

Agent System Operations: Categorization, Challenges, and Future Directions

Zexin Wang^{*||}, Changhua Pei^{*†}, Yuanhao Liu[†], Jingjing Li^{*}, Yintong Huo[¶], Quan Zhou^{*}, Haotian Si^{*}, Hang Cui^{*}, Zihan Liu^{*}, Jianhui Li^{*}, Gaogang Xie^{*}, *Senior Member, IEEE*, Fei Sun[§], Dan Pei[§], *Senior Member, IEEE*, David Lo[¶], *Fellow, IEEE*

^{*} Computer Network Information Center, Chinese Academy of Sciences

[†] Hangzhou Institute for Advanced Study, University of Chinese Academy of Sciences [‡] Tsinghua University

[§] Institute of Computing Technology, Chinese Academy of Sciences

[¶] Singapore Management University ^{||} University of Chinese Academy of Sciences

Abstract—As the reasoning capabilities of Large Language Models (LLMs) continue to advance, LLM-based agent systems offer advantages in flexibility and interpretability over traditional systems, garnering increasing attention. However, despite widespread research interest in and industrial application of agent systems, these systems, like their traditional counterparts, frequently encounter anomalies. These anomalies lead to instability and insecurity, hindering their further development. Therefore, a comprehensive and systematic approach to the operation and maintenance of agent systems is urgently needed. Unfortunately, current research on the operations of agent systems is sparse. To address this gap, we have undertaken a survey on agent system operations with the aim of establishing a clear framework for the field, defining the challenges, and facilitating further development. Specifically, this paper begins by systematically defining anomalies within agent systems, categorizing them into intra-agent anomalies and inter-agent anomalies. Next, we introduce a novel and comprehensive operational framework for agent systems, dubbed Agent System Operations (AgentOps). We provide detailed definitions and explanations of its four key stages: monitoring, anomaly detection, root cause localization, and resolution.

Index Terms—Agent System Operations, Anomaly Detection, Root Cause Localization, Resolution

I. INTRODUCTION

WITH the advent of technologies such as DeepSeek-R1 [1] and Claude [2], the reasoning capabilities of current Large Language Models (LLMs) are continually being enhanced. Leveraging LLMs as powerful cognitive engines, existing LLM-based agent systems, particularly multi-agent systems, have gained the capacity to accomplish a wide array of complex tasks and social simulations [3], especially when equipped with diverse tools [4]. Compared to traditional systems like microservice architectures [5], agent systems offer better automation, enhanced interpretability, and greater flexibility. Consequently, research and industrial applications of agent systems are flourishing, with an increasing number of online services [6], such as customer support and recommendation systems, adopting these agent systems.

However, despite the widespread application of agent systems, they are not without their flaws. Compared to traditional microservice systems, the greater flexibility offered by agent systems also introduces more anomalies [7]. As illustrated in Fig. 3, task execution often fails due to issues such as hallucinations. In role-playing scenarios, an attack on a single agent can lead to the collapse of the entire simulation. Therefore, to maintain the security and stability of agent systems and to facilitate their further development, efficient operations and maintenance are necessary.

Operations technologies have evolved substantially over time, progressing from manual operations to rule-based approaches and, more recently, Artificial Intelligence for IT Operations (AIOps). However, agent systems exhibit fundamentally different characteristics and failure modes from traditional software systems, limiting the direct applicability of existing operations techniques. For instance, conventional log-based anomaly detection may identify abnormal patterns in system logs, but it is inherently inadequate for detecting agent-specific failures, such as orchestration hallucinations. These emerging challenges call for a new generation of operations technologies specifically designed for agent systems.

Currently, there is a lack of comprehensive research on effective operations and maintenance strategies specifically for agent systems. Most studies remain focused on isolated aspects of agent systems rather than addressing their overall operational challenges. For example, Durante et al. [8] expound on agent paradigms and classifications; Chakraborty et al. [9] delve into hallucinations in foundation models, covering their definition and detection methods; Deng et al. [10] explore security issues in multi-agent systems, primarily covering external malicious attacks and categorizing threats into intra-execution security and interaction security. Shi et al. [11] provide detailed insights into security issues and evaluation methods for GUI agents.

To further advance the development of agent systems, this paper introduces the concept of **Agent System Operations (AgentOps)**, a novel operations and maintenance framework specifically designed for agent systems. First, we provide a precise definition of anomalies within agent systems and offer a systematic classification, primarily dividing them into intra-agent and inter-agent anomalies. These two categories

Corresponding author: Changhua Pei. Email: chpei@cnic.cn

This work was funded by the National Natural Science Foundation of China (62202445), and the National Natural Science Foundation of China-Research Grants Council (RGC) Joint Research Scheme (62321166652).

encompass the stages of pre-execution, execution, and post-execution in the agent system lifecycle. Additionally, drawing inspiration from traditional operation practices, we divide the operations and maintenance process for agent systems into four phases: monitoring, anomaly detection, root cause localization, and resolution. For each phase, we identify new challenges that arise within agent systems and propose detailed definitions and potential solutions.

Beyond reviewing existing techniques, this survey distills the stage-specific challenges of AgentOps into two overarching directions for future research: (1) timely and proactive AgentOps, which seeks to anticipate, detect, and mitigate failures before they propagate or escalate; (2) accurate and evidence-grounded AgentOps, which leverages rich, reliable, and context-aware operational evidence to support precise diagnosis and effective recovery. Together, these directions point toward a transition from today's fragmented and predominantly post-hoc practices to proactive, evidence-grounded, and closed-loop operational paradigms for dependable agent systems.

In section II, we present a detailed discussion of anomaly classification within agent systems. Section III introduces the concept of AgentOps. The subsequent four sections focus respectively on key components of AgentOps: monitoring, anomaly detection, root cause localization (also known as failure attribution [12]), and resolution. Section VIII introduces the datasets and benchmarks. Section XI concludes the paper.

II. ANOMALIES IN AGENT SYSTEMS

A. Definition of Anomalies in Agent Systems

The HAL leaderboard [13] shows that success rates remain relatively low across different types of agent systems, suggesting the presence of numerous exceptional cases that impede successful task completion. Prior work, such as *Who&When* [12], hypothesizes that anomalies in agent systems primarily arise at specific steps during task execution. Here, *execution* refers to the phase in which an agent system, after receiving a user instruction, begins reasoning and invokes tools to carry out the task.

However, modern definitions of agent systems extend well beyond the execution phase alone. Both the pre-execution and post-execution phases are also critical to task success. First, in the *pre-execution* phase, as illustrated in Fig. 1, agent frameworks such as Claude Code, Hermes Agent, OpenClaw, and Deep Agent typically perform intent recognition based on the user input and subsequently configure or construct an agent workflow. Such workflows are not static; rather, they are dynamically instantiated and continuously updated as the task evolves. Second, in the *post-execution* phase, task completion does not necessarily imply task success. For example, when Claude Code builds a software project, the process should not be considered successful merely because the build process terminates; the generated project may still contain bugs or fail to satisfy functional requirements. Similarly, in the domain of operations and maintenance, a root-cause analysis task may proceed without any apparent execution-time abnormality, yet still fail due to premature termination or insufficiently fine-grained analysis. These cases demonstrate that post-execution

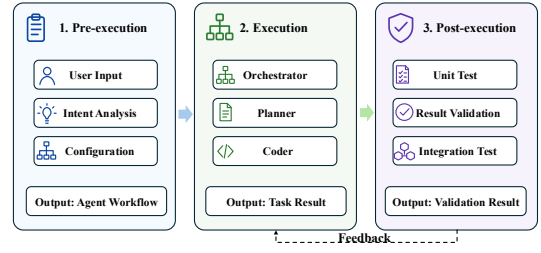


Fig. 1. Workflow of agent systems.

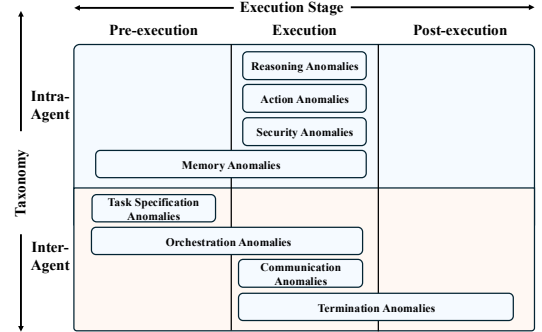


Fig. 2. Definition and taxonomy of anomalies in agent systems.

factors can also critically affect task outcomes. Therefore, we define anomalies in agent systems as any events occurring during the pre-execution, execution, or post-execution phases that disrupt task progression or prevent effective task completion.

Building on the above phase-aware definition, we further analyze anomalies from an orthogonal structural perspective. The pre-execution, execution, and post-execution phases clarify *when* anomalies may occur, but they do not specify *where* such anomalies originate within an agent system. Since agent systems can be broadly classified into single-agent and multi-agent systems, anomalies may originate either from the internal workflow of an individual agent or from interactions among multiple agents. This is analogous to traditional service architectures, where failures may arise within a single service or during inter-service communication. Therefore, as depicted in Fig. 2, we categorize anomalies into two major types: *intra-agent anomalies* and *inter-agent anomalies*. Intra-agent anomalies refer to disruptions rooted in an individual agent's reasoning or tool use, while inter-agent anomalies refer to failures caused by coordination, communication, or dependency issues among agents.

B. Intra-Agent Anomalies

Intra-agent anomalies refer to failures that arise within an individual agent or subagent during task execution. Since most existing agent implementations follow the ReAct paradigm [14], such anomalies commonly manifest as reasoning and action failures. In addition, when agents are deployed in complex vertical domains, naive designs often suffer from memory degradation and security risks caused by long-context processing, unreliable retrieval, and insufficiently constrained tool permissions. These observations highlight the importance

of harness engineering for improving the robustness, controllability, and safety of agent systems.

1) *Reasoning Anomalies*: Reasoning is the cognitive foundation that guides an agent’s subsequent actions and enables complex task completion. Recent studies have proposed various techniques to enhance reasoning, including fine-tuning methods such as SFT [15] and DeepSeek-R1 [1], as well as prompting strategies such as CoT [16]. Despite these advances, reasoning anomalies remain prevalent, with hallucination being the most representative example.

The definition of hallucination has been continuously refined in the literature. Rawte et al. [17] define hallucinations as unreliable generations that contradict known facts, while Gallifant et al. [18] regard them as responses unrelated to the original prompt. Chakraborty et al. [9] summarize hallucinations through four characteristics: compliance, desirability, relevancy, and plausibility. Yang et al. [19] further interpret hallucination as a form of dishonesty, where uncertain answers are expressed with unwarranted confidence. In general, hallucinations can be viewed as anomalous generations that deviate from facts, logic, or task requirements. They are difficult to eliminate because LLMs are sensitive to training data, prone to knowledge forgetting, and unable to synchronously incorporate newly emerging knowledge after training.

2) *Action Anomalies*: Action anomalies occur when agents fail to correctly execute intended operations through tool invocation or function calls. In practice, such failures may arise from inconsistent interfaces, incorrect API selection, delayed execution, malformed parameters, or system-level failures [20]. Moreover, tool use introduces additional safety risks. Wu et al. [21] show that attackers can exploit function-calling mechanisms through jailbreak prompts, inducing LLMs to invoke sensitive functions or bypass restrictions. Although the Model Context Protocol (MCP) standardizes interactions between LLMs and external tools, it does not fully eliminate action anomalies. In real deployments, configuration changes and misconfigured MCP servers may still lead to failed, unintended, or unsafe actions [22].

3) *Memory Anomalies*: Agent memory can be broadly divided into short-term and long-term memory. Short-term memory corresponds to the LLM context. Although the context windows of LLMs continue to expand, they often remain insufficient for complex tasks. Many agent frameworks therefore adopt sliding-window strategies, which may discard early but important instructions. Even when the context length is technically sufficient, LLMs may still underutilize information located in the middle of long contexts, a phenomenon known as the “lost-in-the-middle” problem [23]. PI-LLM [24] also shows that LLMs face intrinsic bottlenecks in working memory.

Long-term memory is typically implemented through vector databases and recalled using RAG. However, retrieval-based memory is vulnerable to inaccurate recall, noisy evidence, and unreliable generation. QE-RAG [25] shows that existing RAG systems are highly sensitive to noise, while Astute RAG [26] highlights conflicts between internal parametric knowledge and retrieved external knowledge. Such conflicts may further induce RAG hallucinations [27]. Chen et al. [28] also observe

that despite extensive technical improvements, the accuracy of current RAG systems remains limited.

4) *Security Anomalies*: Security anomalies refer to cases in which an agent or subagent performs operations that violate predefined security policies, access-control rules, or identity constraints. Typical examples include unauthorized tool invocation, actions beyond assigned privileges, misuse of delegated credentials, and operations executed under an incorrect or unverified identity. Since LLM-based agents interact with tools and APIs through natural-language reasoning, their security boundaries are often implicit and difficult to enforce.

