.90
Evidence quality (0-30)	24.60	22.90	25.10	23.70	21.80	24.30
Pattern detection (0-40)	32.65	30.89	32.63	31.69	30.43	33.15
Total score (0-100)	**82.65**	**77.59**	**82.43**	**79.49**	**74.53**	**82.35**

slightly below that of the LLM-as-a-judge baseline, demonstrating strong detection capabilities in identifying maliciously injected information retrieved from external sources, which can be attributed to its understanding of the system's expected normal behavior. In contrast, MAST shows comparatively poorer performance for MP, which may stem from its reliance on a fixed predefined taxonomy that fails to detect this type of threat which is not fully addressed by their taxonomy. Examining the IPI attack results, both our method and LumiMAS outperform the LLM-as-a-judge and agent-as-a-judge, with MAST following closely, likely due to the nature of this threat, which requires cross-agent awareness to be effectively detected. While IPI attack share some characteristics with DPI attacks, MAStitch also excels in detecting threats that differ substantially from DPI attacks, such as MP and insecure coding. Overall, on average, our method outperforms the baselines, achieving the lowest average FPR (0.078) and the highest average accuracy, F1 score, recall, and precision values of 0.902, 0.900, 0.882, and 0.930, respectively. MAStitch's strong performance suggests that integrating local agent-level signals, global system-level patterns, configuration-aware reasoning, and auxiliary tailored tools can facilitate the detection of subtle and unseen threats, providing a comprehensive perspective that improves the robustness of anomaly detection in MASs. However, as seen in Table 1, these benefits come at the cost of increased computational overhead compared to lighter-weight approaches; this is likely due to the multiple executions of the LAAs and GAA, and the rich input processed by each agent. The supplementary material provides additional results, including experiments on the CrewAI platform with o3-mini as the underlying model, on the LangGraph platform, and a large-scale experiment (10 agents) demonstrating MAStitch's effectiveness.

4.2.2 Anomaly Explanation Results. We instructed the GAA to produce its output in a structured format that contains specific diagnostic details to support the interpretability of the generated alerts. The objective of our evaluation is to assess whether the explanations produced are coherent, contextually grounded, and provide information that supports subsequent forensic analysis. We evaluate the explanations using both an automated rule-based scoring metric and human validation across the examined applications. The explanations were assessed along three key dimensions: (1) completeness (0-30), which evaluates whether the reasoning addresses all analyzed chunks and provides sufficient detail; (2) evidence quality (0–30), which measures the extent to which supporting details justify the claims; and (3) pattern detection (0–40), which examines whether predefined threat-specific patterns are correctly referenced. Table 2 presents the automated evaluation results. As can be seen, the explanations obtain an average score of 79.84 (out of 100), reflecting their informativeness and relevance. Our method's high proficiency in capturing relevant threat patterns and articulating causal relationships between the observed anomalies and their underlying sources contributing to the interpretability of the alerts. However, the explanations on the insecure code generation threat were found to be comparatively less effective, likely due to its intrinsic nature; this threat originates from internal system behavior rather than externally induced adversarial manipulation, thereby posing greater challenges for interpretive consistency. Similar conclusions are also reached in the human validation systematic analysis performed to validate the coherent and contextual alignment of the generated explanations. Additional details on the explanation evaluations are provided in the supplementary material.

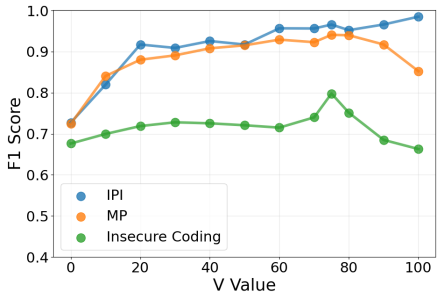

Figure 2: Effect of altering the validity threshold (V) on MAStitch's performance across different threat types.

4.3 Ablation Study

To demonstrate the contribution of the MAS's configuration context and the internet search tool, we conducted three additional experiments on the CrewAI applications, with GPT-4o mini as the underlying model. Table 3 reports the average values of the performance metrics for the full MAStitch method (with the configuration context and internet search tool), along with the results from the three experiments for the ablation variants, which are called MAStitch w/o Config + Tool, MAStitch w/o Config, and MAStitch w/o Tool. The full MAStitch method achieves the best results across all performance metrics, while the ablation variants exhibit modest but consistent degradation. Excluding only the configuration context (MAStitch w/o Config) diminishes the system's capacity to leverage predefined normal behavior expectations, resulting primarily in reduced recall. In contrast, removing the internet search tool (MAStitch w/o Tool) prevents the system from performing external evidence validation for fact-dependent anomalies, leading to an increase in false positives. Interestingly, the performance of

Table 3: Ablation study results obtained on the CrewAI applications, with GPT-4o mini as the underlying model.

