From Observability Data to Diagnosis: An Evolving Multi-agent System for Incident Management in Cloud Systems

Yu Luo
Nankai University
Tianjin, China
luoyu@mail.nankai.edu.cn

Jiamin Jiang
Nankai University
Tianjin, China
jiangjiamin@mail.nankai.edu.cn

Jingfei Feng
Nankai University
Tianjin, China
2320250836@mail.nankai.edu.cn

Lei Tao
Nankai University
Tianjin, China
leitao@mail.nankai.edu.cn

Qingliang Zhang
Nankai University
Tianjin, China
zhangqingliang@mail.nankai.edu.cn

Xidao Wen
Bizseer
Beijing, China
wenxidao@bizseer.com

Yongqian Sun *
Nankai University
Tianjin, China
sunyongqian@nankai.edu.cn

Shenglin Zhang
Nankai University
Tianjin, China
zhangsl@nankai.edu.cn

Jielong Huang
Kuaishou Technology
Hangzhou, China
zhangsong08@kuaishou.com

Nan Qi
Kuaishou Technology
Hangzhou, China
qinan03@kuaishou.com

Dan Pei
Tsinghua University
Beijing, China
peidan@tsinghua.edu.cn

***Abstract*—Incident management (IM) is central to the reliability of large-scale cloud 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 [1] 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 cloud systems.**

***Index Terms*—Heterogeneous Observability Data, Multi-agent System, Self-evolution, Incident Management**

I. INTRODUCTION

Cloud systems have become the de facto platform for modern software services, with wide deployments across industries such as IT, government, and finance [2]. However, incidents (*e.g.*, service disruptions and outages) [2], [3] are inevitable due to the complexity of cloud 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 [4]. 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 [2], [5].

Traditionally, on-call engineers (OCEs) manually inspect metrics, logs, and traces to identify the root cause when incidents occur [2]. 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 [3], [6].

To alleviate this burden, automated IM with AI techniques has been extensively explored, falling into two main categories: deep learning (DL)-based IM [3], [7]–[12] and large language model (LLM)-based IM [2], [5], [13]–[17]. Deep learning-based approaches leverage neural networks trained via supervised [8], [12] and self-supervised learning [3], [18] 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

* Yongqian Sun is the corresponding author.
Work done during Yu Luo's internship at Kuaishou Technology

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 [2], [13], [14]. However, 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 [2], [14]; second, without stepwise 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 [2], [13], [14]. 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 [19]–[21], quality assurance [22], [23], and requirements analysis [24], [25]. 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 cloud systems? Cloud 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 [17]. 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 cloud 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 [26] and Flow-of-Action [27], 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.

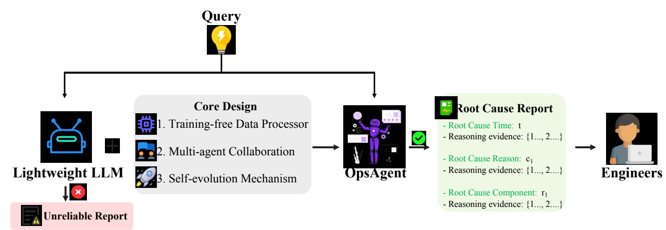

Fig. 1. **From lightweight LLM to MAS-based IM**. *OpsAgent* turns a lightweight LLM into a deployable and sustainable IM system by incorporating (1) training-free data processor (Section III-B), (2) multi-agent collaboration (Section III-C), and (3) self-evolution mechanism (Section III-D).

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 cloud 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 cloud systems.

3) We evaluate *OpsAgent* on the OPENRCA [1] benchmark and demonstrate state-of-the-art performance. Through comprehensive experiments, we also prove its generalizability, interpretability, cost-efficiency and capability of self-evolving. To ensure reproducibility, we release all code, prompts, configurations, and data [1].

II. BACKGROUND

A. Observability in Cloud System

Cloud systems commonly consist of large-scale, distributed services that are independently deployable and elastically scalable, typically realized via microservice-style decomposition [28]. Their operational state evolves rapidly due to frequent releases, autoscaling, and heterogeneous runtime substrates (*e.g.*, VMs, containers, serverless platforms). In this setting, effective monitoring and diagnosis hinge on three fundamental types of observability data that expose system behavior from complementary angles:

- **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.

B. Reinforcement Learning

Reinforcement Learning (RL) is a machine learning paradigm that models learning as an interaction between the agent and the environment. The agent learns to make decisions by observing the current state of the environment, taking an action, and receiving feedback in the form of reward signal. Through repeated interactions, the agent learns an optimal policy that maximizes cumulative reward.

Among various RL algorithms, **proximal policy optimization (PPO)** has emerged as one of the most widely adopted methods due to its strong empirical performance and training stability [29], [30]. PPO is a policy-gradient RL algorithm designed to improve training stability and sampling efficiency by limiting policy update magnitudes through a clipped objective function. This prevents drastic updates that may cause training collapse and balances exploration with exploitation.

C. Retrieval-Augmented Generation

While LLMs demonstrate exceptional capabilities in text generation and semantic comprehension, they exhibit two significant limitations [31]. First, their knowledge remains static, rendering them unable to incorporate recent events or dynamically updated information. Second, their domain expertise is inherently limited, lacking the depth required for specialized vertical domains.