Importantly, security anomalies do not necessarily originate from an agent’s autonomous intention or internal malfunction. They may be induced by adversarial inputs, prompt injection, malicious tool outputs, or compromised external environments, which manipulate the agent into actions outside its intended authorization scope. These risks are closely related to excessive agency, where agents are granted more autonomy or privileges than necessary, allowing erroneous or manipulated outputs to trigger unsafe operations [29], [30]. In multi-agent systems, security anomalies can further arise from improper authorization propagation during delegation or inter-agent communication, where a compromised agent indirectly causes other agents to perform unauthorized actions. Therefore, robust agent systems require explicit identity management, least-privilege tool access, adversarially robust input and tool-output handling, and policy-aware execution mechanisms.

C. Inter-Agent Anomalies

Inter-agent anomalies refer to failures emerging from the interaction, coordination, and collective behavior of multiple agents. Unlike fixed workflows, many recent frameworks, such as Hermes, dynamically construct subagents from available skills and tools according to user inputs, then perform planning, execution, and result verification. As a result, inter-agent anomalies may occur throughout the entire lifecycle. In the pre-execution stage, they mainly involve task specification and orchestration failures caused by ambiguous, incomplete, or malicious inputs and inappropriate subagent construction. During execution, communication anomalies may arise when agents exchange information, invoke tools, or rely on unreliable intermediate outputs. In the post-execution stage, termination anomalies occur when the system incorrectly determines task completion or result validity.

1) *Task Specification Anomalies*: Task specification anomalies arise when user goals, task constraints, agent roles, or configurations are ambiguous, incomplete, or inconsistent. Pan et al. [31] show that many task-level failures originate from unclear task definitions, such as insufficiently specified prompts. Altmann et al. [32] further observe that poorly defined tasks may lead to chasing and blocking, even when each agent’s local behavior appears reasonable. SentinelAgent [33] reports that insufficiently specified collaboration modes may cause agents to deviate from intended behaviors, collude, or become vulnerable to prompt injection. Therefore, evaluating task completeness before execution and enabling reflection during execution are important for robust multi-agent operation.

Agent configuration is also an essential component of task specification, including prompts, roles, skills, and tools. Pan et al. [31] identify incorrect role configuration as a common source of anomalies. Platon et al. [34] show that unclear configurations and role confusion frequently lead to coordination failures and adversarial misalignment. Similarly, OG [35] argues that actions outside an agent’s designated responsibility can directly cause conflicts, inconsistencies, and inefficiency, while AgentFM [36] identifies vague role configuration as a common failure source for database agents.

2) *Orchestration Anomalies*: Orchestration anomalies refer to failures in the global planning process of agent systems. Modern LLM-based multi-agent systems usually involve task decomposition, role assignment, tool and skill selection, inter-agent coordination, and workflow control [37], [38], [39]. Errors at this stage may propagate to later execution even before any individual subagent takes action. Typical orchestration anomalies include missing or redundant subtasks, mismatched agent–tool allocation, circular or conflicting dependencies, inefficient workflows, and plans that violate task constraints or security policies. These anomalies are challenging because they are often not attributable to a single faulty agent, but instead emerge from improper global coordination, incomplete task understanding, or unstable planning decisions.

3) *Communication Anomalies*: Communication anomalies occur during message exchange among agents. Bronsdon [40] highlights message storms as a typical failure mode, where excessive inter-agent communication causes resource exhaustion, increased latency, and eventual task failure. AgentPrune [41] similarly observes that redundant messages do not necessarily improve system performance; instead, excessive communication may distract agents and reduce coordination efficiency. Therefore, effective communication control is essential for scalable and reliable multi-agent systems.

4) *Termination Anomalies*: Termination anomalies occur when an agent system incorrectly determines whether a task should stop or continue. Premature termination is a representative case, where the system stops before completing the required reasoning or execution steps. Pan et al. [31] and Microsoft [42] both regard premature stopping as an important category of agent-system failures. Smurfs [43] shows that in single-agent settings, DFSDT may invoke the termination tool too early, thereby harming the completeness and logical coherence of complex tasks. Zhang et al. [12] further identify premature termination as a common and locatable failure source in multi-agent systems.

In contrast, agents may also fail to terminate. Zhu et al. [44] identify undercommitment anomalies, where an agent repeatedly delegates tasks to subagents, forming an infinite recursion chain that eventually reaches recursion-depth limits or timeouts. Drake [45] describes a related phenomenon, termed “neural howlround,” in which agents become trapped in recursive self-optimization or reasoning loops without a clear endpoint. Such non-termination anomalies may stall the system and lead to severe inefficiency or cognitive stagnation.

TABLE I
DEFINITION OF SOME CONCEPTS IN THE PAPER.

Name	Definition
DevOps [46]	DevOps merges software development and IT operations to shorten the development lifecycle and deliver high-quality software.
AIOps [47]	AIOps uses machine learning to automate IT operations.
LLMOps [48]	LLMOps extends MLOps to support the deployment, monitoring, and optimization of LLM-based applications. AgentOps focuses on agent systems.
AgenticOps [49]	Agentic Operations (AgenticOps) refers to the use of agents for the maintenance of traditional systems.
AgentOps	AgentOps refers to the use of various operational techniques for the maintenance of agent systems.

III. WHAT IS AGENTOPS?

To start, Table I presents key concepts that will be referenced throughout the discussion.

A. Evolution of Operations

As shown in Fig. 4, the evolution of operations can be understood from two dimensions: the evolution of operational technologies and the evolution of operational objects. Along the technology dimension, operations have advanced from manual intervention and rule-based automation to platform-based observability, intelligent analysis, and, more recently, autonomous operations enabled by LLM agents. Along the object dimension, the target of operations has expanded from traditional IT infrastructure and cloud-native applications to AI/ML systems and emerging agent systems.

Different operational paradigms correspond to different stages in this two-dimensional evolution. Site Reliability Engineering (SRE) focuses on ensuring the reliability of applications and infrastructure through engineering practices such as Service Level Objective (SLO) management, monitoring, alerting, and change control. AIOps further introduces intelligent techniques to maintain the stability of diverse systems, including anomaly detection, root cause analysis, alert noise reduction, and predictive optimization. As AI/ML systems became increasingly complex, MLOps emerged to manage the lifecycle and reliability of data pipelines, model training, deployment, inference, and drift monitoring.

Recently, LLM-based agents have become a new class of operational objects. Unlike traditional systems, whose behaviors are largely deterministic and specified by program logic, agent systems rely on probabilistic foundation models, dynamic tool invocation, memory, and multi-step reasoning. Their failures may arise not only from infrastructure faults or model degradation, but also from reasoning errors, unsafe actions, tool misuse, and misalignment with user or system goals. These characteristics make it insufficient to directly apply existing operational techniques to agent systems.

This insufficiency can be illustrated by two representative failures. As shown in Fig. 3, in a medical scheduling agent, the user asks to reschedule an appointment while keeping

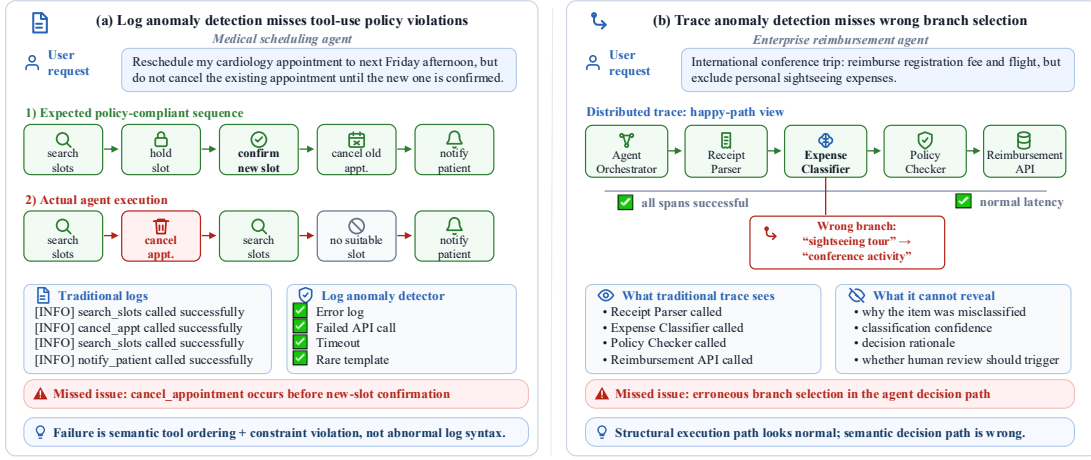


Fig. 3. Examples of applying traditional operations methods to agent systems. Traditional log-based anomaly detection (left) and trace-based anomaly detection (right) fail to detect anomalies in agent systems.

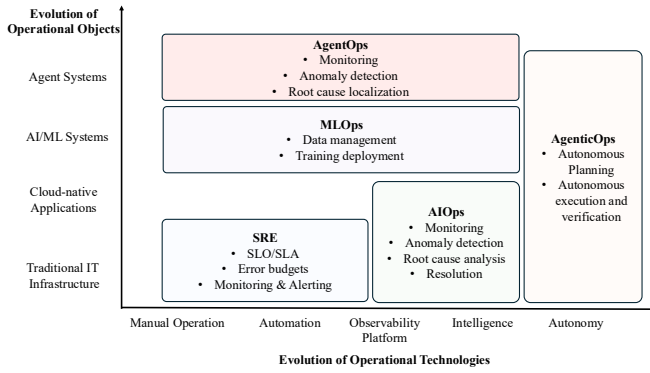


Fig. 4. Evolution of operational objects and operational technologies over time. SLO denotes Service Level Objective, while SLA denotes Service Level Agreement.

the original one until a new slot is confirmed. Although the correct trajectory should call `cancel_appointment` only after `confirm_reschedule`, the agent may prematurely cancel the existing appointment and later fail to find a new slot. Traditional log anomaly detection may not flag this behavior, since all tool calls can return successful status codes. The failure is instead a semantic tool-use policy violation: the agent violates the required ordering constraint. Similarly, in an enterprise reimbursement agent, a conventional trace may show a normal path from receipt parsing to expense classification, policy checking, and reimbursement submission, with all spans completing successfully. However, the agent may classify a “sightseeing tour” as a conference activity and reimburse it. The trace captures the structural service-call path, but not the semantic decision path behind the incorrect classification. These examples show that agent failures may be trajectory-level and semantic, involving intent, constraints, tool-use order, and decision rationale, rather than conventional anomalies in logs, status codes, latency, or traces.

Therefore, we propose **AgentOps**, which refers to operational techniques for maintaining the reliability, safety, and

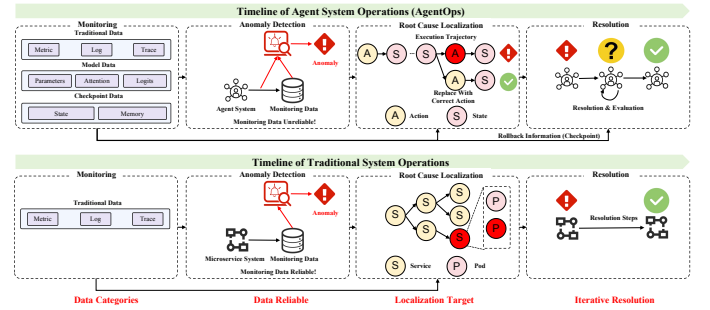


Fig. 5. Comparison of traditional system operations and agent system operations across four key stages.

controllability of agent systems. **AgentOps** is distinct from **AgenticOps**: the former operates *on* agent systems, whereas the latter uses agent systems as operational actors to maintain other systems. This distinction motivates the need for a dedicated operational framework for agent systems.

B. Difference between Traditional Operations and AgentOps

As shown in Fig. 5, although traditional system operations and agent system operations follow a similar timeline, they differ substantially at each stage. Table II further provides a task-level mapping of reliability practices in DevOps and AgentOps. These differences motivate the need for a dedicated operations framework, **AgentOps**, as detailed below.

- **Monitoring:** The primary difference lies in the monitored data. Traditional system operations, typically based on OpenTelemetry [50], focus on metrics, logs, and traces that reflect system runtime states. Agent systems require additional monitoring because their services are driven by inherently stochastic LLM agents. For the LLM component, relevant signals include model parameters, attention maps, token logits, and other internal states; for the agent component, checkpoints such as memory and environment states should be recorded at each step. Such agent-level monitoring

TABLE II
TASK-LEVEL MAPPING OF RELIABILITY PRACTICES IN DEVOPS AND AGENTOPS.

Stage	Task	DevOps	AgentOps
Anomaly detection	Process validation	Detects sequential breaks in text logs.	Detects uncertainty or hallucination signals in outputs.
	Health monitoring	Finds anomalies in infrastructure metric streams.	Finds loops, abnormal actions, or trajectory deviations.
	Output validation	Checks formats and deterministic thresholds.	Uses LLM-as-a-judge or fact-checking for semantics.
Root cause localization	Causal analysis	Localizes faults through static dependency graphs.	Localizes flawed actions or reasoning steps via trajectories.
Resolution	System recovery	Executes predefined runbooks without re-evaluation.	Uses self-correction loops and prompt refinement.
	State rollback	Rolls back static database snapshots.	Restores agent state to previous stable checkpoints.

improves system observability and enables rollback, providing stronger operational flexibility than traditional systems.