Method	Performance					Overhead	
	Accuracy ↑	F1 ↑	Recall ↑	Precision ↑	FPR ↓	Latency ↓	Token Count ↓
MAStitch w/o Config + Tool	0.854	0.849	0.838	0.884	0.130	**12.282**	**53407.0**
MAStitch w/o Config	0.842	0.827	0.783	0.902	0.100	13.815	57890.5
MAStitch w/o Tool	0.849	0.833	0.815	0.897	0.117	12.973	60238.7
MAStitch (Ours)	**0.902**	**0.900**	**0.882**	**0.930**	**0.078**	12.653	60215.3

MAStitch w/o Config + Tool is relatively close to that of the two single-component ablation variants. This can be attributed to partial redundancy between the context and evidence components; while each independently contributes to performance, both rely on the same underlying anomaly scoring and aggregation mechanisms, which preserve a baseline level of detection capability even in their absence. The ablation variants still perform better than the baselines on average, underscoring the robustness of the architecture and demonstrating the contribution of both components to the performance gains. The fact that the configuration-free variant of MAStitch outperforms the examined baselines even without normal behavior context indicates that MAStitch remains effective in environments where system metadata is unavailable.

We also calibrate the selected hyperparameters of our method, C and V. To this end, we assess the impact of varying each parameter's value while keeping the other fixed and compare the resulting F1 scores on unseen threats. Figure 2 illustrates the changes in the F1 score in response to variations in the threshold V. As can be seen, the threshold of $V = 75$, selected based on validation results against the DPI attack, may not be strictly optimal, yet it demonstrates appropriate threshold calibration and generalization. Similar analysis for C is provided in the supplementary material.

5 DISCUSSION

5.1 MAStitch's Limitations

While MAStitch demonstrates strong performance as a real-time anomaly detection system, it has some limitations. First, the ease of use and strong threat detection performance come at the cost of relatively high overhead, which in some cases can reduce the relevance or timeliness of the alert. This overhead primarily stems from the use of LLM-based agents, which are highly capable but computationally expensive to operate. Importantly, each detected anomaly is accompanied by an explanation that describes the rationale behind the alert, thereby providing guidance about how it can be addressed. Second, the method's reliance on LLM inference introduces a potential point of failure, as the agents can hallucinate or be influenced by outside attackers. Finally, MAStitch is constrained by the inherent context window limitations of current LLMs. As a result, MAStitch's performance may degrade when handling execution traces that exceed its context window, as the input must be truncated, which can lead to the loss of anomaly-related details.

5.2 Early Stopping

Early stopping offers efficiency benefits by reducing runtime and token consumption, but it also introduces a trade-off: by halting too soon, subtle or delayed anomalies may be overlooked, while waiting too long reduces the efficiency gains. Moreover, in some cases it may be valuable to continue analyzing the full execution log to capture any anomalies that emerge after the alert, ensuring that propagation effects or additional issues are not overlooked. Our design balances this tension by combining strict per-chunk threshold with a global confidence threshold, ensuring that termination occurs only when the evidence is strong enough to justify it. Note that analysis of the entire execution log can be enabled by setting the parameters C and V to 100, disabling the early stopping mechanism.

5.3 Overhead vs. Accuracy

The proposed method introduces computational and token overhead, because it employs multiple LLM-based agents' executions rather than a single lightweight classifier, especially when analyzing large logs with executions of many agents. However, as presented in Section 4.2, this overhead is balanced by strong detection performance and interpretability. Unlike the baselines, MAStitch's workflow produces structured explanations, detailed reasoning, and identified anomalous patterns. In practice, the overhead remains acceptable due to optimizations such as early termination, and limiting the usage of tools. As a result, the system prioritizes detection reliability and actionable insights over latency and token cost, an important consideration in MASs where overlooking a single threat can lead to critical failures (e.g., in the healthcare domain).

