OpsAgent: An Evolving Multi-agent System for Incident Management in Microservices

Yu Luo
Nankai University
Tianjin, China

Jiamin Jiang
Nankai University
Tianjin, China

Jingfei Feng
Nankai University
Tianjin, China

Lei Tao
Nankai University
Tianjin, China

Qingliang Zhang
Nankai University
Tianjin, China

Xidao Wen
Alibaba Cloud
Beijing, China

Yongqian Sun*
Nankai University
Tianjin, China

Shenglin Zhang
Nankai University
Tianjin, China

Tong Liu
Lenovo
Tianjin, China

Wenjie Zhang
Lenovo
Tianjin, China

Dan Pei
Tsinghua University
Beijing, China

Abstract

Incident management (IM) is central to the reliability of large-scale microservice systems. Yet manual IM, where on-call engineers examine metrics, logs, and traces is labor-intensive and error-prone in the face of massive and heterogeneous observability data. Existing automated IM approaches often struggle to generalize across systems, provide limited interpretability, and incur high deployment costs, which hinders adoption in practice. In this paper, we present *OpsAgent*, a lightweight, self-evolving multi-agent system for IM that employs a training-free data processor to convert heterogeneous observability data into structured textual descriptions, along with a multi-agent collaboration framework that makes diagnostic inference transparent and auditable. To support continual capability growth, *OpsAgent* also introduces a dual self-evolution mechanism that integrates internal model updates with external experience accumulation, thereby closing the deployment loop. Comprehensive experiments on the OPENRCA [44] benchmark demonstrate state-of-the-art performance and show that *OpsAgent* is generalizable, interpretable, cost-efficient, and self-evolving, making it a practically deployable and sustainable solution for long-term operation in real-world microservice systems. Notably, its deployment in Lenovo's production environment further validates its effectiveness in real-world industrial settings.

CCS Concepts

• **Software and its engineering → Software maintenance tools**.

Keywords

Heterogeneous Observability Data, Multi-agent System, Self-evolution, Incident Management

1 Introduction

Microservice systems have become the de facto architecture for building modern software services, with wide deployments across industries such as IT, government, and finance [5]. However, incidents (*e.g.*, service disruptions and outages) [5, 35] are inevitable due to the complexity of microservice systems, often resulting in catastrophic economic and operational consequences. For instance, on June 12, 2025, a faulty quota-control deployment in Google Cloud triggered a global outage that lasted nearly eight hours, disrupting more than 80 GCP services and cascading into failures across e-commerce, finance, AI applications, entertainment platforms, and transportation systems worldwide. The economic impact of this incident was substantial, as it encompassed not only Google's direct losses but also widespread hidden costs borne by countless enterprises and end users affected by the disruption [1]. Thus, comprehensive incident management (IM) that integrates anomaly detection (AD), failure triage (FT), and root cause localization (RCL) is essential to recover from such disruptions [5, 52].

Traditionally, on-call engineers (OCEs) manually inspect metrics, logs, and traces to identify the root cause when incidents occur [5]. This process yields interpretable results, as OCEs reason step by step and accumulate expertise, but it becomes infeasible at scale due to the overwhelming volume and heterogeneity of observability data [22, 35].

To alleviate this burden, automated IM with AI techniques has been extensively explored, falling into two main categories: deep learning (DL)-based IM [15, 23, 35, 37, 47, 51, 56] and large language model (LLM)-based IM [3, 5, 9, 40, 52–54]. Deep learning-based approaches leverage neural networks trained via supervised [15, 51] and self-supervised learning [33, 35] to extract complex failure patterns from observability data. However, the inherent "black-box" nature of neural networks yields predictions without transparent reasoning chains, making them hard for OCEs to trust and adopt. Furthermore, these models generalize poorly as they are trained on system-specific data, which usually require costly data collection and retraining when moved to new systems. LLM-based approaches exploit the strong reasoning and natural-language understanding of LLMs, generating diagnoses directly from incident titles and summaries or by matching to similar historical cases [3, 5, 54]. However,

*Yongqian Sun is the Corresponding author.

two limitations hinder practical deployment: first, many methods depend on large closed-source models (*e.g.*, GPT-4), which impose high deployment costs as well as privacy-exposure risks [5, 54]; second, without step-wise inference over raw observability data, the pipeline skews toward shallow similarity matching, which limits accuracy and offers no mechanism to accumulate expertise through continued usage [3, 5, 54]. As a result, despite promising research progress and good performance on public datasets, automated IM techniques have yet to achieve widespread adoption in practice, and many companies still rely heavily on manual IM.

Thus, OCEs urgently call for ***a deployable and sustainable IM approach***. Recent advances in multi-agent systems (MAS) suggest a promising direction, as they excel at decomposing complex reasoning tasks through collaboration and have already shown success in software engineering domains such as code generation [12, 49, 50], quality assurance [11, 36], and requirements analysis [4, 14]. Nevertheless, realizing a practically deployable and sustainable MAS-based IM requires overcoming three key technical challenges.

(C1) How to generalize MAS-based IM under heterogeneous and shifting microservice systems? Microservice systems vary widely in architecture and observability instrumentation, yielding voluminous and heterogeneous observability data. A common practice in existing MAS-based IM approaches is to let specialized agents directly operate on raw observability data [53]. While this design simplifies data ingestion, it forces each agent to rely on different modality-specific inputs (*e.g.*, metrics, logs, traces), which can lead them to form divergent understandings of the same incident and make their conclusions difficult to reconcile. The challenge is to design MAS-compatible data processing pipelines that can transform heterogeneous observability data into coherent representations. This is non-trivial, as formats, semantics, and granularities differ drastically across modalities, and naive abstraction may discard diagnostic cues essential for accurate reasoning.

(C2) How to ensure interpretable reasoning in MAS-based IM? In practice, OCEs cannot rely on predictions alone—they must audit the reasoning process before acting on diagnostic results, since misdiagnosis may lead to cascading incidents. In a MAS, interpretability hinges on clear role specification and a well-structured collaboration workflow that exposes intermediate steps, where the central challenge is balancing granularity with clarity. Roles that are too coarse obscure responsibility, whereas overly fine-grained roles impose excessive coordination overhead. At the same time, poorly designed interaction workflows may lead to ad hoc communications between agents, reducing transparency and hindering human auditing.

(C3) How to enable continual capability growth in MAS-based IM? Incidents in real-world microservice systems are diverse, evolving with frequent software updates. Traditional supervised or self-supervised training paradigms produce static models that tend to memorize patterns rather than improve intrinsic diagnostic skills. Existing MAS-based approaches, such as D-Bot [57] and Flow-of-Action [24], are built on closed-source LLMs with fixed capabilities (*e.g.*, GPT-4) and static SOPs, neither of which provides mechanisms for autonomous learning or capability evolution, making them ill-suited to adapt to novel incidents. The challenge is thus to enable continual capability growth in MAS-based IM. However, enhancing diagnostic capability is difficult because internal parameter updates are often unstable and fail to consolidate acquired experience, while relying solely on external knowledge (*e.g.*, SOPs and knowledge base) limits the improvement of the agent's intrinsic reasoning capability.

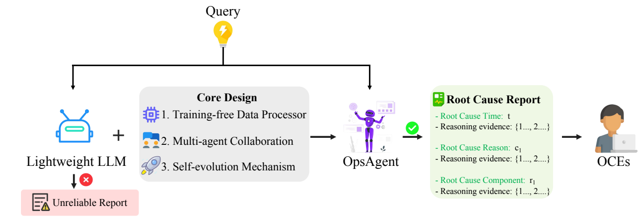

Figure 1: From lightweight LLM to MAS-based IM. *OpsAgent* turns a lightweight LLM into a deployable and sustainable IM system by incorporating (1) training-free data processor (Section 3.2), (2) multi-agent collaboration (Section 3.3), and (3) self-evolution mechanism (Section 3.4).