- **Anomaly Detection:** The main difference lies in where anomalies may originate. In traditional systems, traces, logs, and metrics are generally generated by deterministic or rule-based mechanisms and can therefore serve directly as reliable inputs for anomaly detection. In agent systems, however, monitoring data may be produced or mediated by probabilistic LLM-based agents, whose reasoning and generation processes can themselves introduce anomalies. Consequently, anomaly detection must examine not only the observed monitoring data but also the underlying generation process. This motivates the use of model-level signals, such as attention maps and logits, to detect reasoning or generation anomalies, as discussed in section V.
- **Root Cause Localization:** The primary distinction lies in the localization space and diagnosis target. Traditional operations localize faults over microservice dependency graphs, targeting infrastructure-level entities such as pods, services, or nodes. In contrast, agent systems localize failures along agent execution trajectories, where individual agent actions become the diagnosis targets. Accordingly, root cause localization in agent systems is also commonly referred to as failure attribution [12].
- **Resolution:** The primary distinction lies in the resolution schema. In traditional systems, once the fault location and root cause are identified, remediation can often follow deterministic procedures based on domain knowledge. In agent systems, inherent stochasticity makes resolution more complex, often requiring iterative testing, refinement, and rollback based on monitoring data. Moreover, traditional operations primarily adopt post-execution remediation, whereas AgentOps increasingly emphasizes intervention before and during execution. Because LLM-based agent systems may fail frequently and unpredictably, relying solely on reactive remediation can impose substantial operational overhead. Therefore, emerging approaches aim to prevent failures

before execution or correct them in a timely manner during execution.

C. Difference from Previous AgentOps Work

Existing literature on AgentOps generally approaches the domain through the lenses of observability, architectural adaptation, and role-based automation. For instance, Dong et al. [51] focus on establishing tracking and tracing pipelines for agent artifacts to ensure system observability. Meanwhile, Biswas et al. [52] and Moshkovich et al. [53] focus on the dynamic nature of agents, proposing frameworks, such as CHANGE and the AgentOps Automation Pipeline, to manage continuous learning, multi-agent consensus, and role-specific optimization.

In contrast to prior work, this study conceptualizes AgentOps as a structured lifecycle for failure diagnostics and recovery. The distinction is twofold: (1) We shift the focus from traditional metric-based tracing to a taxonomy rooted in agent structural boundaries, encompassing both intra-agent and inter-agent dimensions. (2) Bridging the gap between high-level recommendations and low-level control, our framework introduces concrete execution-level interventions; by utilizing checkpointing and state rollbacks, it effectively addresses the stochastic nature of agent execution paths.

D. Rationale for the Four-Stage Design

Inspired by the differences discussed above, we define **Agent System Operations (AgentOps)** as a comprehensive operational framework that spans the pre-execution, execution, and post-execution stages of agent systems. Similar to traditional system operations, AgentOps consists of four key phases: monitoring, anomaly detection, root cause localization, and resolution. Fig. 6 illustrates these phases through a customer refund processing example, where multiple agents collaboratively decompose a refund request, check refund eligibility, invoke order/payment/refund APIs, and verify the final refund status. When a schema change in the Policy Agent propagates downstream and causes failures in the Tool Agent and Verifier Agent, AgentOps first detects the abnormal patterns from metrics, logs, and traces, then localizes the true upstream root cause, and finally restores the workflow by rolling back to a healthy checkpoint and replaying the affected steps. We introduce each phase in the following sections.

To test whether each phase of AgentOps is useful, we create small workflows that resemble familiar online services. Consider a refund task: an automated program receives a refund request, reads the refund policy and two order records, and then uses tools to select and update the requested order. This tool-using program is the agent in our experiment. In a faulty run, we deliberately alter one decision—for example, the program selects the other order, skips the policy check, or uses an outdated observation. These mistakes represent reasoning or action anomalies and may affect later operations. Moreover, every order update leaves a permanent audit record. The program may subsequently correct the final order status, but it cannot erase the fact that it previously modified the wrong order. We therefore evaluate both whether the final result

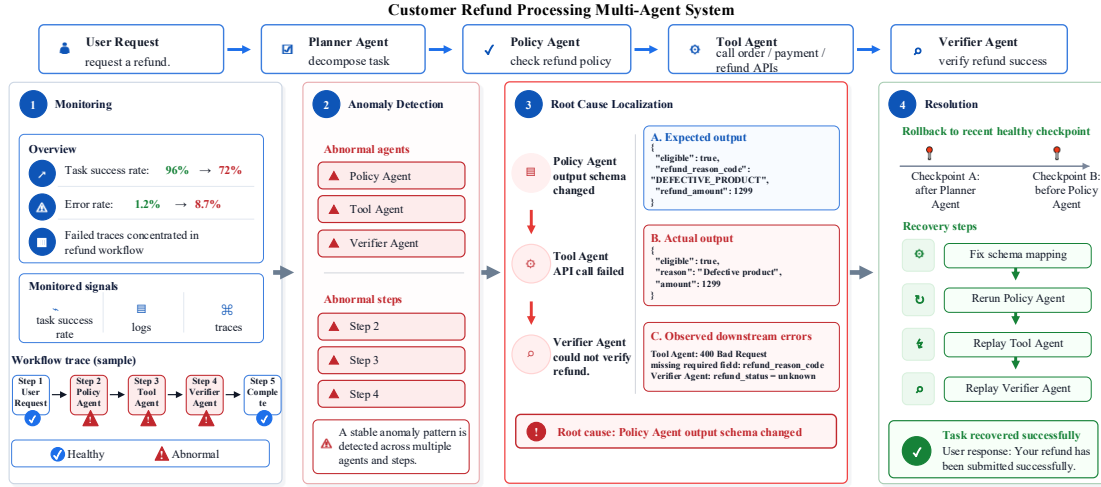


Fig. 6. Illustration of AgentOps for agent systems. AgentOps first detects anomalies and identifies the responsible steps and agents within the agent system, then performs root cause localization to pinpoint the underlying causes, and finally leverages checkpoint-based rollback for recovery.

TABLE III
CUMULATIVE EVALUATION OF THE AGENTOPS PHASES OVER 96 PAIRED SAMPLES. PASS AND SAFETY VALUES ARE PERCENTAGES; STEPS IS THE MEAN NUMBER OF TOOL STEPS.

Setting	Composite	Functional	Safety	Steps
Naive	25.0	25.0	50.0	8.00
+ Resolution	50.0	99.0	50.0	16.11
+ Anomaly detection	85.4	97.9	87.5	12.48
+ Root cause localization	87.5	100.0	87.5	11.24

is correct (*functional correctness*) and whether the workflow avoided unsafe operations (*operational safety*); *composite pass* requires both conditions.

The benchmark contains 32 tasks across configuration management, order processing, incident recovery, and data publication (similar to Fig. 6). Each task is repeated three times, producing 96 paired samples. We compare four cumulative settings: (1) a naive agent receives no intervention; (2) resolution is triggered only after the final evaluator reports a binary failure; (3) anomaly detection monitors the complete trajectory and interrupts execution when a high-confidence anomaly is observed; and (4) root cause localization further identifies the earliest causal step and provides this evidence to the resolver. All settings reuse the same trajectory prefix, environment state, and original agent. The evaluator never reveals the expected state or a failure diff. We use `deepseek-v4-flash` for the agent and operational modules, with thinking mode disabled. Monitoring includes assistant messages, tool calls, tool observations, errors, and state snapshots.

Table III demonstrates the distinct contribution of each phase. Monitoring is a passive prerequisite rather than a repair mechanism: it preserves both the agent’s stated rationale and the resulting system behavior, enabling semantic inconsistencies to be distinguished from tool failures. With this evidence, anomaly detection identified all 72 faulty trajectories and did not intervene in any of the 24 clean trajectories in the controlled benchmark. Because monitoring is shared by all set-

tings, however, its marginal contribution is not independently estimated here.

Resolution alone doubled composite pass from 25.0% to 50.0% and increased functional correctness to 99.0% (exact McNemar $p = 1.19 \times 10^{-7}$). Safety nevertheless remained at 50.0%, showing that post-execution repair can correct state but cannot undo an irreversible side effect. Adding anomaly detection increased composite pass to 85.4% and safety to 87.5% ($p = 5.40 \times 10^{-9}$), while reducing the mean execution length by 3.64 steps relative to terminal resolution. The improvement arises because the agent is interrupted before an upstream anomaly propagates into downstream actions. Thus, anomaly detection contributes not merely another feedback message, but the timing required to prevent error propagation.

Root cause localization yielded a smaller yet consistent incremental benefit. It repaired two additional samples without regressions and reduced execution by a further 1.24 steps. The pass-rate difference was not statistically significant ($p = 0.50$), so we do not claim a general reliability improvement. Nevertheless, the localizer identified the exact causal step in all 72 faulty trajectories, and the reduced recovery length indicates that localized evidence helped the agent avoid unnecessary diagnostic actions. Taken together, the results support the proposed division of responsibilities: monitoring provides evidence, anomaly detection determines when to intervene, root cause localization narrows the repair target, and resolution executes the recovery. The remaining failures occurred when a harmful action was itself irreversible before the next monitoring checkpoint, highlighting a boundary of checkpoint-based operation rather than implying complete fault coverage.

IV. HOW TO MONITOR AGENT SYSTEMS?

A. Illustration of Monitoring Data

In traditional monitoring data, metrics, logs, and traces collected using OpenTelemetry [50] exist within both microservice systems and agent systems.

First, with respect to metrics, traditional microservice systems typically focus on system-level metrics and Application

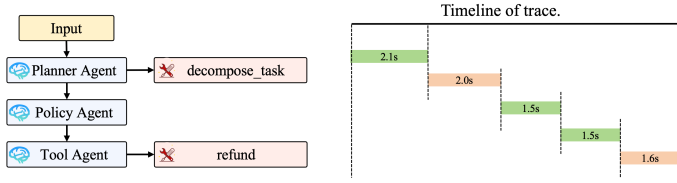


Fig. 7. Trace data of the agent system in Fig. 6.

Performance Monitoring (APM) metrics. In contrast, agent systems that incorporate LLM-based agents introduce additional observability requirements, particularly due to the use of LLMs, external tools, and retrieval components. Based on current agent-system applications, we summarize four commonly used categories of metrics: system metrics, APM metrics, cost-related metrics, and RAG-related metrics. It should be noted that these categories are not exhaustive. Instead, they are intended to capture representative and frequently adopted metric types in practical agent-system deployments.

Regarding traces, microservice traces typically capture interactions between services through API calls, whose interfaces and parameters are largely defined by fixed API schemas and application logic. In contrast, as illustrated by the customer refund scenario in Fig. 6, traces in agent systems need to capture not only the interaction paths among agents and tools, but also the dynamically generated inputs and outputs at each execution step. Fig. 7 further illustrates the trace data collected from this scenario, including the refund request, the intermediate output produced by the Policy Agent, the subsequent API call issued by the Tool Agent, and the verification result generated by the Verifier Agent. Since such intermediate contents are often generated or interpreted by LLMs, they may vary across executions and introduce additional uncertainty into the execution process. Therefore, these step-level inputs and outputs, together with agent-to-agent and agent-to-tool interactions, constitute essential trace data in agent systems and provide critical context for understanding and diagnosing their execution behavior.

When it comes to logs, as shown in Fig. 8, microservice systems and agent systems are similar. The logs in a microservice system record the overall behavior of services, whereas in an agent system, they capture the behavior of agents.

B. Monitoring Tools

Observability tools for LLM-based agent systems are rapidly evolving. Most of these tools follow the general observability paradigm of collecting metrics, logs, and traces, while further extending it with agent-specific functions such as dataset management, experiment tracking, evaluations, prompt optimization, and deployment management. Table IV compares the functionalities of representative tools. LangDB [54], the first observability tool implemented entirely in Rust, emphasizes runtime efficiency and integrates router optimization for cost control. LangFuse [55] supports OpenTelemetry integration and is among the most active open-source tools in this area. Helicone [57] combines observability with cache management to reduce latency and resource consumption,



Fig. 8. Log data of the agent system in Fig. 6.

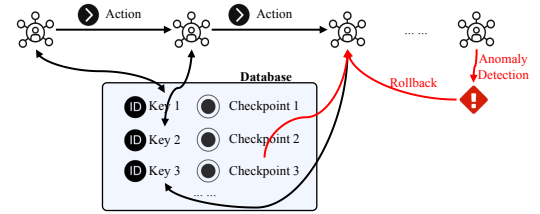


Fig. 9. Collection of checkpoint data and its use in rollback processes.

and improves reliability through mechanisms such as gateway fallback. HoneyHive [66] supports distributed tracing and multimodal systems, allowing users to define custom spans for fine-grained analysis. PromptLayer [67], originally designed for prompt optimization, incorporates observability features such as prompt ranking. TruLens [65], provided as a Python package, integrates with frameworks such as LlamaIndex and supports iterative optimization with human feedback. OpenLL-Metry [60] follows the OpenTelemetry standard and is compatible with multiple frameworks, but provides limited support for prompt optimization and evaluation. LangWatch [58] and Literal AI [62] provide standard observability, evaluation, and development functionalities, with LangWatch also available as an MCP server. MLFlow [56], originating from traditional machine learning workflows, supports custom metrics for agent systems. DeepEval [69] primarily focuses on evaluation rather than observability, whereas AgentOpsMonitor [68] emphasizes operational management in addition to observability. LangSmith [70] integrates tracing, evaluation, and dataset management, with particular strengths in debugging and optimizing LangChain-based agent workflows.