To address these shortcomings, the **Retrieval-Augmented Generation (RAG)** [32]–[34] framework was developed to incorporate external knowledge into the generation process, enabling the model to generate content based on the retrieval results. This framework retains the language generation capabilities of LLMs while enhancing its accuracy and domain adaptability of outputs through external knowledge injection, making it a key technology for enhancing the practicality of large models in professional scenarios.

D. 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. Formally, 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 $\mathbb{C} = \{c_1, c_2, \dots, c_n\}$. Finally, the RCL task localizes the most likely faulty component r from the set of root cause components $\mathbb{R} = \{r_1, r_2, \dots, r_m\}$, encompassing entities across different system levels (*e.g.*, nodes, services, pods). The overall objective is to learn a mapping function $\mathcal{F} : (q, \mathcal{X}) \rightarrow (t, c, r)$.

III. METHODOLOGY

A. Overview

OpsAgent consists of three modules: (1) training-free data processor (Section III-B), (2) multi-agent collaboration (Section III-C), and (3) self-evolution mechanism (Section III-D). Each module directly targets one of the challenges in Section I. *(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

[1] https://anonymous.4open.science/r/OpsAgent-0F48/

low. This design supports a practically deployable and sustainable solution for real-world cloud systems.

B. Training-free Data Processor

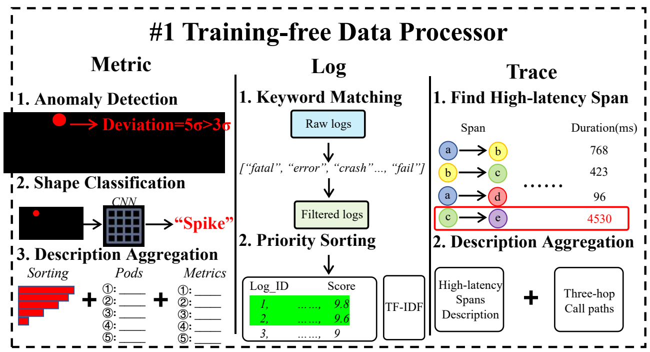

Fig. 2. **Training-free Data Processor**. The processor handles three types of observability data separately: metrics (left), logs (middle), and traces (right).

Unlike DL-based IM methods that require large-scale data to learn feature distributions [3], [8], [12], [18], 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 cloud 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.

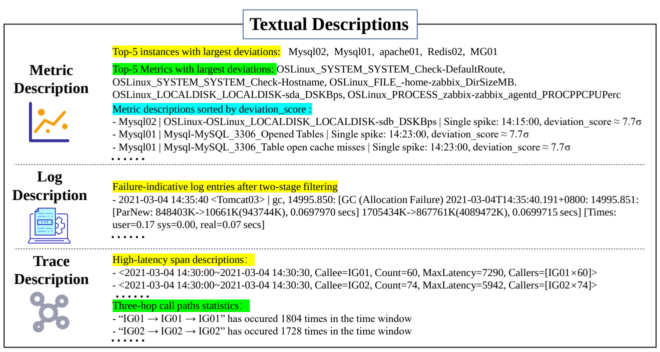

Fig. 3. Illustrative example of data descriptions.

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 [35]. *(3) Description Aggregation.* To provide agents with compact yet informative inputs, anomalies are transformed into textual descriptions of the form $\langle$*service_instance, metric_name, anomaly_pattern: timestamp, deviation_score* = $k\sigma\rangle$. 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 [36] to normalize variable content, then rank templates by TF–IDF [37] 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 [38]) 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 $\langle$*time_interval, callee, count, max_latency, callers*$\rangle$, which highlight when and where latency concentrates. Second, to reveal recurrent patterns that may indicate systemic bottlenecks, we extract three-hop call paths (*grandparent* $\rightarrow$ *caller* $\rightarrow$ *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.

C. 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

[2]We trained this CNN to classify the anomaly shapes, with its design corresponding to the PatternMatcher method [35].