To address these challenges, we present *OpsAgent* (as shown in Fig. 1), a lightweight and modular MAS for IM. Instead of relying on massive closed-source models, *OpsAgent* employs a relatively small 14B-parameter model as its reasoning core, and is explicitly designed to support cross-system generalization, interpretable diagnostics, and continual capability growth. Specifically, *OpsAgent* tackles ***(C1)*** by introducing a training-free data processor that unifies heterogeneous observability data into structured textual representations, enabling consistent reasoning across diverse environments. For ***(C2)***, it employs a multi-agent collaboration framework that mirrors human problem solving, where specialized roles and coordinated workflows make diagnostic reasoning both transparent and auditable. For ***(C3)***, it integrates a dual self-evolution mechanism that combines reinforcement learning with reflection-based knowledge distillation, allowing the system to progressively enhance its diagnostic capability rather than remain static. Our contributions are summarized as follows:

(1) We propose *OpsAgent*, a novel MAS-based IM method that is generalizable, interpretable, cost-efficient and self-evolving, which offers a practically deployable and sustainable solution for real-world microservice systems.
(2) We introduce a dual self-evolution mechanism that integrates internal model updates with external experience accumulation. This closes the deployment loop by turning validated outcomes into consolidated knowledge and guiding subsequent cases, enabling sustained and auditable capability growth for long-term deployment in dynamic microservice systems.
(3) We evaluate *OpsAgent* on the OPENRCA [44] benchmark and demonstrate state-of-the-art performance. Through comprehensive experiments, we also prove its generalizability, interpretability, cost-efficiency and capability of self-evolving. We also deploy it in Lenovo's production environments to validate its practical utility. To ensure reproducibility, we release code, prompts and data [1].

[1] https://anonymous.4open.science/r/OpsAgent-CCC0

2 Background

2.1 Observability in Microservice System

Microservices architecture is a software system design paradigm where applications are decomposed into a set of loosely coupled, independently deployable services [6]. This approach improves scalability, agility, and maintainability by allowing each service to evolve and operate independently. In microservices environments, effective monitoring and understanding of system behavior rely heavily on three fundamental types of observability data:

- **Metrics** are structured time-series data that quantify system performance and resource usage, which offer a high-level overview of system health and trends.
- **Logs** are semi-structured text data that record system event details, such as service start, service shut down, and error stack traces.
- **Traces** are topological records generated by distributed tracing systems that capture the full invocation path of requests across services. Traces also reveal the sequence, duration, and dependencies of service calls, enabling analysis of cross-service interactions.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning paradigm where an agent learns an optimal policy via repeated environment interactions to maximize cumulative reward. Among various RL algorithms, **proximal policy optimization (PPO)** stands out as it is a policy-gradient RL algorithm that improves training stability and sample efficiency by using a clipped objective to limit policy updates, preventing collapse and balancing exploration and exploitation [2, 30].

2.3 Retrieval-Augmented Generation

While LLMs excel at text generation and semantic comprehension, they have two key limitations [8]: static knowledge and insufficient specialized domain expertise. To address these shortcomings, the **Retrieval-Augmented Generation (RAG)** [7, 25, 29] framework integrates external knowledge into generation, enabling content production based on retrieval results. It retains LLMs' language generation capabilities while enhancing output accuracy and domain adaptability via external knowledge injection, serving as a key technology to boost large models' practicality in professional scenarios.

2.4 Problem Definition

In our setting, IM is framed as a natural-language-driven multi-task problem. The input consists of a natural language query q, which may describe one or more incidents within a given time window and specify a combination of three subtasks: AD, FT, and RCL. Let the multimodal observability data be denoted as $\mathcal{X} = (X^M, X^L, X^T)$, where X^M, X^L, and X^T correspond to metrics, logs, and traces, respectively.

The AD task aims to identify the root cause occurrence time t, *i.e.*, the earliest timestamp at which anomalous behavior begins to manifest in the system. The FT task involves selecting the most probable failure type c from a predefined category space $\mathbb{C} = \{c_1, c_2, \ldots, c_n\}$. Finally, the RCL task localizes the most likely faulty component r from the set of root cause components $\mathbb{R} = \{r_1, r_2, \ldots, r_m\}$, encompassing entities across different system levels (*e.g.*, nodes, services, pods). The overall objective is to learn a mapping function $\mathcal{F} : (q, \mathcal{X}) \rightarrow (t, c, r)$.

3 Methodology

3.1 Overview

OpsAgent consists of three modules: (1) training-free data processor (Section 3.2), (2) multi-agent collaboration (Section 3.3), and (3) self-evolution mechanism (Section 3.4). Each module directly targets one of the challenges in Section 1. ***(C1)*** To generalize across heterogeneous deployments, the training-free data processor extracts abnormal patterns from raw metrics, logs, and traces and converts them into unified, semantically aligned textual descriptions that all agents can consume consistently. ***(C2)*** To ensure interpretable reasoning, the multi-agent collaboration framework mirrors human problem solving by defining well-scoped expert roles (*i.e.*, AD, FT, and RCL), coordinating them with an orchestrator, and supporting iterative cross-review that produces explicit intermediate evidence and an auditable diagnostic trail. ***(C3)*** To enable continual capability growth, the self-evolution mechanism integrates intrinsic updates via PPO fine-tuning with explicit experience accumulation through agent reflection, which allows *OpsAgent* to improve over time. Finally, *OpsAgent* is deliberately lightweight as it operates with a modest 14B-parameter open-source model as the reasoning core, avoiding massive closed-source LLMs and keeping deployment costs low. This design supports a practically deployable and sustainable solution for real-world microservice systems.

3.2 Training-free Data Processor

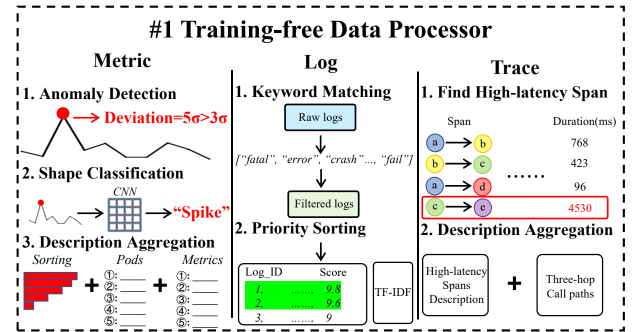

Figure 2: Training-free Data Processor. The processor handles three types of observability data separately: metrics (left), logs (middle), and traces (right).

Unlike DL-based IM methods that require large-scale data to learn feature distributions [15, 33, 35, 51], our data processor adopts a training-free approach as shown in Fig. 2. This design eliminates the need for costly data collection and retraining, while ensuring better generalization across heterogeneous microservice systems. The key idea is to extract abnormal patterns from raw metrics, logs, and traces via statistical and heuristic techniques, then convert them into unified textual descriptions that serve as the input for the

downstream agents in AD, FT, and RCL. We presented an illustrative example of the data descriptions in Fig. 3.

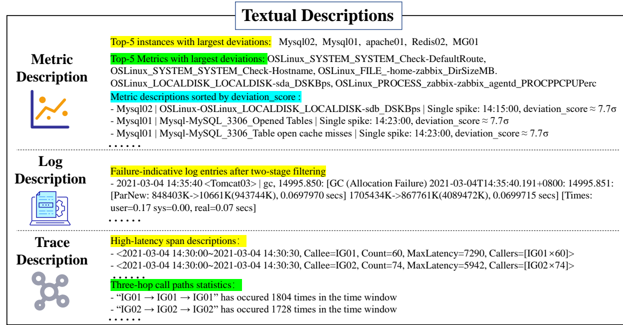

Figure 3: Illustrative example of data descriptions.

Metrics. We process metrics in three stages to filter noise and highlight diagnostically useful signals. *(1) Anomaly Detection.* Raw metrics are often noisy and periodic, so we first identify statistically significant deviations to reduce volume and concentrate on incident-indicative behaviors. We apply the 3-sigma rule within a sliding window, marking samples with $\text{score}(x_t) = \frac{|x_t - \mu|}{\sigma} > 3$ as anomalies, and retain their deviation scores in units of σ. *(2) Shape Classification.* Since different anomaly patterns (*e.g.*, spikes, steady increases, level shifts) imply different physical meanings, we use a pre-trained CNN [2] with a context window of 20 steps before and 10 after to assign each anomaly a discrete shape label [42]. *(3) Description Aggregation.* To provide agents with compact yet informative inputs, anomalies are transformed into textual descriptions of the form ⟨*service_instance, metric_name, anomaly_pattern: timestamp, deviation_score = kσ*⟩. We then sort anomalies by deviation score, aggregate scores per service instance to select the top-5 pods, and per metric name to select the top-5 metrics. This preserves rich contextual and statistical information while filtering for the most diagnostically relevant signals.