C. Future Directions

Current observability for agent systems is still largely designed for monitoring. However, effective observability should serve a broader purpose: it should support anomaly detection, root cause localization, and failure resolution throughout the

TABLE IV
COMPARISON OF REPRESENTATIVE OBSERVABILITY TOOLS. THE LISTED TOOLS ARE REPRESENTATIVE EXAMPLES, AND MANY UNLISTED TOOLS PROVIDE SIMILAR FUNCTIONALITIES.

Tools	Metric				Log	Trace
	System	Cost	RAG	Performance		
LangDB [54]	✓	✓	×	×	✓	✓
LangFuse [55]	✓	✓	✓	✓	✓	✓
MLFlow [56]	×	✓	×	✓	✓	✓
Helicone [57]	✓	✓	×	✓	✓	✓
LangWatch [58]	✓	✓	×	✓	✓	✓
LlamaTrace [59]	✓	✓	×	✓	✓	✓
OpenLLMetry [60]	✓	×	×	✓	✓	✓
Arize Phoenix [61]	✓	✓	×	✓	✓	✓
Literal AI [62]	✓	✓	×	✓	✓	✓
Opik [63]	✓	✓	×	✓	✓	✓
OpenInference [64]	✓	×	×	✓	✓	✓
TruLens [65]	✓	✓	×	✓	✓	✓
HoneyHive [66]	✓	✓	×	✓	✓	✓
PromptLayer [67]	✓	✓	×	✓	✓	✓
AgentOpsMonitor [68]	✓	✓	×	✓	✓	✓
DeepEval [69]	✓	✓	×	✓	✓	✓
LangSmith [70]	✓	✓	✓	✓	✓	✓

lifecycle of agent execution. As discussed earlier in section III-B, existing observable data in agent systems, such as metrics, logs, and traces, is often unreliable, incomplete, or semantically insufficient. As a result, these signals may not even provide adequate support for comprehensive anomaly detection, let alone precise localization and automated recovery.

Therefore, one promising direction is to enrich the types of observable data available to agent systems. In particular, model data and checkpoint data can serve as two important extensions to existing observability signals. As data security concerns continue to grow, an increasing number of online agent systems are adopting locally deployed open-source LLMs as inference engines [71]. This deployment paradigm enables agent systems to move beyond black-box observation, where only inputs and outputs are available, and instead collect white-box model data from internal states, such as hidden-layer representations and token logits [72], [73]. Compared with input-output records alone, such model-level data can provide more fine-grained evidence for detecting abnormal behaviors that originate within the LLM itself, thereby improving the observability of agent reasoning and generation processes.

Checkpoint data provides another complementary type of observable data. Compared with traditional microservice systems, agent systems often provide greater controllability, since their execution state at a given moment can be reproduced from recorded data. Therefore, checkpoints can periodically capture the execution state of an agent system, including memory, environment, intermediate results, and other runtime information. As shown in Fig. 9, once an anomaly is detected, the system can roll back to a previously safe checkpoint, incorporate feedback, and resume execution from a corrected state. In coding-agent systems, repository revisions provide a concrete realization of such checkpoint-aware observability. For example, ARC [74] uses Git revision history to record intermediate repository states, agent modifications, and associated test outcomes during test-driven software construction, providing an auditable execution history for subsequent diagnosis and recovery.

In addition to expanding the categories of observable data,

future observability should also improve the quality of such data. Rather than relying solely on raw metrics, logs, traces, model data, or checkpoint records, agent systems require processed and risk-aware signals that more directly reflect potential failures. For example, hallucination probability inferred from execution traces or token-level uncertainty estimated from logits can provide more actionable evidence than raw records alone. Such high-quality observability signals can help agent frameworks detect, localize, and recover from failures more efficiently, enabling agent systems to remain stable, reliable, and continuously improvable. Another promising direction is to transform raw observations into specification-grounded semantic signals. For example, WebTestPilot [75] represents GUI states with symbolic variables and infers step-wise preconditions and postconditions from natural-language specifications, providing executable semantic oracles for validating multi-step executions.

V. HOW TO DETECT ANOMALIES IN AGENT SYSTEMS?

After obtaining sufficiently diverse and comprehensive monitoring data through multi-dimensional observation, these high-quality data can be utilized to perform operations on the agent system. Among these operations, anomaly detection plays a crucial role, as promptly identifying anomalies within the system is essential for effectively guiding subsequent root cause localization and resolution. As illustrated in Fig. 6, anomaly detection transforms the monitored degradation and abnormal execution traces into actionable signals by identifying the Policy Agent, Tool Agent, and Verifier Agent, together with their corresponding abnormal steps. This narrows the scope of subsequent diagnosis before the underlying cause is further investigated. This section primarily focuses on the methods of anomaly detection. A summary of anomaly detection and mitigation in agent systems is presented in Table V. Below, we primarily introduce detection methods for some common anomalies.

A. Detection Methods

1) *Detecting Reasoning Anomalies*: The methods for detecting and mitigating reasoning anomalies can be categorized into white-box, grey-box, and black-box based on the type of input information used. White-box approaches utilize the token sequence, the associated probabilities of each token, and the model parameters. In contrast, grey-box methods do not use model parameters. Lastly, black-box techniques rely solely on the token sequence.

- **White-box**. SAPLMA [73] posits that LLM parameters contain implicit clues of hallucinations and proposes a parameter-based classifier, trained on a large and diverse dataset, to detect hallucinations in a zero-shot manner. In contrast, OPERA [72] addresses hallucinations during the inference phase by identifying the “partial over-trust” phenomenon in self-attention maps, mitigating it through a column-wise metric, probabilistic penalties, and a rollback strategy. Meanwhile, Honesty [19] attributes hallucinations to queries beyond the model’s knowledge boundaries, advocating for a model that is honest about its limitations by introducing evolutionary metrics and an IDK (“I don’t know”)

TABLE V

A SUMMARY OF ANOMALY DETECTION AND MITIGATION METHODS FOR AGENT SYSTEMS. HERE, MITIGATION DENOTES TARGETED MECHANISMS OR AUXILIARY MODELS DESIGNED TO ADDRESS SPECIFIC ANOMALY CATEGORIES, WHEREAS SECTION VII DISCUSSES ANOMALY RESOLUTION FROM THE PERSPECTIVE OF THE OVERALL AGENTOPS FRAMEWORK.

Anomaly Level	Anomaly Type	Method Category	Method	Input	Output	Detection	Mitigation				
Intra-Agent	Reasoning Anomalies	White-box	SAPLMA [73]	LLM Parameters	Anomaly Probability	✓	✗				
			OPERA [72]	Attention Map	Penalty-regenerated Response	✓	✓				
			Honesty [19]	LLM Parameters (Finetuning)	Revised Response	✓	✓				
		Grey-box	LURE [76]	Token Logits, Revisor Model	Revised Response	✓	✓				
			Conformal [77]	Token Logits	High-quality Response	✓	✓				
		Black-box	Debate [78]	Multiple Agents	Debated Response	✓	✓				
	CoK [79]		Multiple Data Sources	High-quality Response	✓	✓					
	Action Anomalies	MCP	AI-Infra-Guard [22] MCP Guardian [80]	AI Component AI Component	MCP Risks MCP Risks	✓ ✓	✗ ✗				
	Memory Anomalies	Short-term Memory	PI [81] CoA [82]	Long Context Long Context	Response Response	✗ ✗	✓ ✓				
						Long-term Memory / RAG	ReDeep [27] LRP4RAG [83]	Attention, LLM Parameters Token Logits	Hallucination Results Anomaly Probability	✓ ✓	✓ ✗
Security Anomalies		Graph-based	GUARDIAN [84] SentinelAgent [33]	Agent Graph Agent Graph	Anomalous Position Anomalous Position		✓ ✓	✗ ✗			
Task Specification Anomalies	Classifier-based	SpecValidator [85] Ambig-SWE [86] CLAMBER [87]	Task Description Task Instruction User Query	Defect Type / Probability Underspecification Label / Clarification Questions Ambiguity Label / Clarifying Question	✓ ✓ ✓	✗ ✓ ✓					
					Uncertainty-based	Ask-or-Assume [88] Semantic Entropy [89] SelfCheckGPT [90]	Task Instruction, Agent State Sampled Interpretations Sampled Responses	Ask-or-Execute Decision Semantic Uncertainty Score Consistency Score	✓ ✓ ✓	✓ ✗ ✗	
									Orchestration Anomalies	Single-step	Introspective [91] API-bank [92] ReAct [14]
	Multi-step	ToolLLM [4] Reflexion [93] CodeAct [94]	LLM Parameters (Finetuning) External Feedback Tool Description	High-quality Plan High-quality Plan Code							
					Communication Anomalies	Redundancy	AgentPrune [41] G-Designer [95]	Agent Graph Agent Graph		Optimized Agent Graph Optimized Agent Graph	✓ ✓

Note. Termination anomalies are not included in this table because they are typically easy to observe or can be detected through rule-based methods, such as checking task success rates or the number of task execution steps.

response, alongside techniques like prompt engineering and supervised fine-tuning to resolve such anomalies.

- **Grey-box.** LURE [76] analyzes three primary causes of hallucinations—co-occurrence, uncertainty, and object position—and uses LLMs to generate samples for training a lightweight revisor that can directly detect and correct responses during the inference phase. Conformal [77], inspired by conformal prediction, calculates the quality of sampled responses based on token sequence logits and discards them according to a stopping rule.
- **Black-box.** Debate [78] reduces hallucinations by having multiple model instances iteratively evaluate and update their answers to the same question, drawing on the perspective of The Society of Mind. CoK [79] enhances factual accuracy by enabling the LLM to actively query and integrate information from multiple external knowledge sources during inference, iteratively reviewing and correcting the reasoning chain.

2) *Detecting Action Anomalies:* Action anomalies are primarily categorized into function call anomalies and Model Context Protocol (MCP) anomalies. Function call anomalies are relatively straightforward to address due to their clear error messages and direct handling, so they are not elaborated here. In contrast, for more complex MCP anomalies, AI-Infra-Guard [22] employs agent-based techniques to detect potential risks within MCP, such as tool poisoning attacks. The core mechanism of MCP Guardian [80] involves embedding a “security-first” middleware layer within MCP communication, which provides four key functions: authentication, rate limiting, detailed logging of call information, and deep inspection of packet contents, thereby enhancing the communication

between the client (MCP client) and the data source/tool server (MCP server).

3) *Detecting Memory Anomalies:* Since an agent’s memory is divided into short-term memory and long-term memory, memory anomalies can be categorized into short-term memory anomalies and long-term memory anomalies.

- **Short-Term Memory.** To address the challenge of extending the context window length, PI [81] proposes position interpolation, which scales the entire window through position encoding. In contrast, CoA [82] adopts a multi-agent collaborative framework, where different agents handle separate chunks and a managing agent integrates the results, mitigating the “lost in the middle” issue commonly observed in traditional fine-tuning methods.
- **Long-Term Memory.** ReDeep [27] mitigates hallucinations by computing two scores—an external context score and a parametric knowledge score—to modulate the contributions of Knowledge FFNs and Copying Heads within the residual stream. LRP4RAG [83], on the other hand, employs the Layer-wise Relevance Propagation (LRP) algorithm, performing LRP backward after the generator outputs logits to obtain a relevance matrix, which is then fed into a pre-trained classifier to determine hallucinations.

4) *Detecting Security Anomalies:* Security anomalies pose significant threats to system security, and detection methods often rely on graph-based approaches. GUARDIAN [84] highlights that traditional multi-agent voting approaches overlook dependencies between agents. To address this, it models the agent system as a graph, using an encoder-decoder architecture to compress and reconstruct the entire graph, with anomalies detected based on the reconstruction score. SentinelAgent [33],

on the other hand, tackles multi-agent coordination risks by proposing a three-layer hierarchical detection method that operates from global to detailed levels.

5) *Detecting Task Specification Anomalies*: Task specification anomalies arise when user instructions are ambiguous, incomplete, or underspecified, making it difficult for an agent to correctly infer the intended goals, constraints, or execution requirements. Unlike most execution-time anomalies, such failures may already be present before the agent begins acting and can subsequently propagate into planning and execution. Existing detection methods can be broadly categorized into *classifier-based* and *uncertainty-based* approaches.

- **Classifier-based.** Classifier-based methods directly assess whether a task description contains specification defects. SpecValidator [85] detects defective task descriptions and predicts the corresponding defect type or probability. Ambig-SWE [86] focuses on underspecified instructions in software-engineering tasks; in addition to identifying underspecification, it generates clarification questions to acquire missing information before execution. Similarly, CLAMBER [87] studies ambiguous information needs in user queries and enables models to both recognize ambiguity and request clarification. These methods explicitly model task-specification quality and are therefore suitable for intercepting malformed or incomplete requests before they propagate to downstream agent behaviors.
- **Uncertainty-based.** Uncertainty-based methods, in contrast, infer specification anomalies from the uncertainty or inconsistency induced by an instruction. Ask-or-Assume [88] jointly considers the task instruction and the current agent state to determine whether the agent should proceed with execution or first ask for clarification. More general uncertainty-estimation methods can also provide useful signals: Semantic Entropy [89] measures semantic uncertainty across sampled interpretations, while SelfCheckGPT [90] evaluates consistency among sampled responses. Such methods are particularly useful when ambiguity cannot be captured by predefined defect categories, as an underspecified task may admit multiple plausible interpretations and consequently produce high uncertainty or inconsistent agent behaviors.