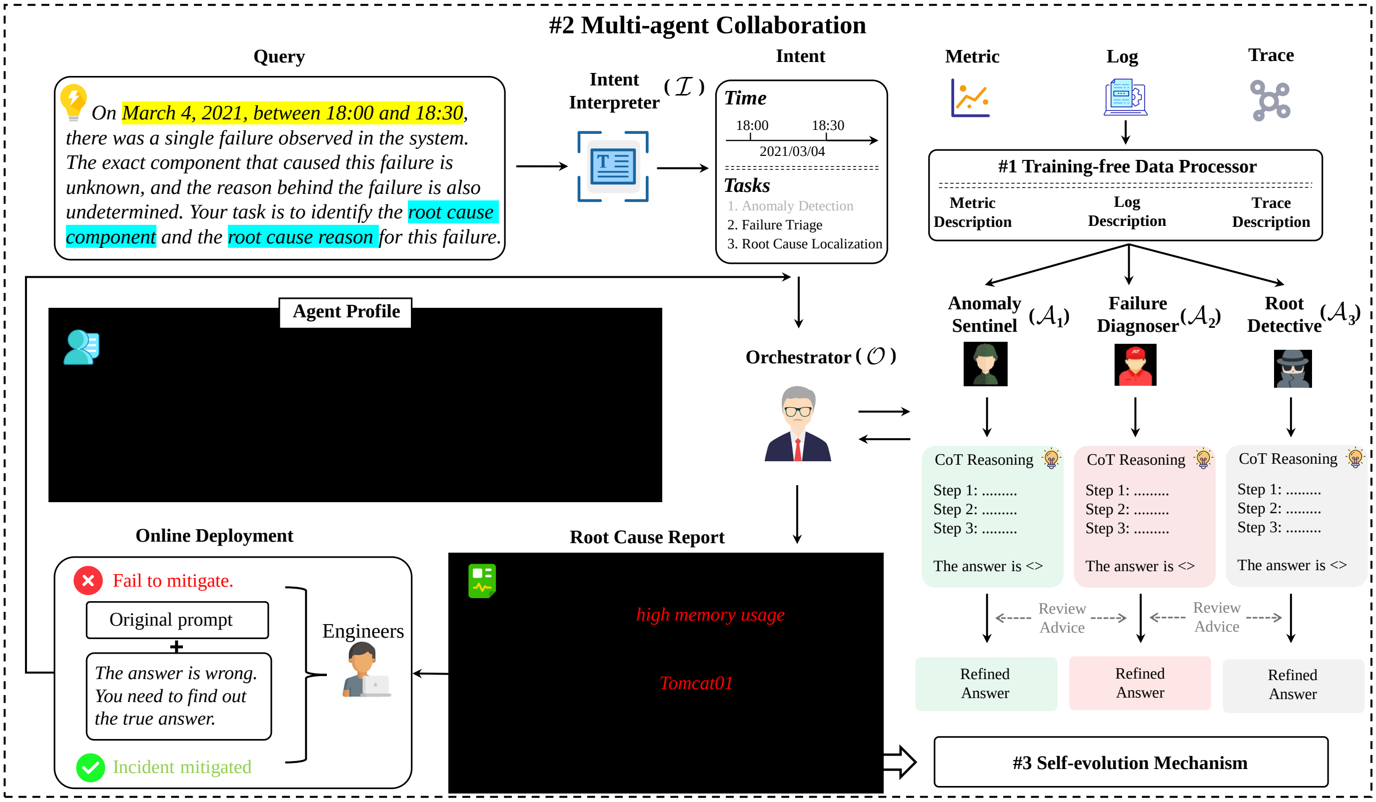

Fig. 4. **Multi-agent Collaboration**. Agents with predefined roles (via agent profile) cooperate under a structured **workflow** and **cross-review mechanism** to enhance reasoning from multiple perspectives. The Root Cause Report not only guides **online incident mitigation** but also feeds **offline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

online deployment and offline training use the *Root Cause Report* to close the loop and sustain capability growth.

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 [39] to produce task-specific answers together with their stepwise rationales. 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 [40], 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*).

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 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) 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 through 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 explicitly 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 III-D) 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.

D. 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.

1) PPO Training: To enhance the diagnostic reasoning capability of *OpsAgent*, we adopt PPO [29], a stable reinforcement learning algorithm that aligns well with capability-centric learning paradigms. 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 final 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 the *Orchestrator*, which is well suited for this role since evaluating reasoning evidence is generally more tractable than generating it [41], and as the coordinator, it can provide unbiased judgments across agents (*i.e.*, $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$). It scores each reasoning chain along four dimensions—consistency, clarity, relevance, and rationality—on a 0–5 scale. In a human validation over 25 cases (4 dimensions per case, 100 in total), *92%* of the *Orchestrator*'s ratings were deemed acceptable by human raters, indicating strong alignment with human assessment. We also found the scoring process to be stable and consistent across iterations, supporting its use as a lightweight yet reliable reward model. 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.

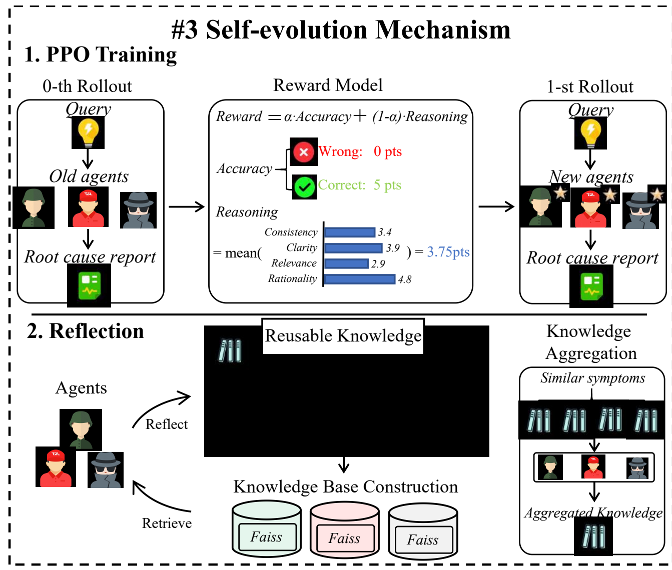

Fig. 5. **Self-evolution Mechanism**. Internally, agents are fine-tuned via PPO training with a carefully designed reward model (top). Externally, a reflection process distills reusable knowledge into a task-specific knowledge base, which is later leveraged through RAG for knowledge injection (bottom).