Logs. We adopt a two-stage pipeline to extract incident-indicative logs while discarding large volumes of irrelevant entries. *(1) Keyword Matching.* Operational logs dominate raw log data, so we first filter by a predefined incident-related lexicon (*e.g.*, *fatal*, *error*, *crash*, *fail*) to remove routine operational messages. *(2) Priority Sorting.* Even after keyword filtering, many entries remain and are bursty. We parse logs into templates with Drain3 [10] to normalize variable content, then rank templates by TF–IDF [28] so that entries that are salient within the current window yet uncommon are prioritized. We keep entries whose templates rank above an adaptive threshold (default: 80th percentile) and deduplicate those sharing the same template within a one-minute window by retaining the earliest instance. The resulting log descriptions are the filtered raw entries, preserving full context for downstream agents while emphasizing high-signal, non-redundant evidence.

Traces. We process traces in two steps to surface latency anomalies and their structural context. *(1) Find High-latency Span.* We classify spans as high latency if their latency exceeds a threshold specific to each call type. Because latency distributions vary across call types, a single global threshold can mislabel normal calls in some services and miss true outliers in others. We therefore use a flexible per-call-type threshold (default: 95th percentile [34]) and mark spans above it as high latency. *(2) Description Aggregation.* First, to capture co-occurring hotspots and context, we group high-latency spans by 60 s windows and callee, computing per window the total calls, maximum latency, and caller distribution. This yields textual records ⟨*time_interval, callee, count, max_latency, callers*⟩, which highlight when and where latency concentrates. Second, to reveal recurrent patterns that may indicate systemic bottlenecks, we extract three-hop call paths (*grandparent* → *caller* → *callee*) from high-latency spans and tally their global frequencies. Together, these descriptions preserve essential temporal and topological cues while suppressing routine or low-latency traffic, providing focused representation for downstream agents.

[2]We trained this CNN to classify the anomaly shapes, with its design corresponding to the PatternMatcher method [42].

3.3 Multi-agent Collaboration

As illustrated in Fig. 4, this module has three parts. We first outline the workflow from a natural-language *Query* to a *Root Cause Report* coordinated by the *Orchestrator* and expert agents. We then describe the cross-review mechanism that lets agents critique and refine one another's reasoning to improve accuracy and auditability. Finally, we explain how online deployment and offline training use the *Root Cause Report* to close the loop and sustain capability growth.

3.3.1 Workflow. All agents in *OpsAgent* are defined through an *agent profile*, a structured prompt template that specifies the agent's name, task description, operational instructions, and illustrative examples. This unified specification ensures consistent behavior and sets the stage for the workflow below: Given a *Query*, the *Intent Interpreter* ($\mathcal{I}$) extracts the analysis time window and the requested tasks (AD, FT, RCL). The *Orchestrator* ($\mathcal{O}$) then retrieves metrics, logs, and traces within that window and invokes the training-free data processor to convert them into unified, semantically aligned descriptions. This normalization provides a shared evidence base for all agents, improving consistency and auditability. With roles aligned to diagnostic tasks rather than data modalities, three expert agents—*Anomaly Sentinel* ($\mathcal{A}_1$) for AD, *Failure Diagnoser* ($\mathcal{A}_2$) for FT, and *Root Detective* ($\mathcal{A}_3$) for RCL—reason over the same descriptions using Chain-of-Thought prompting [41] to produce task-specific answers together with their stepwise rationales in parallel. Mapping roles to AD/FT/RCL avoids early information asymmetry (*e.g.*, splitting by metrics/logs/traces would deprive each agent of complementary signals) and mirrors real operational practice [13], thereby supporting interpretable collaboration. Then, the *Orchestrator* coordinates cross-review to reconcile disagreements and surface missing evidence (details in the next subsection). When refinement terminates, it compiles a *Root Cause Report* that records the final results and the intermediate reasoning evidence, enabling human auditing and directly addressing (***C2***).

3.3.2 Cross-review Mechanism. Cross-review leverages the fact that the three expert agents—$\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$—bring distinct yet complementary perspectives to the same incident. A single perspective can miss context or overfit local evidence, whereas peer

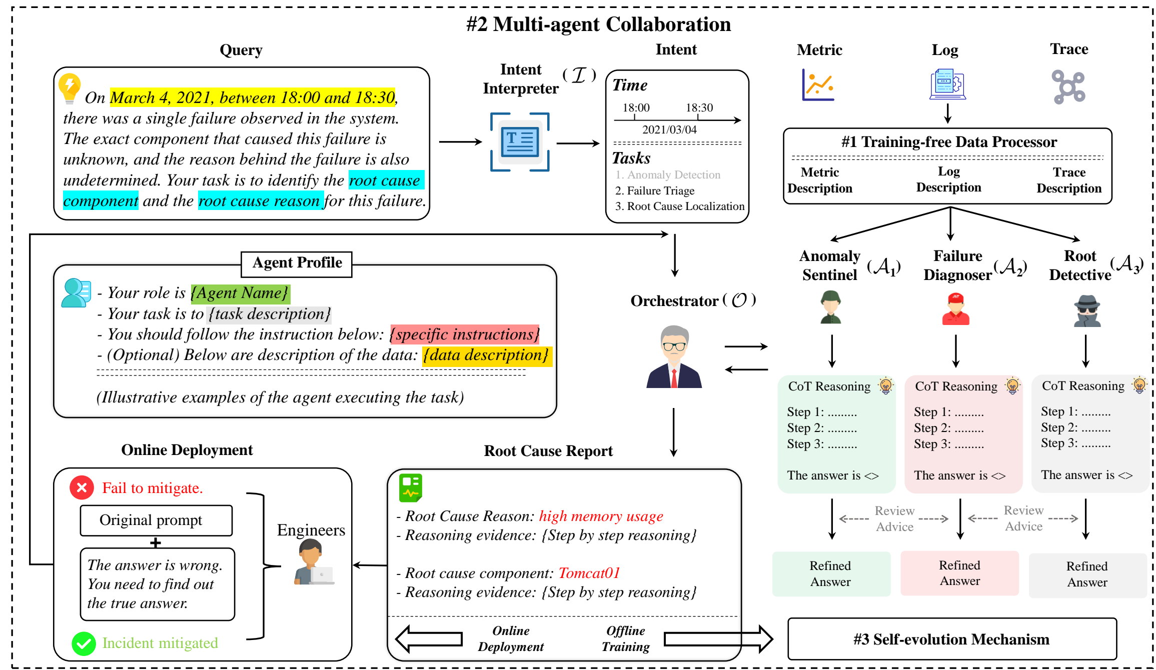

Figure 4: Multi-agent Collaboration. Agents with predefined roles (via agent profile) cooperate under a structured workflow and cross-review mechanism to enhance reasoning from multiple perspectives. The Root Cause Report not only guides online incident mitigation but also feeds offline training, closing the loop for sustainable capability growth.

critique helps surface gaps and reconcile divergent lines of reasoning. By requiring agents to critique others' rationales, cross-review improves accuracy and exposes intermediate reasoning in a form that OCEs can audit. After each agent produces an initial answer with stepwise reasoning, the *Orchestrator* initiates cross-review by bundling each answer and its rationale and dispatching it to the other two agents for peer review (*e.g.*, $\mathcal{A}_1$ reviews $\mathcal{A}_2$ and $\mathcal{A}_3$, and symmetrically for the others). Each agent then returns concise *review advice* to its peers, focusing on overlooked or weak evidence, unclear reasoning that warrants clarification, and plausible alternative hypotheses. Then, the *Orchestrator* prompts agents to refine their answers in accordance with the *review advice*. All review messages and refined rationales are recorded and included in the *Root Cause Report*, creating an auditable trail that substantiates the final diagnosis.

3.3.3 Online Deployment and Offline Training. The *Root Cause Report* serves two purposes: guiding operational remediation and providing learning feedback for continual improvement. In online deployment, engineers validate the report via mitigation actions. A successful mitigation confirms the diagnosis and closes the incident, whereas a failed mitigation triggers system re-analysis. The *Orchestrator* augments the original *Query* with feedback indicating that the previous diagnosis was incorrect, then re-invokes the multi-agent collaboration until a diagnosis is confirmed by successful mitigation. In offline training, accumulated reports are fed to the self-evolution mechanism (Section 3.4) to update agents via PPO-guided optimization and to distill reusable experience. This closes the loop between operations and learning, improving diagnostic capability over time while preserving deployability.