6) *Detecting Orchestration Anomalies*: Orchestration anomalies can be categorized based on their occurrence location into single-step and multiple-step anomalies.

- **Single-step.** Introspective [91] combines RAG recall with human feedback-based Conformal Prediction to address common-sense reasoning issues in LLMs. API-bank [92] improves LLM planning capabilities by finetuning the model with dialogue data collected from API calls, using a data-driven approach. ReAct [14], on the other hand, introduces a reasoning and acting framework that splits planning into two stages—reasoning and acting—and implements it through prompt engineering, providing the model with opportunities for reflective error correction.
- **Multiple-step.** ToolLLM [4] generates both single-tool and multi-tool instructions based on APIs and further proposes a novel depth-first search-based decision tree to construct

coherent planning paths. Reflexion [93] leverages external feedback—such as simple binary environment feedback, predefined heuristics for common failure cases, and self-evaluation mechanisms like binary classification using LLMs (for decision-making) or self-written unit tests (for programming)—to enable agents to reflect on past execution steps and optimize subsequent trajectories. CodeAct, in turn, formulates the entire planning process as executable Python code, fundamentally reducing hallucinations in tool usage by ensuring precise parameter passing.

7) *Detecting Communication Anomalies*: Communication anomalies primarily arise from redundant messages caused by the scaling of agent systems. To address this issue, AgentPrune [41] models the system as a graph and trains a low-rank-principle-guided graph mask to focus on key communication areas, thereby mitigating communication redundancy and improving efficiency. G-Designer [95], from a more macro perspective, argues that the optimal communication topology varies depending on the task. To this end, a graph auto-encoder compresses the entire agent system and reconstructs an optimal topology.

The primary cause of communication anomalies is the continuous expansion of agent systems, leading to an increasing volume of transmitted messages. However, many of these messages are unnecessary, and excessive redundancy can result in task failure.

B. Future Directions

Although existing studies have proposed various detection methods for reasoning, action, memory, orchestration, and communication anomalies, most of them remain category-specific, relying on distinct input signals, detection granularities, and modeling assumptions. Therefore, an important future direction is to move from anomaly-specific detectors toward unified and generalizable detection frameworks. Such frameworks should integrate model-internal states, reasoning traces, tool-call records, memory access patterns, and agent interaction structures, enabling different anomaly patterns to be characterized within a shared representation space. This would improve the applicability of anomaly detection methods across diverse tasks, architectures, and deployment environments.

Moreover, future research should further bridge the intra-agent and inter-agent levels. Existing methods typically focus either on anomalies within individual agents or on anomalies emerging from multi-agent collaboration. However, in real-world agent systems, reasoning, memory, or action anomalies at the individual-agent level may further manifest as orchestration failures or communication anomalies at the system level. Cross-level detection methods should therefore jointly model the internal states of individual agents and the interaction structures among multiple agents, so as to identify local abnormal signals at an early stage and capture their system-level manifestations.

Finally, online anomaly detection for long-horizon tasks represents another key direction. Practical agent systems often operate continuously in dynamic environments, where anomalies may gradually accumulate through iterative planning,

TABLE VI
SUMMARY OF ROOT CAUSE LOCALIZATION METHODS IN AGENT SYSTEMS.

Category	Attribution Basis and Methods
Replay & Spectrum	Success/failure trajectories; state–action–agent tuples; statistical suspiciousness. <i>Methods:</i> FAMAS [96]
Structured Trace	Structured traces, communication logs, and causal/dependency graphs; information-flow and propagation analysis. <i>Methods:</i> GraphTracer [97]; AgenTracer [98]
LLM Attribution	Natural-language logs, segmented trajectories, and failure taxonomies; semantic judgment and taxonomy-guided reasoning. <i>Methods:</i> Who&When [12]; AgentFail [99]

tool use, memory updates, and communication processes. Compared with post-hoc detection based on final outputs or offline logs, online detection enables continuous monitoring of agent behaviors during execution, allowing potential abnormal trends to be identified in a timely manner and providing more fine-grained signals for subsequent root-cause analysis and anomaly resolution.

VI. HOW TO LOCATE THE ROOT CAUSE OF AGENT SYSTEM ANOMALIES?

A. Definition of Root Cause Localization in AgentOps

In the context of AgentOps, root cause localization—also referred to as failure attribution—aims to identify which component causes an anomaly and at which point in the execution it first occurs, by analyzing the execution trajectory of an agent system. Continuing the example in Fig. 6, anomaly detection flags multiple abnormal agents and steps, whereas root cause localization further distinguishes the upstream cause from its downstream manifestations: the schema change in the Policy Agent is identified as the root cause of the subsequent Tool Agent API failure and Verifier Agent failure. In this work, root cause localization primarily focuses on anomalies that arise during the execution phase.

B. Root Cause Localization Methods

1) *Trajectory Replay & Spectrum Analysis*: This category quantifies the “suspiciousness” of actions or agents by comparing the differences between successful and failed trajectories. FAMAS [96] introduces a spectral-analysis framework based on trajectory abstraction, in which execution logs are decomposed into state–action–agent tuples. It further defines a hybrid scoring function that jointly considers failure frequency, internal repetitiveness, and action centrality. By ranking suspiciousness scores, the system can automatically identify the agents and key actions most likely responsible for the failure.

2) *Structured Trace Modeling*: As system complexity increases, researchers have proposed structured trajectory modeling methods to capture long-term and cross-agent dependencies. GraphTracer [97] constructs an information dependency graph to explicitly model inter-agent information flow and identifies root nodes causing downstream errors via backward tracing. AgenTracer [98] generates synthetic failure

trajectories through counterfactual replay and fault injection, training a small dedicated LLM to learn the who and when localization tasks. Ma et al. [100] construct structured causal graphs and apply counterfactual reasoning to attribute system-level failures to specific agents and actions. SentinelAgent [33] models inter-agent interactions and execution dynamics as graphs to capture anomalous coordination and behavioral patterns. More recently, StepFinder [101] models execution traces as temporal semantic sequences. It uses LLMs only for semantic feature construction and subsequently applies temporal modeling and attention mechanisms to capture sequential evolution and cross-step dependencies, enabling efficient step-level root cause localization.

3) *LLM-based Attribution*: Another mainstream direction is to leverage large language models to perform “log adjudication” (LLM-as-a-judge). Zhang et al. [12] propose three inference paradigms: **All-at-once**, where the full failure log is provided and the LLM outputs the faulty agent and step in a single pass; **Step-by-step**, where log segments are fed incrementally and the LLM decides whether the current step is erroneous, terminating immediately upon detection; and **Binary Search**, which recursively bisects the log interval and uses iterative LLM judgments to progressively narrow down the error span. These strategies exploit LLMs’ semantic reasoning ability, enabling soft, inference-driven localization even in the absence of highly structured logs. Although experiments show that pure LLM-based judgment still falls short in step-level accuracy, its advantage lies in supporting semi-automated diagnostic workflows when combined with human oversight, providing interpretable, language-based insights that facilitate system debugging.

AgentFail [99] further observes that LLMs struggle to reason effectively over complex trajectories. To address the difficulty LLMs face in accurately diagnosing the root causes of complex failures, AgentFail proposes a fine-grained taxonomy of agent system failures, which is used to guide the structured knowledge representation of LLMs. In the AgentFail benchmark, incorporating this taxonomy into LLM guidance improves localization identification accuracy by approximately 15%. AgentDebug [102] similarly argues that failures in agent systems are closely tied to failure taxonomies. By analyzing a large number of cases, it identifies that different types of failures exhibit distinct tendencies across different stages of an agent system’s execution. Accordingly, AgentDebug integrates taxonomy-aware analysis at each step of the agent system to jointly assess system behavior. Upon detecting a failure, the agent replans its trajectory, enabling timely resolution.

C. Future Directions

Existing root cause localization methods can be broadly categorized into two groups: LLM-based methods and non-LLM-based methods, including statistical analysis and trace-based models. These two categories exhibit complementary strengths. LLM-based methods leverage the strong semantic reasoning capabilities of LLMs, enabling fine-grained fault localization at both the agent and step levels. However, their effectiveness degrades as the execution trace becomes longer.

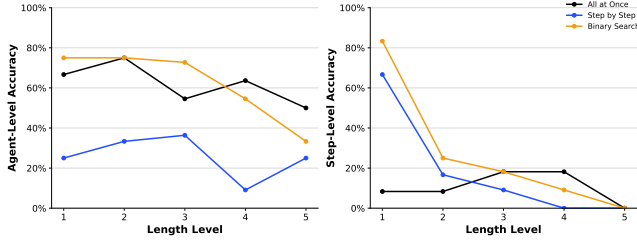


Fig. 10. Root cause localization accuracy of different methods on the Who&When dataset under varying context lengths (numbers of steps). Left: agent-level accuracy for identifying the erroneous agent; right: step-level accuracy for locating the erroneous step. All-at-once feeds the full trace to the LLM at once, Step-by-step processes one step at a time, and Binary-search locates the erroneous step via binary search over the full trace.

As shown in Fig. 10, where context length is measured by the number of steps in a trace, LLM-based methods become increasingly susceptible to the “lost-in-the-middle” problem, which can lead to hallucinated or incorrect attributions in long and complex agent trajectories. In contrast, non-LLM-based methods can learn statistical patterns and structural dependencies among agents and execution steps from historical traces, allowing them to efficiently identify suspicious regions even in long trajectories. Nevertheless, such methods typically provide coarse-grained predictions and lack the semantic reasoning capability required for precise root-cause localization, particularly at the step level.

A promising future direction is to integrate two types of methods in a two-stage framework. The first stage uses a non-LLM-based model to narrow down suspicious regions, while the second stage applies an LLM-as-a-Judge to these regions for fine-grained attribution.

VII. HOW TO RESOLVE THE ANOMALIES?

Once former steps have accurately pinpointed the root location, traditional operations can typically perform rapid recovery actions such as node switching or hardware maintenance to prevent further economic losses. However, the resolution process in agent systems differs substantially, both in its paradigms and methodologies. The following sections provide a detailed discussion of these distinctions.

A. Resolution Methods

To systematically resolve anomalies in complex agentic systems, we categorize resolution strategies according to the stage at which the intervention is applied: *pre-execution preventive resolutions*, *in-execution corrective resolutions*, and *post-execution recovery resolutions*. Rather than focusing on how anomalies are diagnosed, this taxonomy emphasizes how an agent system can actively resolve or mitigate failures. These three classes are complementary and are often combined in practical agent frameworks [14], [38], [103].

1) *Pre-execution Preventive Resolutions*: Pre-execution preventive resolutions aim to avoid anomalies before the agent commits to external actions. These strategies are especially useful when actions are costly, unsafe, or irreversible.

TABLE VII
TARGETED ANOMALY TYPES OF RESOLUTION METHODS. ✓ DENOTES A TARGET, AND – DENOTES AN INDIRECT TARGET.

Method	Intra-agent				Inter-agent			
	Reas.	Act.	Mem.	Sec.	Task	Comm.	Term.	Orch.
<i>Pre-execution preventive</i>								
Task Re-spec.	✓	✓	–	–	✓	–	✓	✓
Plan Verif.	✓	✓	–	–	–	–	–	✓
Action Restrict.	–	✓	–	✓	–	–	–	–
<i>In-execution corrective</i>								
Obs.-grounded Replan.	✓	✓	–	–	–	–	✓	✓
Self-correction	✓	✓	–	–	–	–	–	–
Redundant Exec. & Select.	✓	✓	–	–	–	–	–	✓
<i>Post-execution recovery</i>								
Rollback & Re-exec.	✓	✓	✓	✓	✓	–	✓	✓
Memory & Skill Update	✓	✓	✓	–	–	–	✓	✓
Policy & Prompt Rev.	✓	✓	–	✓	✓	–	✓	✓

Note. Reas.=Reasoning, Act.=Action, Mem.=Memory, Sec.=Security, Task=Task Specification, Comm.=Communication, Term.=Termination, Orch.=Orchestration.

- **Task Re-specification.** A common way to prevent agent failures is to rewrite or refine the task before execution. This includes clarifying ambiguous goals, decomposing complex objectives, adding explicit constraints, specifying success criteria, and converting open-ended requests into executable subtasks. Prompting methods such as Chain-of-Thought, ReAct, and Tree-of-Thoughts make intermediate reasoning or alternative plans explicit, thereby reducing the chance that the agent starts from an ill-formed plan [104], [14], [105].
- **Plan Verification.** Before execution, the agent can generate a candidate plan and check whether it is feasible, safe, and consistent with user constraints. Plan verification may involve rule-based validators, critic models, verifier agents, or human approval. This strategy prevents invalid action sequences before they are carried out. Multi-agent systems such as AutoGen support planner–critic–executor patterns, where one agent proposes actions and another reviews them before execution [38].
- **Action Space Restriction.** Agent behavior can be made more reliable by restricting what actions are available before execution begins. This includes limiting tool access, enforcing typed tool schemas, requiring valid parameters, using sandboxed environments, and defining permission boundaries. Tool-use systems such as Toolformer and agent frameworks such as SWE-agent show that the design of tool interfaces and action spaces strongly affects the reliability of agent execution [106], [103].