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 [42] 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 using this symptom key 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 [43].

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 progressively enhances its diagnostic expertise. This dual-path evolution directly addresses (*C3*), enabling sustained capability growth in MAS-based IM and mirroring how OCEs refine their skills through both practice and knowledge accumulation.

IV. Experiments

A. Experimental Setup

1) Datasets: We conducted extensive experiments on the OPENRCA [1] benchmark dataset which consists of 335 incident cases collected from 3 heterogeneous software systems (*i.e.*, Telecom, Bank, Market) deployed in the real-world cloud systems, accompanied by over 68 GB of de-identified 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 consistently 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.

2) Baselines: We compare *OpsAgent* against five frameworks: two designed specifically for IM (*i.e.*, ART [3] and RCA-Agent [1]), and three general-purpose open-source frameworks (*i.e.*, CoT [39], ReAct [46], Reflexion [45]). For DL-based IM approaches, we select ART [3] due to its state-of-the-art performance, while other methods such as DeepHunt [18] and DiagFusion [12] 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 [1], the official OPENRCA diagnostic framework and the top-performing baseline on its benchmark. We evaluate it with Claude 3.5 Sonnet as the seed LLM to match the original configuration and because it delivered the strongest reported results. We exclude other LLM-based IM methods, specifically mABC [17], ICL_RCA [14], COMET [15], due to their lack of coverage for all required tasks in our scenario, and RCACopilot [2], 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 [39], ReAct [46], and Reflexion [45]—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*.

3) Evaluation Metrics: As described in Section II-D, 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

$$Correct = \frac{num_c}{N}, \qquad Partial = \frac{num_p}{N}$$

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) 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 NVIDIA RTX A6000 GPUs (48GB memory each). To ensure result reliability, we repeated each experiment five times and reported the average performance.

B. Performance Evaluation

Table I presents the performance of *OpsAgent* and baselines on the OPENRCA [1] benchmark across three cloud systems. *OpsAgent* consistently attains the best average scores on both Correct and Partial, surpassing the SOTA by 46.63% in the Correct metric and 27.90% in the Partial metric. General-purpose open-source prompting frameworks (*i.e.*, CoT [39], ReAct [44], Reflexion [45]) 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 [39] uses step-by-step reasoning and tends to be stable, which is why we also introduce CoT prompting in our design. ReAct [44] integrates tool use, but coordinating tools over diverse observations poses great challenges in planning. Reflexion [45] 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 [1] ranks second on average Correct/Partial by introducing an executor agent that synthesizes and runs programs. However, this design

TABLE I
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**	198.22
	CoT [39]	10.00	20.00	0.00	1.85	5.08	13.55	1.50	9.77	16.83
	ReAct [44]	5.00	20.00	0.00	1.85	0.00	5.08	0.75	6.01	164.24
	Reflexion [45]	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	152.37
	CoT [39]	0.00	5.00	0.00	0.00	0.00	0.00	0.00	0.75	27.42
	ReAct [44]	0.00	0.00	0.00	0.00	1.69	1.69	0.75	0.75	81.09
	Reflexion [45]	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	161.07
	CoT [39]	0.00	10.00	0.00	0.00	0.00	0.00	0.00	1.50	20.33
	ReAct [44]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	48.42
	Reflexion [45]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100.05
Claude 3.5 Sonnet	RCA-Agent [1]	20.00	35.00	16.67	35.19	3.39	28.81	11.28	32.33	287.71
	ART [3]	0.00	15.00	0.00	11.11	1.69	23.73	0.75	17.29	**2.27**

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 [3] 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 I. *OpsAgent* processes a case in roughly 2.5–3.3 minutes on average, reflecting the overhead of multi-agent collaboration. Among general-purpose open-source prompting frameworks, CoT [39] completes a case in about 20 seconds but with much lower accuracy, ReAct [44] slows due to multi-round tool calls, and Reflexion [45] is even slower as it reflects upon prior trajectories of ReAcT. RCA-Agent [1] takes close to 5 minutes per case because its executor frequently regenerates and reruns code when errors occur. ART [3] 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.

C. 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 multi-agent collaboration, **A3**: without reflection, **A4**: without PPO fine-tuning. As shown in Table II, *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) **Role-specialized collaboration is pivotal under complex IM scenarios (A2)**. For A2, we replace the multi-agent system with a single LLM to process the textual data descriptions, and both Correct and Partial drop sharply as the model is easily overwhelmed in complex IM scenarios. Therefore, well-designed role specialization and collaboration are essential to decompose the reasoning burden and sustain performance. (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.

D. Interpretability Evaluation

Setup. We evaluate interpretability through an expert survey conducted with three senior graduate researchers specializing

TABLE II
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 Collaboration	4.51	14.29
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, whereas "*OpsAgent* (60%)" applies self-evolution using 60% of the cases.

