Why Do AI Agents Systematically Fail at Cloud Root Cause Analysis?

Taeyoon Kim[*]
tykim7@hanyang.ac.kr
Dept. of Data Science
Hanyang University
Seoul, Republic of Korea

Woohyeok Park[*]
woohyeok@hanyang.ac.kr
Dept. of Data Science
Hanyang University
Seoul, Republic of Korea

Hoyeong Yun
hy.yun@okestro.com
Intelligent Cloud Lab
OKESTRO Co., Ltd.
Seoul, Republic of Korea

Kyungyong Lee[†]
kyungyong@hanyang.ac.kr
Dept. of Data Science
Hanyang University
Seoul, Republic of Korea

Abstract

Failures in large-scale cloud systems incur substantial financial losses, making automated Root Cause Analysis (RCA) essential for operational stability. Recent efforts leverage Large Language Model (LLM) agents to automate this task, yet existing systems exhibit low detection accuracy even with capable models, and current evaluation frameworks assess only final answer correctness without revealing why the agent's reasoning failed. This paper presents a process-level failure analysis of LLM-based RCA agents. We execute the full OpenRCA benchmark across five LLM models, producing 1,675 agent runs, and classify observed failures into 12 pitfall types across *intra-agent* reasoning, *inter-agent* communication, and *agent-environment* interaction. Our analysis reveals that the most prevalent pitfalls, notably hallucinated data interpretation and incomplete exploration, persist across all models regardless of capability tier, indicating that these failures originate from the shared agent framework rather than from individual model limitations. Controlled mitigation experiments further show that prompt engineering alone cannot resolve the dominant pitfalls, whereas enriching the *inter-agent* communication protocol reduces communication-related failures by up to 15 percentage points. The pitfall taxonomy and diagnostic methodology developed in this work provide a foundation for designing more reliable autonomous agents for cloud RCA.

Keywords

Root Cause Analysis, AI Agent, Cloud Operations, AIOps

1 Introduction

Failures in large-scale distributed web services result in significant downtime and financial losses, rendering automated Root Cause Analysis (RCA) essential for service reliability. RCA remains challenging due to complex microservice dependencies and the need to correlate heterogeneous telemetry data across metrics, logs, and traces. The advent of LLMs has accelerated multi-agent systems for automating this task, with frameworks like OpenRCA [8] providing representative benchmarks and baseline agent frameworks.

However, the practical performance of these baseline agents remains very low (Table 1), with overall perfect accuracy ranging from 3.9% to 12.5% across five models spanning different capability tiers. Furthermore existing evaluation frameworks [2, 5, 12] assess only final answer correctness without revealing why the agent's reasoning or collaboration fails. Without such process-level understanding, practitioners cannot determine whether to invest in stronger models, better prompts, or redesigned agent frameworks.

This paper addresses this gap through a systematic, process-level failure analysis. We execute the full OpenRCA benchmark across five LLM models, producing 1,675 agent runs, and classify the observed failures into 12 pitfall types across *intra-agent* reasoning, *inter-agent* communication, and *agent-environment* interaction. We further conduct mitigation experiments for each category, finding that structural modifications to the agent framework yield measurable improvements while prompt-level interventions alone do not.

The main contributions of this work are as follows.

- A process-level analysis methodology and a taxonomy of 12 pitfall types that enables systematic diagnosis of RCA agent failures beyond outcome-based evaluation.
- An empirical analysis showing that the dominant failure modes are shared across all models and originate from the agent framework rather than model limitations.
- Controlled mitigation experiments showing that enriched *inter-agent* communication produces consistent performance gains, while prompt-level interventions alone do not resolve the identified pitfalls.

2 LLM Agent for RCA

RCA is the systematic process of identifying the originating source of failure in software systems. Given an observed anomaly, the goal is to determine the faulty component (e.g.,

[*]Both authors contributed equally to this work.
[†]Corresponding author

Table 1: Accuracy comparison of the baseline OpenRCA agent across three service domains

	Overall (%)		Telecom (%)		Bank (%)		Market (%)	
Model	●	▲	●	▲	●	▲	●	▲
Gemini 2.5 Pro	**12.5**	**22.4**	11.8	13.7	**19.9**	**19.9**	**6.1**	**27.7**
GPT-5 mini	8.4	21.5	**15.7**	**21.6**	11.8	16.2	2.7	26.4
GPT-OSS 120B	6.9	12.2	9.8	15.7	7.4	11.8	5.4	11.5
Solar Pro 2	5.7	15.8	9.8	11.8	6.6	15.4	3.4	17.6
Claude Sonnet 4	3.9	14.3	5.9	7.8	4.4	15.4	2.7	15.5
Claude Sonnet 3.5*	11.3	17.3	–	–	–	–	–	–

* Results from the original RCA-Agent work [8]. ● Perfect / ▲ Partial

a specific microservice), the incident time when the fault first occurred, and the reason for the failure (e.g., memory leak). In practice, these elements are rarely observable directly. A fault in one component propagates through service dependencies and manifests as symptoms in other components, so the engineer must trace the causal chain backward from observed symptoms to the originating source.