2) *In-execution Corrective Resolutions*: In-execution corrective resolutions intervene while the agent is reasoning, calling tools, receiving observations, or updating its state. Their purpose is to correct errors before the execution trajectory becomes unrecoverable.

- **Observation-grounded Replanning.** During execution, new observations may contradict the original plan or reveal that the current trajectory is ineffective. The agent can resolve such deviations by revising its plan based on the latest environment feedback or tool outputs. ReAct is a representative framework that interleaves reasoning and

acting, enabling the agent to update its next action after each observation [14].

- **Self-correction.** Self-correction allows the agent to identify and repair its own mistakes during execution. The agent may critique its intermediate reasoning, verify tool outputs, rerun failed steps, or generate revised actions. Methods such as Self-Refine and Reflexion demonstrate that iterative feedback, critique, and reflection can improve model behavior without updating model parameters [107], [93].
- **Runtime Enforcement and Repair.** Runtime mechanisms can enforce structured constraints on tool calls, intermediate states, and model outputs to intercept unsafe or policy-violating actions before execution [108]. They can also detect and repair incompatibilities between probabilistic LLM outputs and deterministic software components, such as malformed formats, invalid syntax, or interface mismatches [109], thereby preventing failures from propagating through the system.
- **Redundant Execution and Selection.** Instead of committing to a single trajectory, the system may generate multiple candidate actions or run multiple agents in parallel, then select the most reliable output through voting, ranking, or verifier scoring. This resolves stochastic reasoning errors by comparing independent hypotheses. Such mechanisms are commonly used in multi-agent debate and ensemble-style agent execution [110], [78], [38].

3) *Post-execution Recovery Resolutions:* Post-execution recovery resolutions are applied after an execution attempt has failed or completed with unsatisfactory performance. These strategies focus on restoring a valid state, repairing consequences, and reducing recurrence in future runs.

- **Rollback and Re-execution.** If an agent reaches an invalid or undesirable state, the system can restore a previous checkpoint (such as using Git revision history [74]) and re-execute from that point with revised constraints or plans. This requires persistent state tracking, checkpointing, and replayable execution traces. Frameworks such as LangGraph support checkpointing and time-travel execution, which allow developers to resume or fork an agent run from earlier states [111].
- **Memory and Skill Update.** After a failed or successful run, the agent can store useful experience as memory, corrected procedures, or reusable skills. This allows future executions to avoid similar failures. Reflexion stores verbal feedback from prior attempts, while Voyager builds a skill library that can be reused and composed in later tasks [93], [112].
- **Policy and Prompt Revision.** When a failure reflects a recurring weakness, the system can revise prompts, policies, tool schemas, or orchestration rules after execution. This differs from runtime self-correction because the modification is persistent and affects future runs. Automatic prompt optimization methods and evolutionary prompt search provide systematic ways to update instructions based on task-level performance [113], [114], [115], [116].

TABLE VIII
COMPARISON OF DIFFERENT DATASETS AND BENCHMARKS. RC STANDS FOR ROOT CAUSE.

Dataset	#Fail.	#Types	Single RC	Scalability and Agent Systems
Who&When [12]	184	–	✓	Partial. <i>Systems:</i> 127 multi-agent LLM systems.
MASFT [117]	1642	14	✓	Yes. <i>Systems:</i> ChatDev, MetaGPT, HyperAgent, AppWorld, AG2, Magentic-One, OpenManus.
TRAIL [118]	841	21	✓	No. <i>Systems:</i> –.
AgentFail [99]	307	16	✓	Partial. <i>Systems:</i> Dify, Coze.
Aegis [119]	24843	14	✓	Yes. <i>Systems:</i> LLM Debate, MacNet, AgentVerse, Dylan, SmolAgents, Magentic-One.
Agent-Debug [102]	200	17	✓	No. <i>Systems:</i> ALFWorld, WebShop, GAIA.
MP-Bench [120]	289	–	×	No. <i>Systems:</i> MAgentic-One, CaptainAgent.
LongRCA Bench [121]	1140	–	✓	No. <i>Systems:</i> AutoGen-based and custom multi-agents.

B. Future Directions

We identify three promising directions for advancing resolution mechanisms in agent systems. A key direction is to develop a unified framework that integrates preventive, in-process, and post-hoc resolution mechanisms. Rather than treating these stages as isolated modules, future agent systems should adaptively select suitable resolution strategies according to the task context, execution state, uncertainty, and risk of error propagation. Such a framework can better balance robustness, efficiency, and correction cost.

Another important direction is to improve agents' self-correction ability during execution. Existing methods such as Reflexion enable feedback-driven reflection, but they often rely on external feedback and may detect errors only after several subsequent steps. This delay can allow mistakes to propagate, especially in long-horizon or multi-agent settings. Future work should therefore enhance both rapid error detection, allowing agents to identify mistakes within only a few steps, and self-driven detection, enabling agents to monitor their own reasoning and actions without relying on external signals.

Post-hoc resolution is also crucial for enabling agents to learn from practice. In existing frameworks, such as Hermes-style systems, this process is often formulated as skill accumulation, where reusable capabilities are distilled from past trajectories. While recent work explores adversarial or self-learning mechanisms for skill discovery, most frameworks emphasize successful experiences and overlook failures. Future systems should distill not only successful skills but also reusable lessons from failed trajectories, such as constraints, diagnostic patterns, and failure-aware heuristics, so as to reduce repeated errors and improve long-term robustness.

TABLE IX

COVERAGE OF EXISTING STUDIES UNDER OUR PROPOSED FAILURE TAXONOMY. ✓ DENOTES EXPLICIT COVERAGE AND – DENOTES LITTLE OR NO SUBSTANTIAL COVERAGE.

Study	Intra-agent				Inter-agent			
	Reas.	Act.	Mem.	Sec.	Task	Comm.	Term.	Orch.
Who&When [12]	✓	✓	–	–	–	–	–	–
MASFT [117]	✓	–	✓	–	✓	✓	✓	–
TRAIL [118]	✓	✓	✓	✓	✓	–	✓	✓
AgentFail [99]	✓	✓	✓	–	✓	–	✓	✓
Aegis [119]	✓	–	✓	–	✓	✓	✓	–
AgentDebug [102]	✓	✓	✓	✓	–	–	–	–
MP-Bench [120]	✓	✓	–	–	–	–	–	–
LongRCA [121]	✓	✓	–	–	–	–	–	✓

Note. Reas.=Reasoning, Act.=Action, Mem.=Memory, Sec.=Security, Task=Task Specification, Comm.=Communication, Term.=Termination, Orch.=Orchestration.

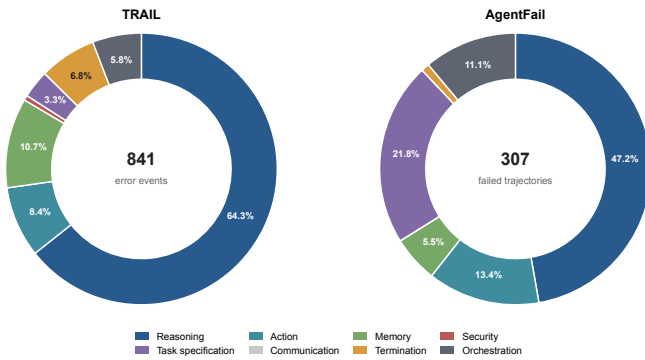


Fig. 11. Distribution of anomaly types in TRAIL and AgentFail under the AgentOps taxonomy.

VIII. DATASETS AND BENCHMARKS FOR AGENTOPS

To better advance the development of AgentOps, recent efforts have introduced a growing number of datasets and benchmarks. Although AgentOps typically consists of four key stages, i.e., monitoring, anomaly detection, root cause localization, and resolution, not all stages are equally amenable to standardized dataset construction. In particular, monitoring and resolution are difficult to benchmark in a unified manner due to their strong dependence on practical deployment environments, system instrumentation, and downstream operational constraints. Meanwhile, as discussed earlier, anomaly detection is still largely characterized by methods tailored to specific and relatively narrow categories of anomalies, rather than a broadly standardized evaluation paradigm. Consequently, root cause localization, also referred to as failure attribution, has emerged as the most mature and standardized direction for dataset and benchmark development in AgentOps. Therefore, in this section, we primarily summarize existing datasets and benchmarks for root cause localization. As shown in Table VIII, most of these datasets are constructed by simulating trajectories using widely adopted agent systems in the community, such as GAIA [122], ALFWorld [123], and WebShop [124], followed by manual or automated annotation. With the recent emergence of end-to-end testing frameworks for real-world Graphical User Interface (GUI) agents [125], we expect trajectory-based datasets derived from realistic GUI

interactions to continue expanding in both scale and diversity.

Several clear trends can be observed. First, the number of failure instances has steadily increased. A larger number of failures enables richer failure combinations and allows for more fine-grained evaluation of different methods. Second, the types of failures have become increasingly diverse. Various agent failure taxonomies have been proposed, as comprehensive coverage of failure types is essential to assess whether an algorithm exhibits systematic bias toward specific error categories. However, despite the numerous taxonomies proposed in prior work (Table IX), existing efforts provide only limited coverage of the anomaly space. In particular, they predominantly focus on intra-agent anomalies, such as reasoning failures, while largely overlooking anomalies that arise from inter-agent interactions. As illustrated in Fig. 11, reasoning anomalies account for 47.2%–64.3% in the two analyzed datasets, indicating a strong bias toward intra-agent issues. Meanwhile, with the emergence of automated multi-agent system construction frameworks such as Hermes Agent, effective inter-agent collaboration is becoming increasingly critical, calling for a more comprehensive characterization of anomalies at the inter-agent level. Third, recent studies have considered multiple plausible root causes. For example, MP-Bench [120] argues that failure attribution can be inherently ambiguous, making single-label deterministic evaluation potentially underestimate attribution methods. However, such ambiguity need not be intrinsic to failure attribution. Following Who&When, the root cause can be defined as the earliest causally responsible step whose correction would allow subsequent execution to proceed normally. From this perspective, multiple root causes may largely reflect the difficulty of obtaining precise causal annotations in complex agent systems. Thus, multi-root-cause datasets can also be viewed as a practical accommodation to annotation uncertainty, rather than evidence that failures inherently have multiple root causes.

IX. DISCUSSION

Timely and Proactive AgentOps. Current AgentOps techniques are still largely reactive, with many anomalies detected only after they have propagated through long execution trajectories or caused task failures. Future AgentOps should therefore shift from post-hoc analysis toward online and proactive operations. This requires continuous anomaly detection during execution, efficient diagnosis, and timely intervention before erroneous states accumulate. For example, root cause localization may adopt a coarse-to-fine strategy, where lightweight models first narrow down suspicious regions and LLMs subsequently perform fine-grained attribution. Combined with checkpointing and in-execution correction, such mechanisms can substantially shorten the delay between anomaly occurrence, diagnosis, and recovery.

Accurate and Evidence-Grounded AgentOps. Reliable AgentOps depends fundamentally on the quality of operational evidence. Conventional metrics, logs, and traces often capture structural execution behaviors but are insufficient for revealing semantic failures in reasoning, decision making, or agent interactions. Future systems should incorporate richer and higher-fidelity signals, including model-level information, execution

states, memory, checkpoints, tool interactions, and inter-agent communications. More importantly, these raw observations should be transformed into semantically meaningful and risk-aware signals, such as uncertainty estimates or consistency indicators. By jointly exploiting heterogeneous evidence across agent and system levels, AgentOps can support more accurate anomaly detection, root cause localization, and resolution.

X. LITERATURE SEARCH METHODOLOGY

We conducted a systematic literature search primarily using Google Scholar, complemented by backward and forward citation tracing and AI-assisted retrieval. The search covered literature published up to August 2026 and used combinations of keywords such as “agent failure,” “agent error,” “failure attribution,” “anomaly detection,” “root cause localization,” “monitoring,” and “resolution,” together with terms related to LLM-based agents and multi-agent systems.

We included studies that investigate failures, operational mechanisms, evaluation benchmarks, or software-engineering techniques directly applicable or transferable to AgentOps. Studies unrelated to agent-system reliability and operations, or without a clear connection to the AgentOps lifecycle, were excluded. Given the rapid evolution of the field, our survey may still be subject to selection bias and temporal-coverage bias: relevant studies may be missed by the adopted search strategy, and new work published after August 2026 is not covered.

XI. CONCLUSION

The continuous advancement of LLM reasoning capabilities has accelerated the development of agent systems, while also introducing new operational challenges arising from their stochastic and dynamic behaviors. This paper provides a systematic definition and taxonomy of agent-system anomalies, categorizing them into intra-agent and inter-agent anomalies, and proposes AgentOps as a comprehensive operational framework encompassing monitoring, anomaly detection, root cause localization, and resolution. Looking forward, future AgentOps should move toward more proactive and evidence-grounded operations by enabling timely online intervention and leveraging richer operational signals for reliable diagnosis and recovery.