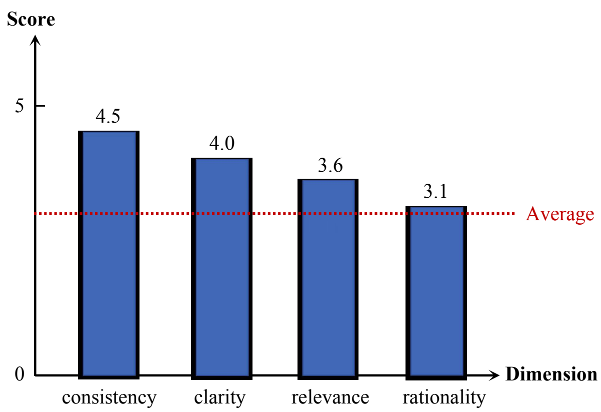

Fig. 6. **Mean scores by dimension.** Results for *OpsAgent* on the test set, trained with 60% of incident cases for self-evolution.

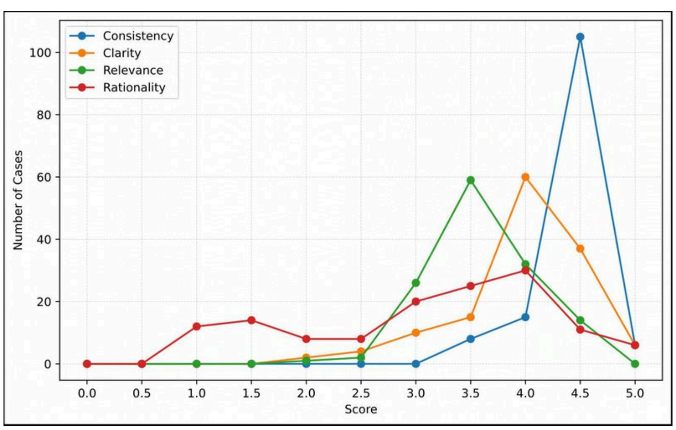

Fig. 7. **Score distributions by dimension.** Results for *OpsAgent* on the test set, trained with 60% of incident cases for self-evolution.

in AIOps, all with substantial on-call engineering experience. We assess outputs produced by *Qwen2.5-14B-Instruct-1M* on the test set, covering 133 incident cases. For each incident case, they independently reviewed the model's root cause report and assigned 0–5 ratings on four dimensions, with 3 indicating a neutral (neither good nor poor) score:

- *consistency*: internal coherence of the reasoning chain, with steps that agree with one another and with the cited evidence.
- *clarity*: readability and structure of the explanation, using precise, unambiguous language.
- *relevance*: focus on observations and arguments that are pertinent to the incident, avoiding extraneous content.
- *rationality*: logical soundness of causal inferences and sufficiency of supporting evidence.

We aggregate ratings across experts and cases, report per-dimension mean scores, and analyze the resulting score distributions, as visualized in Figs. 6 and 7.

Findings. Fig. 6 shows that all four dimensions exceed the neutral score. *Consistency* (4.5), *clarity* (4.0), and *relevance* (3.6) are notably strong, indicating root cause reports that are readable and well aligned with the incident context. However, *rationality* is lower at 3.1, which suggests that causal justification and evidential sufficiency are comparatively weaker. For the distribution view in Fig. 7, we aggregate one-decimal scores by mapping $[a, a+0.5)$ to a (*e.g.*, $[4.5, 5)$ to 4.5), leaving 5 for exact full scores. Only a few cases reach 5. *Consistency*, *clarity*, and *relevance* exhibit high-score concentration with modes around 4.5, 4.0, and 3.5, which reflects stable presentation and good alignment with observations. *Rationality* displays a bimodal pattern, with one mode in the mid–high range $(3.5–4.5)$ and another in the low range $(1.0–1.5)$. Through careful inspection of low-*rationality* cases, we observe broken reasoning chains and causal claims with insufficient evidential support, a pattern consistent with hallucination risk when operating with a lightweight 14B backbone.

Overall, *OpsAgent* produces interpretable reports across multiple dimensions and is well suited for use by OCEs in real-world cloud systems. The *rationality* gap highlights a practical direction for future improvement: strengthen hallucination mitigation or scale the backbone to enhance causal fidelity and reasoning stability.

E. 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 III, *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.

TABLE III
SELF-EVOLUTION CAPABILITY UNDER DIFFERENT TRAINING BUDGETS (SEED LLM: QWEN).

Seed LLM	Training (%)	Avg	
		Correct	Partial
Qwen2.5-14B-Instruct-1M	0	8.27	27.06
	10	8.27	29.32
	20	12.03	33.83
	30	10.52	35.34
	40	14.29	38.35
	50	15.04	39.85
	60	**16.54**	**41.35**

The results validate the self-evolution mechanism and support *OpsAgent*'s suitability for long-term deployment.

V. RELATED WORK

A. DL-based IM approaches.

DL-based approaches typically leverage neural networks to extract failure patterns from observability data and predict root causes, employing either supervised [8], [12] or self-supervised [3], [18] learning paradigms. Early studies typically focused on unimodal data sources, such as metrics [6], [35], [47]–[50], logs [51]–[54], or traces [55]–[58], which limited their ability to capture a comprehensive view of system states. Recent advances have moved toward multimodal fusion, exemplified by ART [3], DeepHunt [18], and DiagFusion [12], which integrate heterogeneous observability signals to improve diagnostic accuracy and efficiency. Despite these improvements, DL-based IM methods face three key limitations. First, models often overfit system-specific statistical patterns, leading to poor generalization across heterogeneous and evolving cloud systems. Second, their black-box nature hinders interpretability, offering little insight into the reasoning process behind root cause predictions, which reduces trustworthiness for OCEs in production settings. Finally, these models are typically trained in a static manner and cannot accumulate diagnostic experience over time, making them ill-suited for sustained deployment in dynamic cloud environments.