3.4 Self-evolution Mechanism

To continuously enhance the causal diagnostic capability of *OpsAgent*, we design a self-evolution mechanism that operates from two complementary perspectives as illustrated in Fig. 5. Internally, agents are updated through PPO-based fine-tuning with well-designed rewards, thereby strengthening their intrinsic reasoning ability. Externally, *OpsAgent* invokes a reflection process in which agents summarize past diagnostic experiences and distill reusable knowledge into a knowledge base. In this manner, the system is able to evolve beyond static pattern memorization, progressively improving its diagnostic performance from real-world cases, akin to how OCEs accumulate expertise over time.

3.4.1 PPO Training. To enhance the diagnostic reasoning capability of *OpsAgent*, we adopt PPO [30], a stable reinforcement learning algorithm that aligns well with capability-centric learning paradigms, to train the three expert agents, $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$. PPO's clipped objective and an adaptive KL penalty help stabilize learning and limit excessive policy shifts while maintaining sample efficiency.

A carefully designed reward model is employed to guide the optimization process. Optimizing only for accuracy can encourage shortcut behavior, whereas incorporating stepwise reasoning quality into the reward improves auditability and strengthens peer critique in cross-review. We therefore design the reward model to combine accuracy and reasoning quality as complementary signals

for PPO training. Accuracy is measured in a binary manner, assigning a score of 5 for a correct diagnosis and 0 otherwise. Reasoning quality is assessed by an external judge model *Qwen3-235B-A22B*, which is well suited for this role since evaluating reasoning evidence is more tractable than generating it [20]. As it is used only for reward estimation rather than as the reasoning backbone, this design decouples reward evaluation from the policy model while incurring only limited additional cost. It scores each reasoning chain along four dimensions—consistency, clarity, relevance, and rationality—on a 0–5 scale. The average of these scores is then combined with the accuracy reward, weighted by a tunable coefficient $\alpha \in [0, 1]$, allowing flexible adjustment of their relative importance. During training, each rollout, which refers to the reasoning trajectory produced by *OpsAgent* for a given *Query*, is assigned a reward according to the above scheme. The reward is then used to update the parameters of the three expert agents individually via PPO, ensuring that each agent improves its task-specific reasoning competence.

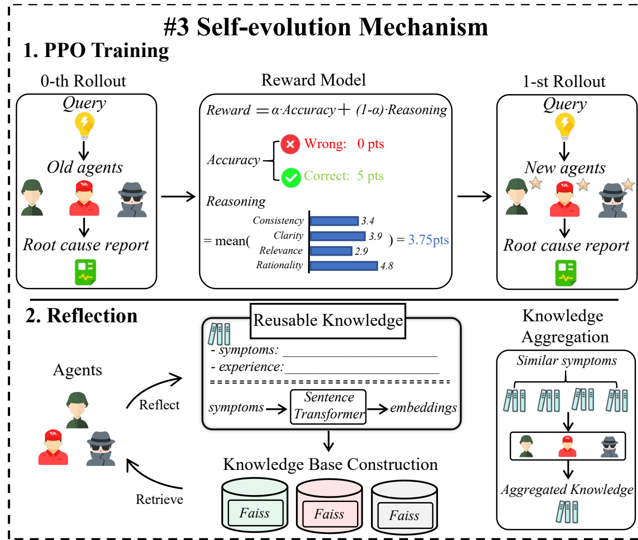

Figure 5: Self-evolution Mechanism. Internally, agents are fine-tuned via PPO training with a carefully designed reward model (top). Externally, a reflection process distills reusable knowledge into a task-specific knowledge base, which is later leveraged through RAG for knowledge injection (bottom).

3.4.2 Reflection. While internal parameter optimization via PPO training enhances task-specific reasoning capability, it is insufficient for sustaining long-term evolution. In practice, diagnostic systems must not only adapt their parameters but also explicitly accumulate reusable experience, much like OCEs who distill repeated operational experience into shared troubleshooting guides. To this end, *OpsAgent* incorporates a reflection mechanism that distills knowledge from successfully resolved cases, ensuring that only validated trajectories contribute to future reasoning. After completing a diagnostic task correctly, the corresponding agent reflects on its reasoning trajectory and outcome, abstracting reusable knowledge, such as characteristic symptom–root cause patterns. Each knowledge entry is structured as a pair $< symptoms, experience >$, where the symptoms serve as the key. The entries are embedded using a Sentence-Transformer [26] and stored in a faiss vector database. Importantly, each expert agent (*i.e.*, $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$) maintains its own knowledge base, as the nature of accumulated experience differs across diagnostic tasks. By grounding reflection in successfully verified cases, the system avoids propagating erroneous reasoning and builds a reliable knowledge repository that complements PPO training.

When handling a new case, the agent first generates a symptom key based on the input data descriptions and then queries its task-specific knowledge base via RAG, injecting the retrieved experiences into its prompt as auxiliary context. To keep the knowledge base compact and reliable, we aggregate entries that share the same symptom key: if two entries conflict, the newer replaces the older to account for obsolete practices; if they are complementary, we merge them so that multiple valid experiences can coexist. This reconciliation preserves conciseness without discarding useful diversity. During early deployment, the knowledge base may be sparse, and if retrieval fails, the agent falls back to pure CoT reasoning without external retrieval to maintain robustness [16].

Together, PPO-based training and reflection provide complementary paths for continual improvement: the former strengthens agents' intrinsic reasoning strategies, while the latter consolidates reusable knowledge from past cases. By combining implicit parameter updates with explicit experience accumulation, *OpsAgent* evolves beyond static pattern memorization and continually enhances its diagnostic expertise. This evolution addresses (***C3***), enabling sustained capability growth in MAS-based IM and mirroring how OCEs refine their skills through both practice and knowledge accumulation.

4 Experiments

4.1 Experimental Setup

4.1.1 Datasets. We conducted extensive experiments on the OPENRCA [44] benchmark dataset which consists of 335 incident cases collected from 3 heterogeneous microservice systems (*i.e.*, Telecom, Bank, Market) deployed in the real-world environment, accompanied by over 68 GB of telemetry data. The dataset underwent rigorous preprocessing, including standardization and balancing, and was further calibrated by 3 experienced engineers who verified that each root cause label could be supported by the associated telemetry, ensuring the reliability of the benchmark. Specifically, (1) **Telecom** includes 5 failure types in $\mathbb{C}$ and 43 candidate components in $\mathbb{R}$, comprising 22 virtual machine operating systems, 8 pods, and 13 database services; (2) **Bank** contains 8 failure types and 14 candidate components corresponding to its 14 pods; (3) **Market** has 15 failure types and 56 candidate components, consisting of 6 nodes, 40 pods, and 10 services.

4.1.2 Baselines. We compare *OpsAgent* against five frameworks: two designed specifically for IM (*i.e.*, ART [35] and RCA-Agent [44]), and three general-purpose open-source frameworks (*i.e.*, CoT [41], ReAct [46], Reflexion [31]). For DL-based IM approaches, we select ART [35] due to its state-of-the-art performance, while other

Table 1: Comparison with baselines across seed LLMs. Metrics (%): Correct and Partial are reported per system and as averages; the final column reports average time per incident (s/case).