REFERENCES

- [1] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, “Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning,” *arXiv preprint arXiv:2501.12948*, 2025.
- [2] A. Anthropic. (2024) The claude 3 model family: Opus, sonnet, haiku. claude-3 model card.
- [3] N. Li, C. Gao, M. Li, Y. Li, and Q. Liao, “Econagent: large language model-empowered agents for simulating macroeconomic activities,” 2024.
- [4] Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian *et al.*, “Toolllm: Facilitating large language models to master 16000+ real-world apis,” in *The Twelfth International Conference on Learning Representations*, 2024.
- [5] C. Pei, Z. Wang, F. Liu, Z. Li, Y. Liu, X. He, R. Kang, T. Zhang, J. Chen, J. Li *et al.*, “Flow-of-action: Sop enhanced llm-based multi-agent system for root cause analysis,” in *Companion Proceedings of the ACM on Web Conference 2025*, 2025, pp. 422–431.
- [6] X. Huang, J. Lian, Y. Lei, J. Yao, D. Lian, and X. Xie, “Recommender ai agent: Integrating large language models for interactive recommendations,” *ACM Transactions on Information Systems*, vol. 43, no. 4, pp. 1–33, 2025.
- [7] M. Wang, X. Xie, and Y. Huo, “Trajaudit: Automated failure diagnosis for agentic coding systems,” *arXiv preprint arXiv:2605.26563*, 2026.
- [8] Z. Durante, Q. Huang, N. Wake, R. Gong, J. S. Park, B. Sarkar *et al.*, “Agent ai: Surveying the horizons of multimodal interaction,” *arXiv preprint arXiv:2401.03568*, 2024.
- [9] N. Chakraborty, M. Ornik, and K. Driggs-Campbell, “Hallucination detection in foundation models for decision-making: A flexible definition and review of the state of the art,” *ACM Computing Surveys*, 2025.
- [10] Z. Deng, Y. Guo, C. Han, W. Ma, J. Xiong, S. Wen, and Y. Xiang, “Ai agents under threat: A survey of key security challenges and future pathways,” *ACM Computing Surveys*, vol. 57, no. 7, pp. 1–36, 2025.
- [11] Y. Shi, W. Yu, W. Yao, W. Chen, and N. Liu, “Towards trustworthy gui agents: A survey,” *arXiv preprint arXiv:2503.23434*, 2025.
- [12] S. Zhang, M. Yin, J. Zhang, J. Liu, Z. Han, J. Zhang, B. Li, C. Wang, H. Wang, Y. Chen *et al.*, “Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems,” in *Forty-second International Conference on Machine Learning*, 2025.
- [13] S. Kapoor, B. Stroebel, P. Kirgis, N. Nadgir, Z. S. Siegel, B. Wei, T. Xue, Z. Chen, F. Chen, S. Utpala, F. Ndzomga, D. Oruganty, S. Luskun, K. Liu, B. Yu, A. Arora, D. Hahm, H. Trivedi, H. Sun, J. Lee, T. Jin, Y. Mai, Y. Zhou, Y. Zhu, R. Bommasani, D. Kang, D. Song, P. Henderson, Y. Su, P. Liang, and A. Narayanan, “Holistic agent leaderboard: The missing infrastructure for ai agent evaluation,” in *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. [Online]. Available: <https://arxiv.org/pdf/2510.11977>
- [14] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafraan, K. Narasimhan, and Y. Cao, “React: Synergizing reasoning and acting in language models,” in *International Conference on Learning Representations*, 2023.
- [15] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, “Finetuned language models are zero-shot learners,” in *International Conference on Learning Representations*, 2022.
- [16] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, “Chain-of-thought prompting elicits reasoning in large language models,” *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.
- [17] V. Rawte, A. Sheth, and A. Das, “A survey of hallucination in large foundation models,” *arXiv preprint arXiv:2309.05922*, 2023.
- [18] J. Gallifant, A. Fiske, Y. A. Levites Strekalova, J. S. Osorio-Valencia, R. Parke, R. Mwau, N. Martinez, J. W. Gichoya, M. Ghassemi, D. Demner-Fushman *et al.*, “Peer review of gpt-4 technical report and systems card,” *PLOS digital health*, vol. 3, no. 1, p. e0000417, 2024.
- [19] Y. Yang, E. Chern, X. Qiu, G. Neubig, and P. Liu, “Alignment for honesty,” *Advances in Neural Information Processing Systems*, vol. 37, pp. 63 565–63 598, 2024.
- [20] M. Wang, Y. Zhang, C. Peng, Y. Chen, W. Zhou, J. Gu, C. Zhuang, R. Guo, B. Yu, W. Wang *et al.*, “Function calling in large language models: Industrial practices, challenges, and future directions,” 2025.
- [21] Z. Wu, H. Gao, J. He, and P. Wang, “The dark side of function calling: Pathways to jailbreaking large language models,” in *Proceedings of the 31st International Conference on Computational Linguistics*. Association for Computational Linguistics, Jan. 2024, pp. 584–592.
- [22] Tencent. (2025) Ai-infra-guard. [Online]. Available: <https://github.com/Tencent/AI-Infra-Guard>
- [23] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, “Lost in the middle: How language models use long contexts,” *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024.
- [24] C. Wang and J. V. Sun, “Unable to forget: Proactive interference reveals working memory limits in llms beyond context length,” *arXiv preprint arXiv:2506.08184*, 2025.
- [25] K. Zhang, Z. Sun, W. Yu, X. Zang, K. Zheng, Y. Song, H. Li, and J. Xu, “Qe-rag: A robust retrieval-augmented generation benchmark for query entry errors,” *arXiv preprint arXiv:2504.04062*, 2025.
- [26] F. Wang, X. Wan, R. Sun, J. Chen, and S. Ö. Arık, “Astute rag: Overcoming imperfect retrieval augmentation and knowledge conflicts for large language models,” *arXiv preprint arXiv:2410.07176*, 2024.
- [27] Z. Sun, X. Zang, K. Zheng, Y. Song, J. Xu, X. Zhang, W. Yu, and H. Li, “Redeep: Detecting hallucination in retrieval-augmented generation via mechanistic interpretability,” in *The Thirteenth International Conference on Learning Representations*, 2025.
- [28] J. Chen, H. Lin, X. Han, and L. Sun, “Benchmarking large language models in retrieval-augmented generation,” in *Proceedings of the AAAI*

- Conference on Artificial Intelligence*, vol. 38, no. 16, 2024, pp. 17754–17762.
- [29] Sanjeev. (2025) Emergent behavior in multi-agent systems: How complex behaviors arise from simple agent interactions.
 - [30] H. Zhang, J. Huang, K. Mei, Y. Yao, Z. Wang, C. Zhan, H. Wang, and Y. Zhang, “Agent security bench (asb): Formalizing and benchmarking attacks and defenses in llm-based agents,” in *International Conference on Learning Representations*, vol. 2025, 2025, pp. 35 331–35 366.
 - [31] M. Z. Pan, M. Cemri, L. A. Agrawal, S. Yang, B. Chopra, R. Tiwari, K. Keutzer, A. Parameswaran, K. Ramchandran, D. Klein *et al.*, “Why do multiagent systems fail?” in *ICLR 2025 Workshop on Building Trust in Language Models and Applications*, 2025.
 - [32] P. Altmann, J. Schönberger, S. Illium, M. Zorn, F. Ritz, T. Haider, S. Burton, and T. Gabor, “Emergence in multi-agent systems: A safety perspective,” in *International Symposium on Leveraging Applications of Formal Methods*. Springer, 2024, pp. 104–120.
 - [33] X. He, D. Wu, Y. Zhai, and K. Sun, “Sentinelagent: Graph-based anomaly detection in multi-agent systems,” *arXiv preprint arXiv:2505.24201*, 2025.
 - [34] E. Platon *et al.*, “Modeling exception management in multi-agent systems.” Ph.D. dissertation, Citeseer, 2007.
 - [35] P. D. OG. (2025) Building high-quality ai agent systems: Best practices.
 - [36] L. Zhang, Y. Zhai, T. Jia, X. Huang, C. Duan, and Y. Li, “Agentfm: Role-aware failure management for distributed databases with llm-driven multi-agents,” in *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, 2025, pp. 525–529.
 - [37] X. Li, S. Wang, S. Zeng, Y. Wu, and Y. Yang, “A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges,” *Vicinearth*, vol. 1, no. 1, p. 9, 2024.
 - [38] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu *et al.*, “Autogen: Enabling next-gen llm applications via multi-agent conversations,” in *First conference on language modeling*, 2024.
 - [39] G. Li, H. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, “Camel: Communicative agents for” mind” exploration of large language model society,” *Advances in neural information processing systems*, vol. 36, pp. 51 991–52 008, 2023.
 - [40] C. Bronsdon. (2025) Real-time anomaly detection for multi-agent ai systems.
 - [41] G. Zhang, Y. Yue, Z. Li, S. Yun, G. Wan, K. Wang, D. Cheng, J. X. Yu, and T. Chen, “Cut the crap: An economical communication pipeline for llm-based multi-agent systems,” in *The Thirteenth International Conference on Learning Representations*, 2025.
 - [42] Microsoft, “Taxonomy of failure mode in agentic ai systems,” 2025.
 - [43] J. Chen, J. Liang, and B. Wang, “Smurfs: Multi-agent system using context-efficient dfsdt for tool planning,” in *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 2025, pp. 3281–3298.
 - [44] A. Zhu, L. Dugan, and C. Callison-Burch, “Redel: A toolkit for llm-powered recursive multi-agent systems,” in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2024, pp. 162–171.
 - [45] S. Drake, “‘neural howlround’ in large language models: a self-reinforcing bias phenomenon, and a dynamic attenuation solution,” *arXiv preprint arXiv:2504.07992*, 2025.
 - [46] J. Smeds, K. Nybom, and I. Porres, “Devops: a definition and perceived adoption impediments,” in *International conference on agile software development*. Springer, 2015, pp. 166–177.
 - [47] Q. Cheng, D. Sahoo, A. Saha, W. Yang, C. Liu, G. Woo, M. Singh, S. Saverese, and S. C. Hoi, “Ai for it operations (aiops) on cloud platforms: Reviews, opportunities and challenges,” *arXiv preprint arXiv:2304.04661*, 2023.
 - [48] S. Pahune and Z. Akhtar, “Transitioning from mlops to llmops: Navigating the unique challenges of large language models,” *Information*, vol. 16, no. 2, p. 87, 2025.
 - [49] Cisco, “What is agentic operations (agenticops)?” <https://www.cisco.com/site/us/en/learn/topics/artificial-intelligence/what-is-agentic-operations-agenticops.html>, 2026, accessed: 2026-05-15.
 - [50] O. Authors, “Opentelemetry: A cloud native observability framework,” <https://opentelemetry.io>, 2019.
 - [51] L. Dong, Q. Lu, and L. Zhu, “Agentops: Enabling observability of llm agents,” *arXiv preprint arXiv:2411.05285*, 2024.
 - [52] S. Biswas, H. Bhatt, and K. Vaidhyanathan, “Architecting agentops needs change,” no. arXiv:2601.06456, Jan. 2026, arXiv:2601.06456 [cs.SE]. [Online]. Available: <http://arxiv.org/abs/2601.06456>
 - [53] D. Moshkovich and S. Zeltyn, “Taming uncertainty via automation: Observing, analyzing, and optimizing agentic ai systems,” no. arXiv:2507.11277, Nov. 2025, arXiv:2507.11277 [cs.AI]. [Online]. Available: <http://arxiv.org/abs/2507.11277>
 - [54] langdb, “Langdb: Llm-enhanced database exploration,” 2025. [Online]. Available: <https://github.com/langdb/ai-gateway>
 - [55] Langfuse, “Langfuse: Open-source llm tracing and observability,” 2025. [Online]. Available: <https://github.com/langfuse/langfuse>
 - [56] MLflow Project, a Series of LF Projects, LLC, “MLflow for GenAI: Build Production-Ready AI Applications,” <https://mlflow.org/genai>, 2025.
 - [57] Helicone, “Helicone: Llm observability platform,” 2025. [Online]. Available: <https://github.com/Helicone/helicone>
 - [58] LangWatch, “Langwatch,” 2025. [Online]. Available: <https://github.com/langwatch>
 - [59] Arize AI, Inc. (2025) Llamatrace — hosted phoenix: Llm tracing & evaluation platform. [Online]. Available: <https://phoenix.arize.com/llamatrace/>
 - [60] Traceloop. (2025) Openllmetry: Open-source observability for your llm application. [Online]. Available: <https://github.com/traceloop/openllmetry>
 - [61] Arize AI, Inc. (2025) Arize phoenix: Open-source llm tracing & evaluation platform. [Online]. Available: <https://phoenix.arize.com/>
 - [62] Literal AI, Inc. (2025) Literal ai: Rag llm evaluation & observability platform. [Online]. Available: <https://www.literalai.com/>
 - [63] Comet ML, Inc. (2025) Opik — open-source llm evaluation platform. [Online]. Available: <https://www.comet.com/site/products/opik/>
 - [64] Arize AI, Inc. (2025) Openinference: Opentelemetry instrumentation for ai observability. [Online]. Available: <https://github.com/Arize-ai/openinference>
 - [65] TruEra Inc. (2025) Trulens: Open-source llm evaluation & observability platform. [Online]. Available: <https://www.trulens.org/>
 - [66] HoneyHive AI, Inc. (2025) Honeyhive: Ai observability and evaluation platform. [Online]. Available: <https://www.honeyhive.ai/>
 - [67] Magniv, Inc. (2025) Promptlayer: Platform for prompt engineering, management, evaluation, and llm observability. [Online]. Available: <https://www.promptlayer.com/>
 - [68] AgentOps.ai. (2025) Agentops: Developer platform for ai agent observability. [Online]. Available: <https://www.agentops.ai/>
 - [69] confident-ai. (2024) Deepeval: The llm evaluation framework. [Online]. Available: <https://github.com/confident-ai/deepeval>
 - [70] LangChain, “LangSmith: The Agent Engineering Platform,” 2026. [Online]. Available: <https://www.langchain.com/langsmith-platform>
 - [71] L. Hanzlik, Y. Zhang, K. Grosse, A. Salem, M. Augustin, M. Backes, and M. Fritz, “Mlcapsule: Guarded offline deployment of machine learning as a service,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 3300–3309.
 - [72] Q. Huang, X. Dong, P. Zhang, B. Wang, C. He, J. Wang, D. Lin, W. Zhang, and N. Yu, “Opera: Alleviating hallucination in multi-modal large language models via over-trust penalty and retrospection-allocation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 13 418–13 427.
 - [73] A. Azaria and T. Mitchell, “The internal state of an llm knows when it’s lying,” in *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023, pp. 967–976.
 - [74] W. Kong, Y. Lin, X. Teoh, D.-M. Nguyen, R. Ren, J. Chang, H. Hu, and H. Chen, “Arc: Compiling large multi-modal requirement documents into runnable software systems,” *arXiv preprint arXiv:2602.13723*, 2026.
 - [75] X. Teoh, Y. Lin, D.-M. Nguyen, R. Ren, W. Zhang, and J. S. Dong, “Webtestpilot: Agentic end-to-end web testing against natural language specification by inferring oracles with symbolized gui elements,” *Proceedings of the ACM on Software Engineering*, vol. 3, no. FSE, pp. 1933–1956, 2026.
 - [76] Y. Zhou, C. Cui, J. Yoon, L. Zhang, Z. Deng, C. Finn, M. Bansal, and H. Yao, “Analyzing and mitigating object hallucination in large vision-language models,” in *The Twelfth International Conference on Learning Representations*, 2024.
 - [77] V. Quach, A. Fisch, T. Schuster, A. Yala, J. H. Sohn, T. S. Jaakkola, and R. Barzilay, “Conformal language modeling,” in *The Twelfth International Conference on Learning Representations*, 2024.
 - [78] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch, “Improving factuality and reasoning in language models through multiagent debate,” in *Forty-first International Conference on Machine Learning*, 2023.