TABLE IV
COMPARISON OF IM CATEGORIES.

Categories	**Gen?**	**Int?**	**Cos?**	**Evo?**
DL-based [3], [12], [18]	□	□	■	□
LLM-based [2], [13], [14]	◧	◧	□	□
MAS-based [17], [26], [38]	◧	■	■	□
OpsAgent	■	■	■	■

Notes: "Gen?": Generalizable? "Int?": Interpretable? "Cos?": Cost-Efficient? "Evo?": Self-Evolving? "■": full support, "◧": partial support, "□": no support.

B. LLM-based IM approaches.

With the rapid progress of LLMs, recent studies have begun to explore their potential in IM. Ahmed et al. [13] fine-tuned GPT-3.X on domain-specific corpora to directly predict root causes from incident titles and summaries. RCACopilot [2] and Zhang et al. [14] adopt in-context learning strategies, where a few semantically similar past incidents are retrieved as examples to perform 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 the use of LLMs, recent efforts have further investigated multi-agent system (MAS)-based IM that leverage specialized agents for collaborative reasoning in IM. For instance, mABC [17] introduces a blockchain-inspired collaboration mechanism, where multiple LLM-based agents follow an agent workflow and engage in decentralized voting to collectively infer the root cause. TrioXpert [38] proposes a multi-expert architecture that combines modality-specific preprocessing with collaborative agents to jointly solve AD, FT, and RCL, though its reliance on deep neural networks in the numerical pipeline limits generalization and interpretability. D-Bot [26] focuses on database diagnosis, integrating offline knowledge extraction, automated prompt generation, and a tree-search algorithm with agent collaboration to produce timely diagnostic reports. In parallel, OPENRCA [1] introduces the first benchmark designed for evaluating natural-language-driven IM, decomposing the process into AD, FT, and RCL tasks and casting them as question answering over multimodal observability data. Despite employing state-of-the-art closed-source LLMs (*e.g.*, Claude 3.5 Sonnet, GPT-4o, Gemini 1.5 Pro), the best-performing method achieves only 11.34% accuracy, underscoring both the inherent difficulty of IM and the limitations of current approaches.

Overall, despite substantial progress across DL-based and LLM-based approaches, existing IM methods remain limited by poor generalization across heterogeneous environments, insufficient interpretability of reasoning processes, and high deployment overheads. As summarized in Table IV, *OpsAgent* is distinguished as the only method that unifies the generalizable, interpretable, cost-efficient, and self-evolving capabilities, making it well-suited for long-term deployment in real-world cloud environments.

VI. DISCUSSION

Limitations. Our goal is a deployable and sustainable MAS for IM, yet several trade-offs remain. *First*, the system still rests on the base LLM's intrinsic reasoning and language understanding. Because we adopt a deploy–then–adapt regime (*i.e.*, PPO fine-tuning and reflection after deployment), the initial capability can be modest and performance improves with accumulated usage. *Second*, the evaluation setting is stricter than routine operations: AD is graded at minute-level precision and FT/RCL require top-1 correctness. These metrics are informative for benchmarking but diverge from operational practice, where bounded timing tolerance and top-k support are acceptable and often preferred. *Third*, while the training-free data processor generalizes well across systems, achieving optimal performance in new or highly customized environments may still require minor adaptation, such as extending keyword lists and aligning threshold conventions.

Future Work. We highlight three directions that build on these observations. (1) *Stronger initialization.* Construct a domain-specialized, operations-oriented LLM so the system starts with better diagnostic priors, easing the cold-start phase and reducing reliance on post-deployment evolution. (2) *Practice-aligned evaluation.* Augment strict metrics with operationally meaningful ones, such as tolerance bands for AD timing, top-k coverage for FT/RCL, and human-in-the-loop effectiveness (*e.g.*, reduced Time-to-Mitigate or lower OCEs' effort). (3) *More adaptive data processing.* Develop mechanisms for embedding-guided vocabulary expansion to automatically recognize unseen terminology, and a dynamic threshold learning algorithm that adjusts sensitivity in real time by raising thresholds to suppress noise or lowering thresholds to capture informative signals, thereby providing more comprehensive and useful evidence for downstream diagnosis.

VII. 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 cloud 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.

References

[1] J. Xu, Q. Zhang, Z. Zhong, S. He, C. Zhang, Q. Lin, D. Pei, P. He, D. Zhang, and Q. Zhang, "Openrca: Can large language models locate the root cause of software failures?" in *The Thirteenth International Conference on Learning Representations*, 2025.

[2] Y. Chen, H. Xie, M. Ma, Y. Kang, X. Gao, L. Shi, Y. Cao, X. Gao, H. Fan, M. Wen *et al.*, "Automatic root cause analysis via large language models for cloud incidents," in *Proceedings of the Nineteenth European Conference on Computer Systems*, 2024, pp. 674–688.

[3] Y. Sun, B. Shi, M. Mao, M. Ma, S. Xia, S. Zhang, and D. Pei, "Art: A unified unsupervised framework for incident management in microservice systems," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 1183–1194.