6 CONCLUSION

In this paper, we introduced MAStitch, a novel, platform- and threat-agnostic, plug-and-play method for threat detection in LLM-based MASs. MAStitch stitches together fine-grained local agent insights with the results of global system dynamic analysis, enabling comprehensive anomaly detection. Our method utilizes the system's configuration to validate execution logs, defining the expected system behavior, which was proven useful throughout the comprehensive evaluation performed. The evaluation also demonstrates MAStitch's robust performance against diverse attacks and its ability to provide clear, interpretable insights regarding MASs' integrity.

Our work advances agentic research by addressing critical challenges in verification, safety, and evaluation. Future work may focus on reducing the method's overhead and integrating recently standardized communication protocols (e.g., the Model Context Protocol (MCP)). The incorporation of domain-specific threat detection auxiliary tools could further extend MAStitch's functionality across diverse operational settings and strengthen its overall performance, and this could be explored in future research as well.

REFERENCES

[1] F Bianchi, M Suzgun, G Attanasio, P Rottger, D Jurafsky, T Hashimoto, J Zou, et al. 2024. SAFETY-TUNED LLAMAS: LESSONS FROM IMPROVING THE SAFETY OF LARGE LANGUAGE MODELS THAT FOLLOW INSTRUCTIONS. In *12th International Conference on Learning Representations, ICLR 2024*. International Conference on Learning Representations, ICLR.

[2] Angana Borah, Marwa Houalla, and Rada Mihalcea. 2025. Mind the (belief) gap: Group identity in the world of llms. In *Findings of the Association for Computational Linguistics: ACL 2025*. 18441–18463.

[3] Yuheng Cheng, Ceyao Zhang, Zhengwen Zhang, Xiangrui Meng, Sirui Hong, Wenhao Li, Zihao Wang, Zekai Wang, Feng Yin, Junhua Zhao, et al. 2024. Exploring large language model based intelligent agents: Definitions, methods, and prospects. *arXiv preprint arXiv:2401.03428* (2024).

[4] MLflow Contributors. 2024. MLflow. https://mlflow.org/. Open source platform for managing the ML lifecycle: experimentation, reproducibility, deployment, and a central model registry.

[5] Crew AI Inc. 2024. CrewAI Examples. https://github.com/crewAIInc/crewAI-examples.

[6] CrewAI Inc. 2025. Agents — CrewAI Documentation. https://docs.crewai.com/en/concepts/agents.

[7] CrewAI Inc. 2025. CrewAI Documentation. https://docs.crewai.com/.

[8] Bandit Developers. 2025. Bandit Documentation. https://bandit.readthedocs.io/en/latest/.

[9] Mohamed Amine Ferrag, Norbert Tihanyi, Djallel Hamouda, Leandros Maglaras, Abderrahmane Lakas, and Merouane Debbah. 2025. From prompt injections to protocol exploits: Threats in LLM-powered AI agents workflows. *ICT Express* (2025).

[10] Galileo-AI. 2024. Agent Leaderboard. https://huggingface.co/spaces/galileo-ai/agent-leaderboard.

[11] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM workshop on artificial intelligence and security*. 79–90.

[12] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: a survey of progress and challenges. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*. 8048–8057.

[13] Brandon Hancock. 2024. CrewAI RAG Deep Dive. https://github.com/bhancockio/crewai-rag-deep-dive.

[14] Xu He, Di Wu, Yan Zhai, and Kun Sun. 2025. SentinelAgent: Graph-based Anomaly Detection in Multi-Agent Systems. *arXiv preprint arXiv:2505.24201* (2025).

[15] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. 2023. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The twelfth international conference on learning representations*.

[16] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276* (2024).

[17] Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, et al. 2023. Llama guard: Llm-based input-output safeguard for human-ai conversations. *arXiv preprint arXiv:2312.06674* (2023).

[18] Neil Kale, Chen Bo Calvin Zhang, Kevin Zhu, Ankit Aich, Paula Rodriguez, Scale Red Team, Christina Q Knight, and Zifan Wang. 2025. Reliable Weak-to-Strong Monitoring of LLM Agents. *arXiv preprint arXiv:2508.19461* (2025).

[19] Enis Karaarslan, Esin Güler, Efe Emir Yüce, and Cagatay Coban. 2025. Towards Log Analysis with AI Agents: Cowrie Case Study. *arXiv preprint arXiv:2509.05306* (2025).

[20] LangChain. 2024. LangSmith. https://www.langchain.com/langsmith. Unified observability and evaluation platform for AI agents and LLM applications. Includes tracing, prompt engineering, and performance evaluation features.