- [79] X. Li, R. Zhao, Y. K. Chia, B. Ding, S. Joty, S. Poria, and L. Bing, "Chain-of-knowledge: Grounding large language models via dynamic knowledge adapting over heterogeneous sources," in *The Twelfth International Conference on Learning Representations*, 2024.
- [80] S. Kumar, A. Girdhar, R. Patil, and D. Tripathi, "Mcp guardian: A security-first layer for safeguarding mcp-based ai system," in *Proceedings of the 6th International Conference on Advanced Natural Language Processing (AdNLP 2025)*, 2025.
- [81] S. Chen, S. Wong, L. Chen, and Y. Tian, "Extending context window of large language models via positional interpolation," *arXiv preprint arXiv:2306.15595*, 2023.
- [82] Y. Zhang, R. Sun, Y. Chen, T. Pfister, R. Zhang, and S. Arik, "Chain of agents: Large language models collaborating on long-context tasks," *Advances in Neural Information Processing Systems*, vol. 37, pp. 132208–132237, 2024.
- [83] H. Hu, Y. Sun, and Q. Zhang, "Lrp4rag: Detecting hallucinations in retrieval-augmented generation via layer-wise relevance propagation," *arXiv preprint arXiv:2408.15533*, 2024.
- [84] J. Zhou, L. Wang, and X. Yang, "Guardian: Safeguarding llm multi-agent collaborations with temporal graph modeling," *arXiv preprint arXiv:2505.19234*, 2025.
- [85] A. Akli, M. Papadakis, M. Cordy, and Y. L. Traon, "Defective task descriptions in llm-based code generation: Detection and analysis," *arXiv preprint arXiv:2604.24703*, 2026.
- [86] S. Vijayvargiya, X. Zhou, A. Yerukola, M. Sap, and G. Neubig, "Ambig-swe: Interactive agents to overcome underspecificity in software engineering," in *The Fourteenth International Conference on Learning Representations*, 2026.
- [87] T. Zhang, P. Qin, Y. Deng, C. Huang, W. Lei, J. Liu, D. Jin, H. Liang, and T.-S. Chua, "Clamber: A benchmark of identifying and clarifying ambiguous information needs in large language models," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 10746–10766.
- [88] N. Edwards and S. Schuster, "Ask or assume? uncertainty-aware clarification-seeking in coding agents," *arXiv preprint arXiv:2603.26233*, 2026.
- [89] L. Kuhn, Y. Gal, and S. Farquhar, "Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation," *arXiv preprint arXiv:2302.09664*, 2023.
- [90] P. Manakul, A. Liusie, and M. Gales, "Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models," in *Proceedings of the 2023 conference on empirical methods in natural language processing*, 2023, pp. 9004–9017.
- [91] K. Liang, Z. Zhang, and J. Fernández Fisac, "Introspective planning: Guiding language-enabled agents to refine their own uncertainty," *arXiv e-prints*, pp. arXiv–2402, 2024.
- [92] M. Li, Y. Zhao, B. Yu, F. Song, H. Li, H. Yu, Z. Li, F. Huang, and Y. Li, "Api-bank: A comprehensive benchmark for tool-augmented llms," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 3102–3116.
- [93] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 8634–8652, 2023.
- [94] X. Wang, Y. Chen, L. Yuan, Y. Zhang, Y. Li, H. Peng, and H. Ji, "Executable code actions elicit better llm agents," in *Forty-first International Conference on Machine Learning*, 2024.
- [95] G. Zhang, Y. Yue, X. Sun, G. Wan, M. Yu, J. Fang, K. Wang, T. Chen, and D. Cheng, "G-designer: Architecting multi-agent communication topologies via graph neural networks," in *Forty-second International Conference on Machine Learning*, 2025.
- [96] Y. Ge, L. Xie, Z. Li, Y. Pei, and T. Zhang, "Who is introducing the failure? automatically attributing failures of multi-agent systems via spectrum analysis," *arXiv preprint arXiv:2509.13782*, 2025.
- [97] H. Zhang, Y. Shi, X. Gu, H. You, Z. Zhang, L. Gan, Y. Yuan, and J. Huang, "Graphtracer: Graph-guided failure tracing in llm agents for robust multi-turn deep search," *arXiv preprint arXiv:2510.10581*, 2025.
- [98] G. Zhang, J. Wang, J. Chen, W. Zhou, K. Wang, and S. Yan, "Agentracer: Who is inducing failure in the llm agentic systems?" *arXiv preprint arXiv:2509.03312*, 2025.
- [99] X. Ma, X. Xie, Y. Wang, J. Wang, B. Wu, M. Li, and Q. Wang, "Diagnosing failure root causes in platform-orchestrated agentic systems: Dataset, taxonomy, and benchmark," *arXiv preprint arXiv:2509.23735*, 2025.
- [100] G. Ma, J. Zhu, H. Guo, W. Shi, J. Shen, J. Liu, and Y. Liang, "Automatic failure attribution and critical step prediction method for multi-agent systems based on causal inference," *arXiv preprint arXiv:2509.08682*, 2025.
- [101] T. Zhu, Y. Wu, W. Jin, Y. Li, and G. Huang, "Stepfinder: A temporal semantic framework for failure attribution in multi-agent systems," *arXiv preprint arXiv:2606.03467*, 2026.
- [102] K. Zhu, Z. Liu, B. Li, M. Tian, Y. Yang, J. Zhang, P. Han, Q. Xie, F. Cui, W. Zhang *et al.*, "Where llm agents fail and how they can learn from failures," *arXiv preprint arXiv:2509.25370*, 2025.
- [103] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press, "Swe-agent: Agent-computer interfaces enable automated software engineering," *arXiv preprint arXiv:2405.15793*, 2024.
- [104] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, 2022.
- [105] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," in *Advances in Neural Information Processing Systems*, 2023.
- [106] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," in *Advances in Neural Information Processing Systems*, 2023.
- [107] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegrefe, U. Alon, N. Dziri, S. Prabhunoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh, and P. Clark, "Self-refine: Iterative refinement with self-feedback," in *Advances in Neural Information Processing Systems*, 2023.
- [108] H. Wang, C. M. Poskitt, and J. Sun, "Agentspec: Customizable runtime enforcement for safe and reliable llm agents," *arXiv preprint arXiv:2503.18666*, 2025.
- [109] Y. Shao, Y. Huang, J. Zou, Y. Shi, L. Yang, L. Ma, T. Su, and C. Wan, "Comfrey: Mitigating integration failures in llm-enabled software at run-time," 2026.
- [110] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," in *International Conference on Learning Representations*, 2023.
- [111] LangChain, "Langgraph: Persistence and time travel," Documentation for checkpointing, replay, and fork-based agent execution, 2024, available at: <https://docs.langchain.com/oss/python/langgraph/persistence>.
- [112] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, "Voyager: An open-ended embodied agent with large language models," *arXiv preprint arXiv:2305.16291*, 2023.
- [113] T. Shin, Y. Razeghi, R. L. Logan IV, E. Wallace, and S. Singh, "Autoprompt: Eliciting knowledge from language models with automatically generated prompts," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.
- [114] X. Liu, K. Ji, Y. Fu, W. L. Tam, Z. Du, Z. Yang, and J. Tang, "P-tuning: Prompt tuning can be comparable to fine-tuning universally across scales and tasks," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2022.
- [115] Z. Shi and A. Lipani, "Dept: Decomposed prompt tuning for parameter-efficient fine-tuning," in *The Twelfth International Conference on Learning Representations*, 2024.
- [116] C. Fernando, D. Banarse, H. Michalewski, S. Osindero, and T. Rocktäschel, "Promptbreeder: Self-referential self-improvement via prompt evolution," in *Proceedings of the 41st International Conference on Machine Learning*, 2024.
- [117] M. Cemri, M. Z. Pan, S. Yang, L. A. Agrawal, B. Chopra, R. Tiwari, K. Keutzer, A. Parameswaran, D. Klein, K. Ramchandran *et al.*, "Why do multi-agent llm systems fail?" *arXiv preprint arXiv:2503.13657*, 2025.
- [118] D. Deshpande, V. Gangal, H. Mehta, J. Krishnan, A. Kannappan, and R. Qian, "Trail: Trace reasoning and agentic issue localization," *arXiv preprint arXiv:2505.08638*, 2025.
- [119] F. Kong, R. Zhang, H. Yin, G. Zhang, X. Zhang, Z. Chen, Z. Zhang, X. Zhang, S.-C. Zhu, and X. Feng, "Aegis: Automated error generation and identification for multi-agent systems," *arXiv e-prints*, pp. arXiv–2509, 2025.
- [120] Y. In, M. Tanjim, J. Subramanian, S. Kim, U. Bhattacharya, W. Kim, S. Park, S. Sarkhel, and C. Park, "Rethinking failure attribution in multi-agent systems: A multi-perspective benchmark and evaluation," *arXiv preprint arXiv:2603.25001*, 2026.

- [121] Y. Zhang, B. Feng, C. Pei, Z. Wang, Z. Peng, X. Liu, H. Jiang, D. Ma, J. Zhang, Y. Yao, Y. Zhao, F. Sun, Y. Huo, Z. Liu, J. Li, G. Xie, and D. Pei, “Longrca bench: Diagnosing responsible roles and root causes in long-horizon agent failures,” 2026. [Online]. Available: <https://arxiv.org/abs/2608.15242>
- [122] G. Mialon, C. Fourier, T. Wolf, Y. LeCun, and T. Scialom, “Gaia: a benchmark for general ai assistants,” in *The Twelfth International Conference on Learning Representations*, 2023.
- [123] M. Shridhar, X. Yuan, M.-A. Côté, Y. Bisk, A. Trischler, and M. Hausknecht, “Alfworld: Aligning text and embodied environments for interactive learning,” *arXiv preprint arXiv:2010.03768*, 2020.
- [124] S. Yao, H. Chen, J. Yang, and K. Narasimhan, “Webshop: Towards scalable real-world web interaction with grounded language agents,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 20 744–20 757, 2022.
- [125] S. Y. Ahmed, S. Feng, C. Bae, and C. B. X. Zhang, “Specops: A fully automated ai agent testing framework in real-world gui environments,” *arXiv preprint arXiv:2603.10268*, 2026.