Seed LLM	Method	Telecom		Bank		Market		Avg		s/case
		Correct	Partial	Correct	Partial	Correct	Partial	Correct	Partial	
Qwen2.5-14B-Instruct-1M	*OpsAgent*	30.00	45.00	**18.52**	**40.74**	**10.17**	**40.68**	**16.54**	**41.35**	71.18
	CoT [41]	10.00	20.00	0.00	1.85	5.08	13.55	1.50	9.77	16.83
	ReAct [27]	5.00	20.00	0.00	1.85	0.00	5.08	0.75	6.01	164.24
	Reflexion [31]	0.00	10.00	3.70	11.11	1.69	5.08	2.26	8.27	241.55
gpt-oss-20b	*OpsAgent*	**40.00**	**55.00**	5.56	11.11	6.78	16.95	11.28	20.30	53.48
	CoT [41]	0.00	5.00	0.00	0.00	0.00	0.00	0.00	0.75	27.42
	ReAct [27]	0.00	0.00	0.00	0.00	1.69	1.69	0.75	0.75	81.09
	Reflexion [31]	0.00	0.00	0.00	0.00	1.69	3.38	0.75	1.50	165.74
Phi-3-medium-128k-instruct	*OpsAgent*	10.00	35.00	0.00	9.26	3.39	10.17	3.01	13.53	57.94
	CoT [41]	0.00	10.00	0.00	0.00	0.00	0.00	0.00	1.50	20.33
	ReAct [27]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	48.42
	Reflexion [31]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.05
Claude 3.5 Sonnet	RCA-Agent [44]	20.00	35.00	16.67	35.19	3.39	28.81	11.28	32.33	287.71
	ART [35]	0.00	15.00	0.00	11.11	1.69	23.73	0.75	17.29	**2.27**

methods such as DeepHunt [33] and DiagFusion [51] are excluded as they do not cover all tasks required in our scenario (*e.g.*, AD). For LLM-based IM approaches, we include RCA-Agent [44], the official OPENRCA diagnostic framework and the top-performing approach on its benchmark. We evaluate it with *Claude 3.5 Sonnet* as the seed LLM to follow its strongest configuration, since RCA-Agent becomes ineffective when instantiated with ≤20B-scale models due to its strong dependence on code generation and error handling capabilities. We exclude other LLM-based IM methods, specifically mABC [53], ICL_RCA [54], COMET [40], due to their lack of coverage for all required tasks in our scenario, and RCA-Copilot [5], as it is not an open-source framework. To ensure a comprehensive and fair comparison, we also incorporate three well-known general-purpose open-source frameworks—CoT [41], ReAct [46], and Reflexion [31]—all of which leverage LLMs for reasoning and decision-making. For the seed LLMs, we restrict to models with at most 20B parameters and select *Qwen2.5-14B-Instruct-1M*, *gpt-oss-20b*, and *Phi-3-medium-128k-instruct*.

4.1.3 Evaluation Metrics. As described in Section 2.4, our system needs to handle arbitrary combinations of three subtasks: AD, FT, and RCL. Each subtask follows a clearly defined success criterion. An AD subtask is considered correct if the predicted timestamp lies within one minute (±60 seconds) of the ground-truth label. FT and RCL are evaluated in a single-pass setting: a prediction is counted as correct only when it exactly matches the ground-truth label (Top-1 match). Then, for any input query that may combine multiple subtasks, we utilize $Correct = \frac{num_c}{N}$ and $Partial = \frac{num_p}{N}$ as evaluation metrics, where N denotes the total number of queries in the dataset, num_c is the number of queries for which every requested subtask is correctly solved, and num_p is the number of queries for which at least one requested subtask is correctly solved.

4.1.4 Implementations. We implemented *OpsAgent* using Python 3.10.16 with PyTorch 2.6.0, Transformers 4.51.1, and accelerate 1.7.0. We perform a random split, assigning 60% of the data to the training set and the remainder to the test set. Experiments were conducted on a server with 16-core Intel Xeon Gold 5416S CPU, 376GB RAM, and 8 GPUs (48GB memory each). To ensure result reliability, we repeated each experiment five times and reported the average performance.

4.2 Performance Evaluation

Table 1 presents the performance of *OpsAgent* and baselines on the OPENRCA [44] benchmark across three microservice systems. *OpsAgent* consistently attains the best average scores on both Correct and Partial, surpassing the SOTA by **46.63%** in Correct and **27.90%** in Partial. General-purpose open-source prompting frameworks (*i.e.*, CoT [41], ReAct [27], Reflexion [31]) perform unevenly, as IM is demanding even for experienced OCEs, and approaches not tailored to the task struggle with heterogeneous observability data and tightly coupled causal relations. CoT [41] uses step-wise reasoning and tends to be stable, which is why we also introduce CoT prompting in our design. ReAct [27] integrates tool use, but coordinating tools over diverse observations poses great challenges in planning. Reflexion [31] builds on ReAct by reflecting on prior reasoning paths and yields some improvements, but the gains are limited because ReAct often produces noisy and disorganized trajectories. Among LLM-based baselines, RCA-Agent [44] ranks second on average Correct/Partial by introducing an executor agent that synthesizes and runs programs. However, this design relies heavily

on the error-handling and planning capacity of a closed-source LLM (Claude 3.5 Sonnet). This dependency incurs substantial inference costs and poses privacy-exposure risks when sensitive observability and incident data are sent to third-party APIs. The DL-based baseline ART [35] achieves relatively acceptable performance as it fits the system-specific data patterns with its well-designed architecture. At the same time, we observe significant sensitivity to the underlying backbone: as the seed LLM changes, performance varies for both *OpsAgent* and the general-purpose baselines, yet under every seed LLM *OpsAgent* achieves the best Correct/Partial averages.

To quantify runtime cost, we report the average time per incident case in the final column of Table 1. *OpsAgent* processes a case in roughly 1 minute on average, demonstrating its high efficiency while maintaining strong performance. Among general-purpose open-source prompting frameworks, CoT [41] completes a case in about 20 seconds but with much lower accuracy, ReAct [27] slows due to multi-round tool calls, and Reflexion [31] is even slower as it reflects upon prior trajectories of ReAcT. RCA-Agent [44] takes close to 5 minutes per case because its executor frequently regenerates and reruns code when errors occur. ART [35] is the fastest due to lightweight neural networks and attains moderate accuracy. Overall, *OpsAgent* achieves a balanced profile across efficiency and accuracy, while operating on a local 14B model that avoids the high costs associated with closed-source APIs.

4.3 Ablation Study

To validate the contribution of *OpsAgent*'s core modules, we conduct an ablation study under different conditions: **A1**: without data processor, **A2**: without cross-review mechanism, **A3**: without reflection, **A4**: without PPO fine-tuning. As shown in Table 2, *OpsAgent* (trained with 60% cases) outperforms all the variants and the results reveal three key findings: (1) **Textual data descriptions are necessary for LLM reasoning (A1)**. For A1, we still filter the anomalous observability data to fit the context window, but feed them in their original form rather than as structured textual descriptions. We observed that LLMs struggle to deal with massive numerical inputs, leading to unstable reasoning, while structured textual descriptions expose salient cues and enable more reliable inference. (2) **Cross-review is critical for robust diagnosis under complex IM scenarios (A2)**. For A2, we remove the cross-review mechanism and let the three expert agents generate their diagnoses independently without reviewing or refining each other's outputs. Both Correct and Partial drop sharply, indicating that cross-review is essential because it incorporates complementary perspectives, reconciles divergent reasoning, and uncovers missing or weak evidence across agents, all of which are crucial for maintaining diagnostic quality. (3) **Internal training and external reflection are complementary for sustained gains (A3, A4)**. For A3, we fine-tune the agents with PPO only, whereas for A4 we disable PPO and rely solely on reflection-based knowledge distillation with RAG at inference. Both variants underperform the full model, with lower Correct/Partial across systems, indicating that both internal training and external reflection to be indispensable for improving performance, and in combination they deliver complementary gains as the system evolves.

Table 2: Ablation study on key components (Seed LLM: Qwen2.5-14B-Instruct-1M). Results are dataset-averaged Correct/Partial (%).

Variant	Average	
	Correct	Partial
A1: w/o Data Processor	2.26	6.02
A2: w/o Cross-review	6.77	21.80
A3: w/o Reflection	10.53	26.32
A4: w/o PPO	12.78	33.08
OpsAgent (0%)	8.27	27.07
OpsAgent (60%)	**16.54**	**41.35**

Notes: Percentages indicate the fraction of incident cases used for the self-evolution mechanism (PPO and reflection). "*OpsAgent* (0%)" disables self-evolution, "*OpsAgent* (60%)" applies self-evolution using 60% of the cases.

Table 3: Self-evolution capability under different training budgets.