[21] LangChain. 2025. Workflows and Agents — LangGraph Documentation. https://docs.langchain.com/oss/python/langgraph/workflows-agents.

[22] LangGraph. 2023. LangGraph: Building stateful multi-actor applications with LLMs. https://github.com/langchain-ai/langgraph.

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

[24] Yang Liu, Yuanshun Yao, Jean-Francois Ton, Xiaoying Zhang, Ruocheng Guo, Hao Cheng, Yegor Klochkov, Muhammad Faaiz Taufiq, and Hang Li. 2023. Trustworthy LLMs: a Survey and Guideline for Evaluating Large Language Models' Alignment. In *Socially Responsible Language Modelling Research*.

[25] Rui Miao, Yixin Liu, Yili Wang, Xu Shen, Yue Tan, Yiwei Dai, Shirui Pan, and Xin Wang. 2025. BlindGuard: Safeguarding LLM-based Multi-Agent Systems under Unknown Attacks. *arXiv preprint arXiv:2508.08127* (2025).

[26] Ana Nunez, Nafis Tanveer Islam, Sumit Kumar Jha, and Peyman Najafirad. 2024. Autosafecoder: A multi-agent framework for securing llm code generation through static analysis and fuzz testing. *arXiv preprint arXiv:2409.10737* (2024).

[27] OpenAI. 2025. OpenAI o3-mini System Card. https://cdn.openai.com/o3-mini-system-card-feb10.pdf. System card technical report.

[28] OWASP Foundation. 2023. OWASP Top 10 for Large Language Model Applications. https://owasp.org/www-project-top-10-for-large-language-model-applications/.

[29] Melissa Z Pan, Mert Cemri, Lakshya A Agrawal, Shuyi Yang, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Kannan Ramchandran, Dan Klein, et al. 2025. Why do multiagent systems fail?. In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*.

[30] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. 2024. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*. 15174–15186.

[31] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2024. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *The Twelfth International Conference on Learning Representations*.

[32] Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. 2024. Identifying the Risks of LM Agents with an LM-Emulated Sandbox. In *The Twelfth International Conference on Learning Representations*.

[33] Ron Solomon, Yarin Yerushalmi Levi, Lior Vaknin, Eran Aizikovich, Amit Baras, Etai Ohana, Amit Giloni, Shamik Bose, Chiara Picardi, Yuval Elovici, et al. 2025. LumiMAS: A Comprehensive Framework for Real-Time Monitoring and Enhanced Observability in Multi-Agent Systems. *arXiv preprint arXiv:2508.12412* (2025).

[34] Chengyu Song, Linru Ma, Jianming Zheng, Jinzhi Liao, Hongyu Kuang, and Lin Yang. 2024. Audit-llm: Multi-agent collaboration for log-based insider threat detection. *arXiv preprint arXiv:2408.08902* (2024).

[35] Hang Su, Jun Luo, Chang Liu, Xiao Yang, Yichi Zhang, Yinpeng Dong, and Jun Zhu. 2025. A Survey on Autonomy-Induced Security Risks in Large Model-Based Agents. *arXiv preprint arXiv:2506.23844* (2025).

[36] Langfuse Team. 2024. Langfuse. https://langfuse.com/. Open source LLM engineering platform for tracing, evaluations, prompt management, and metrics to debug and improve LLM applications.

[37] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

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

[39] Junjie Ye, Sixian Li, Guanyu Li, Caishuang Huang, Songyang Gao, Yilong Wu, Qi Zhang, Tao Gui, and Xuan-Jing Huang. 2024. Toolsword: Unveiling safety issues of large language models in tool learning across three stages. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2181–2211.

[40] Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, et al. 2024. R-Judge: Benchmarking Safety Risk Awareness for LLM Agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. 1467–1490.

[41] Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. 2025. Agent Security Bench (ASB): Formalizing and Benchmarking Attacks and Defenses in LLM-based Agents. In *ICLR*.

[42] Pengyu Zhao, Zijian Jin, and Ning Cheng. 2023. An in-depth survey of large language model-based artificial intelligence agents. *arXiv preprint arXiv:2309.14365* (2023).

[43] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems* 36 (2023), 46595–46623.

[44] Mingchen Zhuge, Changsheng Zhao, Dylan R Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, et al. 2024. Agent-as-a-Judge: Evaluate Agents with Agents. In *Forty-second International Conference on Machine Learning*.