[4] "Incident report of google cloud outage," https://status.cloud.google.com/incidents/ow5i3PPK96RduMcb1SsW, 2025.

[5] S. Zhang, S. Xia, W. Fan, B. Shi, X. Xiong, Z. Zhong, M. Ma, Y. Sun, and D. Pei, "Failure diagnosis in microservice systems: A comprehensive survey and analysis," *ACM Transactions on Software Engineering and Methodology*, 2024.

[6] M. Ma, J. Xu, Y. Wang, P. Chen, Z. Zhang, and P. Wang, "Automap: Diagnose your microservice-based web applications automatically," in *Proceedings of The Web Conference 2020*, 2020, pp. 246–258.

[7] Y. Meng, S. Zhang, Y. Sun, R. Zhang, Z. Hu, Y. Zhang, C. Jia, Z. Wang, and D. Pei, "Localizing failure root causes in a microservice through causality inference," in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, 2020, pp. 1–10.

[8] C. Lee, T. Yang, Z. Chen, Y. Su, and M. R. Lyu, "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, 2023, pp. 1750–1762.

[9] L. Tao, S. Zhang, Z. Jia, J. Sun, M. Ma, Z. Li, Y. Sun, C. Yang, Y. Zhang, and D. Pei, "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*, 2024, pp. 1107–1119.

[10] L. Zheng, Z. Chen, J. He, and H. Chen, "Mulan: multi-modal causal structure learning and root cause analysis for microservice systems," in *Proceedings of the ACM Web Conference 2024*, 2024, pp. 4107–4116.

[11] G. Yu, P. Chen, Y. Li, H. Chen, X. Li, and Z. Zheng, "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*, 2023, pp. 553–565.

[12] S. Zhang, P. Jin, Z. Lin, Y. Sun, B. Zhang, S. Xia, Z. Li, Z. Zhong, M. Ma, W. Jin *et al.*, "Robust failure diagnosis of microservice system through multimodal data," *IEEE Transactions on Services Computing*, vol. 16, no. 6, pp. 3851–3864, 2023.

[13] T. Ahmed, S. Ghosh, C. Bansal, T. Zimmermann, X. Zhang, and S. Rajmohan, "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, 2023, pp. 1737–1749.

[14] X. Zhang, S. Ghosh, C. Bansal, R. Wang, M. Ma, Y. Kang, and S. Rajmohan, "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*, 2024, pp. 266–277.

[15] Z. Wang, J. Li, M. Ma, Z. Li, Y. Kang, C. Zhang, C. Bansal, M. Chintalapati, S. Rajmohan, Q. Lin *et al.*, "Large language models can provide accurate and interpretable incident triage," in *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2024, pp. 523–534.

[16] Y. Han, Q. Du, Y. Huang, J. Wu, F. Tian, and C. He, "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*, 2024, pp. 931–943.

[17] W. Zhang, H. Guo, J. Yang, Z. Tian, Y. Zhang, Y. Chaoran, Z. Li, T. Li, X. Shi, L. Zheng *et al.*, "mabc: Multi-agent blockchain-inspired collaboration for root cause analysis in micro-services architecture," in *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024, pp. 4017–4033.

[18] Y. Sun, Z. Lin, B. Shi, S. Zhang, S. Ma, P. Jin, Z. Zhong, L. Pan, Y. Guo, and D. Pei, "Interpretable failure localization for microservice systems based on graph autoencoder," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 2, pp. 1–28, 2025.

[19] M. A. Islam, M. E. Ali, and M. R. Parvez, "Mapcoder: Multi-agent code generation for competitive problem solving," in *Annual Meeting of the Association of Computational Linguistics 2024*. Association for Computational Linguistics (ACL), 2024, pp. 4912–4944.

[20] D. Zan, A. Yu, W. Liu, D. Chen, B. Shen, W. Li, Y. Yao, Y. Gong, X. Chen, B. Guan *et al.*, "Codes: Natural language to code repository via multi-layer sketch," *CoRR*, 2024.

[21] H. Zhang, W. Cheng, Y. Wu, and W. Hu, "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*, 2024, pp. 1319–1331.

[22] M. Taeb, A. Swearngin, E. Schoop, R. Cheng, Y. Jiang, and J. Nichols, "Axnav: Replaying accessibility tests from natural language," in *Pro-*

ceedings of the 2024 CHI Conference on Human Factors in Computing Systems, 2024, pp. 1–16.

[23] S. Hu, T. Huang, F. İlhan, S. F. Tekin, and L. Liu, "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, 2023, pp. 297–306.

[24] D. Jin, Z. Jin, X. Chen, and C. Wang, "Mare: Multi-agents collaboration framework for requirements engineering," *arXiv preprint arXiv:2405.03256*, 2024.

[25] M. Ataei, H. Cheong, D. Grandi, Y. Wang, N. Morris, and A. Tessier, "Elicitron: A large language model agent-based simulation framework for design requirements elicitation," *Journal of Computing and Information Science in Engineering*, vol. 25, no. 2, p. 021012, 2025.