Seed LLM	Training (%)	Avg	
		Correct	Partial
Qwen2.5-14B-Instruct-1M	0	8.27	27.06
	20	12.03	33.83
	40	14.29	38.35
	60	**16.54**	**41.35**

4.4 Self-evolution Evaluation

We further evaluate self-evolution to verify continual capability growth. We vary the number of training cases used for updates after deployment and always report Correct and Partial on a held-out test set. As shown in Table 3, *OpsAgent* exhibits a steady increase as more incidents are processed. The gains arise from two complementary sources: internal parameter updates via PPO that align the agents with the reward signal, and external reflection that distills experience into a knowledge base for retrieval during inference. This pattern mirrors how OCEs accumulate expertise through practice. The results validate the self-evolution mechanism and support *OpsAgent*'s suitability for long-term deployment.

5 Discussion

5.1 Deployment

We deployed *OpsAgent* as a real-time incident diagnostic system in Lenovo's production environment, using *Qwen2.5-14B-Instruct-1M* as the seed LLM. The environment spans on the order of 2.5×10^4 infrastructure instances across data-center facilities, hardware, virtualization/cloud platforms, middleware, databases, and application/service layers. Its observability stack includes *VictoriaMetrics* for metrics and *OpenTelemetry* for logs and traces. Over 53 days, *OpsAgent* processed 10,492 incidents in production environment.

Effectiveness. Over the 53-day deployment, *OpsAgent* achieved an overall diagnostic accuracy of 84.09%. Here, accuracy is defined

by whether the final validated report correctly identifies the root-cause reason and component, as verified through OCE-confirmed mitigation. This result is higher than those in Table 1 because production queries provide richer incident context (*e.g.*, alert information) than OPENRCA [44], which narrows the search space and improves diagnostic fidelity. In traditional operations, resolving one incident typically required the collaboration of 3 experienced OCEs for about 2.5 hours, whereas *OpsAgent* reduced to 126 seconds. Routine and localized incidents accounted for roughly 70% of all incidents, mainly involving recurring types like *Disk Space Exhaustion*, and *Proxy Misconfiguration*. For these incidents, *OpsAgent* achieved 97% accuracy with an average diagnosis time of only 30 seconds, demonstrating high efficiency and effectiveness. The remaining incidents characterized by cross-component interactions, ambiguous symptoms, or incomplete evidence were challenging, for which *OpsAgent* was allowed to perform up to three rounds of re-analysis yet still achieved only 54% accuracy. Nevertheless, even when the diagnosis was wrong, the generated *Root Cause Report* still provided OCEs with useful evidence and plausible hypotheses, eliminating the need for purely manual investigation over massive observability data. Therefore, *OpsAgent* still substantially reduced the effort required from OCEs and accelerated incident mitigation in difficult cases. Importantly, we observed clear benefits from self-evolution during deployment. Some complex incidents that initially required multiple re-analysis rounds could later be diagnosed correctly in a single attempt, and some failure types that were initially unresolved became diagnosable when they reappeared. This trend indicates that *OpsAgent* continuously strengthens its diagnostic capability after deployment, especially for recurring complex incident patterns.

5.2 Case Study

To illustrate how *OpsAgent* operates after deployment, we present a representative case study from Lenovo's production environment. All service names and incident details have been anonymized for security and privacy reasons. As shown in Fig. 6, the diagnostic process began when an incident ticket was automatically generated for an order-processing application, reporting repeated request failures between 09:10 and 09:20. (1) After interpreting the ticket, *OpsAgent* retrieved multimodal observations within this time range and converted them into unified textual descriptions for analysis. In this case, it identified a sharp increase in disk utilization on instance `db-03`, repeated "No space left on device" errors in the logs of `db-03`, and high-latency spans concentrated on calls from `order-service` to `db-03`. (2) Based on these evidences, the anomaly detection agent inferred that the earliest abnormal timestamp was 09:12, the failure triage agent concluded that the incident was caused by disk space exhaustion, and the root cause localization agent identified `db-03` as the faulty instance. (3) During cross-review, the failure triage agent agreed with the localization result because the storage-related log errors and trace bottlenecks both pointed to `db-03`; the anomaly detection agent further confirmed that the disk-utilization surge on `db-03` appeared before the request failures; and the localization agent ruled out `order-service` itself because its latency increase occurred only after calls to `db-03` became slow. After integrating

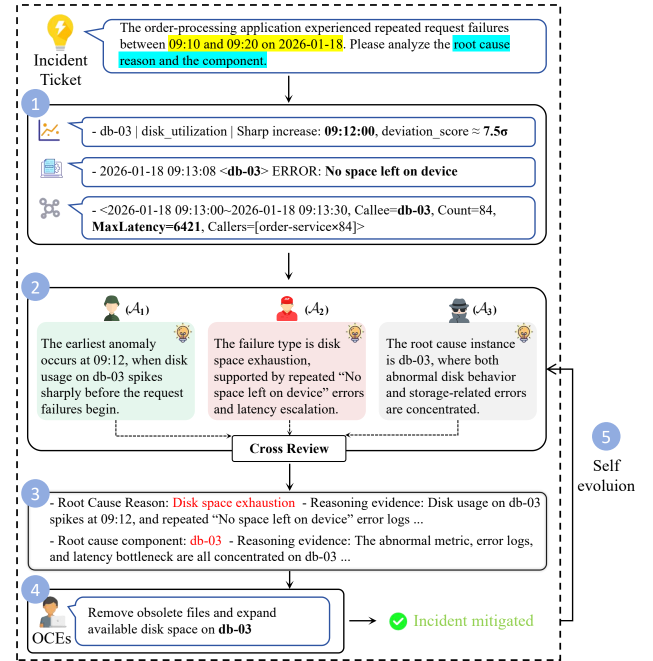

Figure 6: An illustrative case.

these consistent reviews, *OpsAgent* generated a final *Root Cause Report*, identifying disk space exhaustion on `db-03` as the root cause. (4) The report was delivered to the frontline OCE, who cleaned obsolete files and expanded the available disk space on `db-03`, after which the service recovered successfully. (5) Since the mitigation validated the diagnosis, the confirmed case was fed back to *OpsAgent* for self-evolution, where it was used to improve the agents and distill reusable experience for future similar incidents.

5.3 Interpretability in Deployment

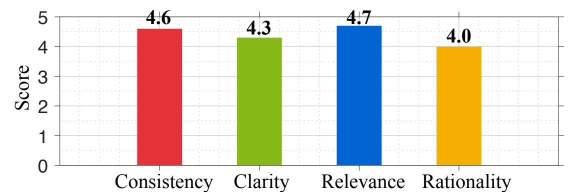

Figure 7: Interpretability assessment in deployment. Mean scores given by frontline OCEs on four dimensions.

To understand whether *OpsAgent*'s reports are practically useful after deployment, we asked three frontline OCEs who use the system in production to assess its interpretability. Specifically, we randomly sampled 150 incidents from the 10,492 incidents, and the three OCEs rated the generated root cause reports on four dimensions, *i.e.*, *consistency*, *clarity*, *relevance*, and *rationality*, using a 1-5 scale. As shown in Fig. 7, *OpsAgent* receives consistently favorable ratings overall, with mean scores of 4.6, 4.3, 4.7, and 4.0, respectively. These results suggest that the reports are generally coherent, easy to follow, and well aligned with the incident context. Overall,

they demonstrate that *OpsAgent* provides strong interpretability in real deployment and serves as a practical diagnostic assistant for frontline OCEs.

5.4 Lessons Learned

5.4.1 Diagnostic Interpretability. After deployment, we found that a diagnosis alone was often insufficient for OCEs to trust and act upon the system's output. In practice, OCEs need clear evidence and a traceable reasoning process to understand why a conclusion is reached. We therefore observed that interpretability is not merely a desirable property, but a prerequisite for practical adoption: when the diagnosis is correct, it increases trust and actionability; even when it is incorrect, the exposed reasoning process and collected evidence can still substantially assist OCEs in incident mitigation.

5.4.2 Self-Feedback for Sustainability. We also learned that static IM approaches are difficult to sustain in dynamic production environments, where previously unseen incident types continuously emerge. Without an effective feedback mechanism, diagnostic performance can gradually degrade as the environment evolves. By feeding validated deployment outcomes back into *OpsAgent* for self-evolution, the system gradually adapts to new failure patterns and reduces the need for repeated manual maintenance, which is critical for long-term deployability.