[26] X. Zhou, G. Li, Z. Sun, Z. Liu, W. Chen, J. Wu, J. Liu, R. Feng, and G. Zeng, "D-bot: Database diagnosis system using large language models," *Proceedings of the VLDB Endowment*, vol. 17, no. 10, pp. 2514–2527, 2024.

[27] 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.

[28] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," *Present and ulterior software engineering*, pp. 195–216, 2017.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[30] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[31] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, Q. Guo, M. Wang *et al.*, "Retrieval-augmented generation for large language models: A survey," *CoRR*, 2023.

[32] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, D. Metropolitansky, R. O. Ness, and J. Larson, "From local to global: A graph rag approach to query-focused summarization," *arXiv preprint arXiv:2404.16130*, 2024.

[33] H. Qian, Z. Liu, P. Zhang, K. Mao, D. Lian, Z. Dou, and T. Huang, "Memorag: Boosting long context processing with global memory-enhanced retrieval augmentation," in *Proceedings of the ACM on Web Conference 2025*, 2025, pp. 2366–2377.

[34] B. Sarmah, D. Mehta, B. Hall, R. Rao, S. Patel, and S. Pasquali, "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*, 2024, pp. 608–616.

[35] C. Wu, N. Zhao, L. Wang, X. Yang, S. Li, M. Zhang, X. Jin, X. Wen, X. Nie, W. Zhang *et al.*, "Identifying root-cause metrics for incident diagnosis in online service systems," in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2021, pp. 91–102.

[36] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE international conference on web services (ICWS)*. IEEE, 2017, pp. 33–40.

[37] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information processing & management*, vol. 24, no. 5, pp. 513–523, 1988.

[38] Y. Sun, Y. Luo, X. Wen, Y. Yuan, X. Nie, S. Zhang, T. Liu, and X. Luo, "Trioxpert: An automated incident management framework for microservice system," *arXiv preprint arXiv:2506.10043*, 2025.

[39] 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.

[40] S. T. A. C. J. M. J. Y. Jennifer Mace, Jelena Oertel, "Sre book, chapter 9: Incident response," https://sre.google/workbook/incident-response/.

[41] Y. Liu, D. Iter, Y. Xu, S. Wang, R. Xu, and C. Zhu, "G-eval: Nlg evaluation using gpt-4 with better human alignment," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 2511–2522.

[42] N. Reimers and I. Gurevych, "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, 11 2019. [Online]. Available: http://arxiv.org/abs/1908.10084

[43] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.

[44] D. Roy, X. Zhang, R. Bhave, C. Bansal, P. Las-Casas, R. Fonseca, and S. Rajmohan, "Exploring llm-based agents for root cause analysis," in *Companion proceedings of the 32nd ACM international conference on the foundations of software engineering*, 2024, pp. 208–219.

[45] 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.

[46] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[47] D. Wang, Z. Chen, Y. Fu, Y. Liu, and H. Chen, "Incremental causal graph learning for online root cause analysis," in *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*, 2023, pp. 2269–2278.

[48] Z. Li, N. Zhao, M. Li, X. Lu, L. Wang, D. Chang, X. Nie, L. Cao, W. Zhang, K. Sui *et al.*, "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*, 2022, pp. 996–1008.

[49] P. Wang, J. Xu, M. Ma, W. Lin, D. Pan, Y. Wang, and P. Chen, "Cloudranger: Root cause identification for cloud native systems," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2018, pp. 492–502.

[50] M. Ma, W. Lin, D. Pan, and P. Wang, "Ms-rank: Multi-metric and self-adaptive root cause diagnosis for microservice applications," in *2019 IEEE International Conference on Web Services (ICWS)*. IEEE, 2019, pp. 60–67.

[51] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "Swisslog: Robust anomaly detection and localization for interleaved unstructured logs," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 4, pp. 2762–2780, 2022.

[52] Y. Sui, Y. Zhang, J. Sun, T. Xu, S. Zhang, Z. Li, Y. Sun, F. Guo, J. Shen, Y. Zhang *et al.*, "Logkg: Log failure diagnosis through knowledge graph," *IEEE Transactions on Services Computing*, vol. 16, no. 5, pp. 3493–3507, 2023.

[53] Y. Xie, K. Yang, and P. Luo, "Logm: Log analysis for multiple components of hadoop platform," *IEEE Access*, vol. 9, pp. 73 522–73 532, 2021.

[54] X. Zhang, Y. Xu, S. Qin, S. He, B. Qiao, Z. Li, H. Zhang, X. Li, Y. Dang, Q. Lin *et al.*, "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*, 2021, pp. 1253–1263.

[55] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, "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*, 2019, pp. 683–694.

[56] G. Yu, Z. Huang, and P. Chen, "Tracerank: Abnormal service localization with dis-aggregated end-to-end tracing data in cloud native systems," *Journal of Software: Evolution and Process*, vol. 35, no. 10, p. e2413, 2023.

[57] Z. Li, J. Chen, R. Jiao, N. Zhao, Z. Wang, S. Zhang, Y. Wu, L. Jiang, L. Yan, Z. Wang *et al.*, "Practical root cause localization for microservice systems via trace analysis," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 2021, pp. 1–10.

[58] J. Yang, Y. Guo, Y. Chen, and Y. Zhao, "Tracenet: Operation aware root cause localization of microservice system anomalies," in *2023 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2023, pp. 758–763.