6 Related work

6.1 DL-based IM approaches.

DL-based approaches leverage neural networks to extract failure patterns from observability data and predict root causes, employing either supervised [15, 51] or self-supervised [33, 35] learning. Early studies focused on unimodal data sources, such as metrics [19, 21, 22, 38, 39, 42], logs [17, 32, 43, 55], or traces [18, 45, 48, 58], which limited their ability to capture a comprehensive view of system states. Recent advances have adopted multimodal fusion, exemplified by ART [35], DeepHunt [33], and DiagFusion [51], which integrate heterogeneous observability data to improve diagnostic accuracy and efficiency. Despite these improvements, DL-based IM methods face three key limitations. First, models often overfit system-specific patterns, leading to poor generalization across heterogeneous and evolving microservice systems. Second, their black-box nature hinders interpretability, offering little insight into the reasoning process behind root cause predictions, which reduces trustworthiness for OCEs. Finally, these models are trained in a static manner and cannot accumulate diagnostic experience over time, making them ill-suited for sustained deployment in dynamic microservice environments.

6.2 LLM-based IM approaches.

With the rapid progress of LLMs, recent studies have begun to explore their potential in IM. Ahmed et al. [3] fine-tuned GPT-3.X on domain corpora for direct root cause prediction from incident titles and summaries. RCACopilot [5] and Zhang et al. [54] use in-context learning, retrieving semantically similar past incidents as examples for few-shot prediction. While these approaches demonstrate the feasibility of LLMs for IM, they also exhibit critical limitations. First, most methods rely on surface-level similarity matching or prompt engineering, rather than performing grounded reasoning over raw observability data, which weakens alignment with actual system behavior. Second, their outputs typically lack transparent reasoning chains, offering little interpretability or auditability for OCEs in high-stakes operational scenarios. Finally, they are often built on closed-source proprietary models such as GPT-4, resulting in high inference costs and strong dependency on external APIs, which poses substantial barriers to cost-efficient and trustworthy deployment in real-world environments. Building on LLMs, recent efforts have explored multi-agent system (MAS)-based IM, leveraging specialized agents for collaborative reasoning. For instance, mABC [53] adopts a blockchain-inspired decentralized voting mechanism among LLM-based agents for collective root cause inference; TrioXpert [34] proposes a multi-expert architecture integrating modality-specific preprocessing and agent collaboration for IM; D-Bot [57] focuses on database diagnosis, leveraging agent collaboration with specialized roles (*e.g.*, CPU, Network) to generate timely diagnostic reports. In parallel, OPENRCA [44] introduces the first benchmark for natural-language-driven IM. Notably, even top methods using closed-source LLMs (*e.g.*, Claude 3.5 Sonnet, GPT-4o, Gemini 1.5 Pro) only achieve 11.34% accuracy, highlighting IM's inherent difficulty and current approach limitations.

Overall, existing IM methods lack generalization, interpretability, cost-efficiency and self-evolution (Table 4), while *OpsAgent* uniquely unifies these capabilities, enabling long-term deployment in real-world microservice environments.

Table 4: Comparison of IM categories.

Categories	Gen?	Int?	Cos?	Evo?
DL-based [33, 35, 51]	○	○	●	○
LLM-based [3, 5, 54]	◐	◐	○	○
MAS-based [34, 53, 57]	◐	●	●	○
OpsAgent	●	●	●	●

Notes: "Gen?": Generalizable? "Int?": Interpretable? "Cos?": Cost-Efficient? "Evo?": Self-Evolving? "●": full support, "◐": partial support, "○": no support.

7 Conclusion

This work introduces *OpsAgent*, a lightweight, self-evolving multi-agent system for IM. We incorporate a training-free data processor to handle massive, heterogeneous observability data, a multi-agent collaboration framework that renders diagnostic inference transparent and auditable, and a dual self-evolution mechanism integrating internal model updates with external experience accumulation. With this design, *OpsAgent* is generalizable, interpretable, cost-efficient, and self-evolving, making it a practically deployable and sustainable solution for real-world microservice systems. We believe that the concept of constructing a multi-agent system with a well-designed data processor, a multi-agent collaboration framework, and a self-evolution mechanism can generalize to other complex scenarios that involve massive, heterogeneous data.

8 Data Availability Statement

All code, prompts and data are available in the repo: https://anonymous.4open.science/r/OpsAgent-CCC0.

References

[1] 2025. Incident Report of google cloud outage. https://status.cloud.google.com/incidents/ow5i3PPK96RduMcb1SsW.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[3] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending root-cause and mitigation steps for cloud incidents using large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1737–1749.

[4] Mohammadmehdi Ataei, Hyunmin Cheong, Daniele Grandi, Ye Wang, Nigel Morris, and Alexander Tessier. 2025. Elicitron: A large language model agent-based simulation framework for design requirements elicitation. *Journal of Computing and Information Science in Engineering* 25, 2 (2025), 021012.

[5] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, et al. 2024. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 674–688.

[6] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. 2017. Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering* (2017), 195–216.

[7] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130* (2024).

[8] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, et al. 2023. Retrieval-Augmented Generation for Large Language Models: A Survey. *CoRR* (2023).

[9] Yongqi Han, Qingfeng Du, Ying Huang, Jiaqi Wu, Fulong Tian, and Cheng He. 2024. The potential of one-shot failure root cause analysis: Collaboration of the large language model and small classifier. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 931–943.

[10] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*. IEEE, 33–40.

[11] Sihao Hu, Tiansheng Huang, Fatih İlhan, Selim Furkan Tekin, and Ling Liu. 2023. Large language model-powered smart contract vulnerability detection: New perspectives. In *2023 5th IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*. IEEE, 297–306.

[12] Md Ashraful Islam, Mohammed Eunus Ali, and Md Rizwan Parvez. 2024. MapCoder: Multi-Agent Code Generation for Competitive Problem Solving. In *Annual Meeting of the Association of Computational Linguistics 2024*. Association for Computational Linguistics (ACL), 4912–4944.

[13] Stephen Thorne Arup Chakrabarti Jian Ma Jessie Yang Jennifer Mace, Jelena Oertel. [n. d.]. SRE Book, Chapter 9: Incident Response. https://sre.google/workbook/incident-response/.

[14] Dongming Jin, Zhi Jin, Xiaohong Chen, and Chunhui Wang. 2024. Mare: Multi-agents collaboration framework for requirements engineering. *arXiv preprint arXiv:2405.03256* (2024).

[15] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R Lyu. 2023. Eadro: An end-to-end troubleshooting framework for microservices on multi-source data. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1750–1762.

[16] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.

[17] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. 2022. SwissLog: Robust anomaly detection and localization for interleaved unstructured logs. *IEEE Transactions on Dependable and Secure Computing* 20, 4 (2022), 2762–2780.

[18] Zeyan Li, Junjie Chen, Rui Jiao, Nengwen Zhao, Zhijun Wang, Shuwei Zhang, Yanjun Wu, Long Jiang, Leiqin Yan, Zikai Wang, et al. 2021. Practical root cause localization for microservice systems via trace analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 1–10.

[19] Zeyan Li, Nengwen Zhao, Mingjie Li, Xianglin Lu, Lixin Wang, Dongdong Chang, Xiaohui Nie, Li Cao, Wenchi Zhang, Kaixin Sui, et al. 2022. Actionable and interpretable fault localization for recurring failures in online service systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 996–1008.

[20] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-Eval: NLG Evaluation using Gpt-4 with Better Human Alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 2511–2522.

[21] Meng Ma, Weilan Lin, Disheng Pan, and Ping Wang. 2019. Ms-rank: Multi-metric and self-adaptive root cause diagnosis for microservice applications. In *2019 IEEE International Conference on Web Services (ICWS)*. IEEE, 60–67.

[22] Meng Ma, Jingmin Xu, Yuan Wang, Pengfei Chen, Zonghua Zhang, and Ping Wang. 2020. Automap: Diagnose your microservice-based web applications automatically. In *Proceedings of The Web Conference 2020*. 246–258.

[23] Yuan Meng, Shenglin Zhang, Yongqian Sun, Ruru Zhang, Zhilong Hu, Yiyin Zhang, Chenyang Jia, Zhaogang Wang, and Dan Pei. 2020. Localizing failure root causes in a microservice through causality inference. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[24] Changhua Pei, Zexin Wang, Fengrui Liu, Zeyan Li, Yang Liu, Xiao He, Rong Kang, Tieying Zhang, Jianjun Chen, Jianhui Li, et al. 2025. Flow-of-Action: SOP Enhanced LLM-Based Multi-Agent System for Root Cause Analysis. In *Companion Proceedings of the ACM on Web Conference 2025*. 422–431.

[25] Hongjin Qian, Zheng Liu, Peitian Zhang, Kelong Mao, Defu Lian, Zhicheng Dou, and Tiejun Huang. 2025. Memorag: Boosting long context processing with global memory-enhanced retrieval augmentation. In *Proceedings of the ACM on Web Conference 2025*. 2366–2377.

[26] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. http://arxiv.org/abs/1908.10084

[27] Devjeet Roy, Xuchao Zhang, Rashi Bhave, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. 2024. Exploring llm-based agents for root cause analysis. In *Companion proceedings of the 32nd ACM international conference on the foundations of software engineering*. 208–219.

[28] Gerard Salton and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Information processing & management* 24, 5 (1988), 513–523.

[29] Bhaskarjit Sarmah, Dhagash Mehta, Benika Hall, Rohan Rao, Sunil Patel, and Stefano Pasquali. 2024. Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction. In *Proceedings of the 5th ACM International Conference on AI in Finance*. 608–616.

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[31] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems* 36 (2023), 8634–8652.

[32] Yicheng Sui, Yuzhe Zhang, Jianjun Sun, Ting Xu, Shenglin Zhang, Zhengdan Li, Yongqian Sun, Fangrui Guo, Junyu Shen, Yuzhi Zhang, et al. 2023. Logkg: Log failure diagnosis through knowledge graph. *IEEE Transactions on Services Computing* 16, 5 (2023), 3493–3507.

[33] Yongqian Sun, Zihan Lin, Binpeng Shi, Shenglin Zhang, Shiyu Ma, Pengxiang Jin, Zhenyu Zhong, Lemeng Pan, Yicheng Guo, and Dan Pei. 2025. Interpretable failure localization for microservice systems based on graph autoencoder. *ACM Transactions on Software Engineering and Methodology* 34, 2 (2025), 1–28.

[34] Yongqian Sun, Yu Luo, Xidao Wen, Yuan Yuan, Xiaohui Nie, Shenglin Zhang, Tong Liu, and Xi Luo. 2025. TrioXpert: An automated incident management framework for microservice system. *arXiv preprint arXiv:2506.10043* (2025).

[35] Yongqian Sun, Binpeng Shi, Mingyu Mao, Minghua Ma, Sibo Xia, Shenglin Zhang, and Dan Pei. 2024. Art: A unified unsupervised framework for incident management in microservice systems. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 1183–1194.

[36] Maryam Taeb, Amanda Swearngin, Eldon Schoop, Ruijia Cheng, Yue Jiang, and Jeffrey Nichols. 2024. Axnav: Replaying accessibility tests from natural language. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–16.

[37] Lei Tao, Shenglin Zhang, Zedong Jia, Jinrui Sun, Minghua Ma, Zhengdan Li, Yongqian Sun, Canqun Yang, Yuzhi Zhang, and Dan Pei. 2024. Giving every modality a voice in microservice failure diagnosis via multimodal adaptive optimization. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 1107–1119.

[38] Dongjie Wang, Zhengzhang Chen, Yanjie Fu, Yanchi Liu, and Haifeng Chen. 2023. Incremental causal graph learning for online root cause analysis. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*. 2269–2278.

[39] Ping Wang, Jingmin Xu, Meng Ma, Weilan Lin, Disheng Pan, Yuan Wang, and Pengfei Chen. 2018. Cloudranger: Root cause identification for cloud native systems. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 492–502.

[40] Zexin Wang, Jianhui Li, Minghua Ma, Ze Li, Yu Kang, Chaoyun Zhang, Chetan Bansal, Murali Chintalapati, Saravan Rajmohan, Qingwei Lin, et al. 2024. Large language models can provide accurate and interpretable incident triage. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 523–534.

[41] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[42] Canhua Wu, Nengwen Zhao, Lixin Wang, Xiaoqin Yang, Shining Li, Ming Zhang, Xing Jin, Xidao Wen, Xiaohui Nie, Wenchi Zhang, et al. 2021. Identifying root-cause metrics for incident diagnosis in online service systems. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 91–102.

[43] Yuxia Xie, Kai Yang, and Pan Luo. 2021. Logm: Log analysis for multiple components of hadoop platform. *IEEE Access* 9 (2021), 73522–73532.

[44] Junjielong Xu, Qinan Zhang, Zhiqing Zhong, Shilin He, Chaoyun Zhang, Qingwei Lin, Dan Pei, Pinjia He, Dongmei Zhang, and Qi Zhang. 2025. OpenRCA: Can large language models locate the root cause of software failures?. In *The Thirteenth International Conference on Learning Representations*.

[45] Jingjing Yang, Yuchun Guo, Yishuai Chen, and Yongxiang Zhao. 2023. TraceNet: Operation aware root cause localization of microservice system anomalies. In *2023 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 758–763.

[46] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

[47] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. 2023. Nezha: Interpretable fine-grained root causes analysis for microservices on multi-modal observability data. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 553–565.

[48] Guangba Yu, Zicheng Huang, and Pengfei Chen. 2023. TraceRank: Abnormal service localization with dis-aggregated end-to-end tracing data in cloud native systems. *Journal of Software: Evolution and Process* 35, 10 (2023), e2413.

[49] Daoguang Zan, Ailun Yu, Wei Liu, Dong Chen, Bo Shen, Wei Li, Yafen Yao, Yongshun Gong, Xiaolin Chen, Bei Guan, et al. 2024. CodeS: Natural Language to Code Repository via Multi-Layer Sketch. *CoRR* (2024).

[50] Huan Zhang, Wei Cheng, Yuhan Wu, and Wei Hu. 2024. A pair programming framework for code generation via multi-plan exploration and feedback-driven refinement. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 1319–1331.

[51] Shenglin Zhang, Pengxiang Jin, Zihan Lin, Yongqian Sun, Bicheng Zhang, Sibo Xia, Zhengdan Li, Zhenyu Zhong, Minghua Ma, Wa Jin, et al. 2023. Robust failure diagnosis of microservice system through multimodal data. *IEEE Transactions on Services Computing* 16, 6 (2023), 3851–3864.

[52] Shenglin Zhang, Sibo Xia, Wenzhao Fan, Binpeng Shi, Xiao Xiong, Zhenyu Zhong, Minghua Ma, Yongqian Sun, and Dan Pei. 2024. Failure diagnosis in microservice systems: A comprehensive survey and analysis. *ACM Transactions on Software Engineering and Methodology* (2024).

[53] Wei Zhang, Hongcheng Guo, Jian Yang, Zhoujin Tian, Yi Zhang, Yan Chaoran, Zhoujun Li, Tongliang Li, Xu Shi, Liangfan Zheng, et al. 2024. mABC: Multi-Agent Blockchain-inspired Collaboration for Root Cause Analysis in Micro-Services Architecture. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. 4017–4033.

[54] Xuchao Zhang, Supriyo Ghosh, Chetan Bansal, Rujia Wang, Minghua Ma, Yu Kang, and Saravan Rajmohan. 2024. Automated root causing of cloud incidents using in-context learning with GPT-4. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 266–277.

[55] Xu Zhang, Yong Xu, Si Qin, Shilin He, Bo Qiao, Ze Li, Hongyu Zhang, Xukun Li, Yingnong Dang, Qingwei Lin, et al. 2021. Onion: identifying incident-indicating logs for cloud systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1253–1263.

[56] Lecheng Zheng, Zhengzhang Chen, Jingrui He, and Haifeng Chen. 2024. MULAN: multi-modal causal structure learning and root cause analysis for microservice systems. In *Proceedings of the ACM Web Conference 2024*. 4107–4116.

[57] Xuanhe Zhou, Guoliang Li, Zhaoyan Sun, Zhiyuan Liu, Weize Chen, Jianming Wu, Jiesi Liu, Ruohang Feng, and Guoyang Zeng. 2024. D-Bot: Database Diagnosis System using Large Language Models. *Proceedings of the VLDB Endowment* 17, 10 (2024), 2514–2527.

[58] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. 2019. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In *Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 683–694.