This tracing process requires correlating heterogeneous telemetry data across multiple modalities. System metrics such as CPU utilization and memory usage provide time-series signals at the infrastructure level. Application logs record timestamped events and error messages in unstructured text. Distributed traces capture the end-to-end path of individual requests across services, revealing latency and dependency relationships. Each modality provides a partial view of the system state, and effective RCA demands joint analysis across all three. RCA therefore traditionally relies on expert engineers with deep domain knowledge to interpret these signals and localize faults.

Recent work has applied LLM reasoning techniques such as ReAct [9] and Chain of Thought [7] to automate RCA, but embedding large volumes of telemetry data into a single agent's context faces scalability limitations. To address this, recent research [4, 8] has adopted multi-agent architectures that decompose complex responsibilities across specialized agents, reducing individual agent complexity and enabling programmatic management of large-scale operational tasks.

In agent-based RCA, OpenRCA [8] provides a representative benchmark and baseline agent system. The benchmark comprises 335 failure incidents from three web service domains, namely Telecom (51 cases), Bank (136 cases), and Market (148 cases). Each incident is annotated with a ground truth root cause specifying the faulty component and failure reason, and the dataset covers 73 unique components and 28 distinct failure reasons in total. For each incident, OpenRCA provides three types of telemetry data corresponding to the modalities described above, including system metrics, application logs, and distributed traces. The full dataset totals 68.5GB and over 523 million lines across these modalities.

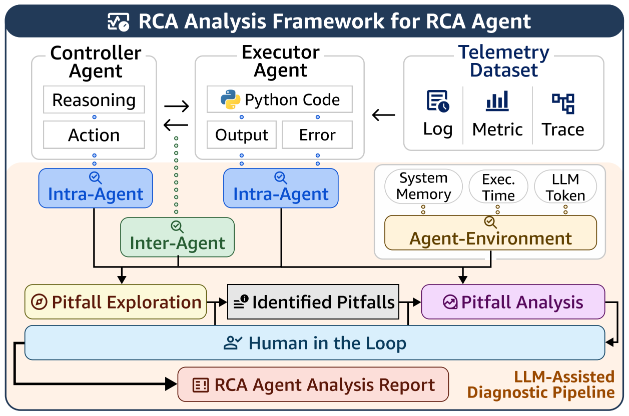

Figure 1: Analysis framework for diagnosing RCA agent failures across three interfaces

Alongside the dataset, OpenRCA proposes an agent system for RCA that adopts a Controller-Executor architecture. The Controller performs high-level reasoning and produces natural language instructions for the Executor, which translates them into Python code, executes it over telemetry data, and returns results. We refer to this orchestration layer collectively as the agent framework, distinct from the underlying LLM. Despite these contributions, the detection accuracy remains very low, as shown in Table 1. A perfect detection requires all three root cause elements namely component, time, and reason to be correct, while Partial indicates that only a subset is identified. Even the best performing model, Gemini 2.5 Pro, achieves only 12.5% perfect detection across all 335 incidents, yet existing evaluation frameworks confine assessment to final answer correctness and provide no insight into why the agent's reasoning failed [2, 12].

3 Agent Failure Diagnosis Methodology

To systematically diagnose the causes of the OpenRCA agent's low detection performance, we developed an analysis framework that classifies failures into three categories according to their architectural origin, providing multiple diagnostic perspectives on whether a failure arises from an individual agent's reasoning, from miscommunication between agents, or from the execution environment. As illustrated in Figure 1, the three categories correspond to *intra-agent* reasoning within each agent, *inter-agent* communication between the Controller and Executor, and *agent-environment* interaction with the telemetry data and execution runtime.

The analysis covers 335 tasks executed across five models, yielding 1,675 individual agent runs. Each run consists of multiple steps, averaging 11.1 steps per run. Collectively, these runs consumed 609.9 hours of wall clock time, approximately 1.38 billion tokens, and an estimated $1,394 in cost.

Table 2: Pitfall taxonomy with diagnostic criteria

Category	Pitfall	Core Issue	Diagnostic Question
Intra-Agent	Hallucination in Interpretation	Narrative fabrication	Did the agent misinterpret what the data means?
	Incomplete Exploration	Scope narrowing	Did the agent skip a relevant KPI & Component?
	Symptom-as-Cause	Premature attribution	Did the agent mistake a symptom for the root cause?
	Code generation error	LLM code quality deficit	Did the generated code execute correctly?
	Limited Telemetry Coverage	Source narrowness	Did the agent rely on only one telemetry source?
	Timestamp Error	Temporal misalignment	Did the agent analyze the correct time window?
	No Cross-Validation	Single-hypothesis bias	Did the agent verify findings with alternative sources?
Inter-Agent	Instruction-Code Mismatch	Implementation gap	Did the code reflect the Controller's intent?
	Meaningless Repetition	Repetitive loop	Did the agent repeat the same failed approach?
	Misattributed Evidence	Opaque handoff	Did the Controller misunderstand what the Executor's results represent?
Agent-Environment	Out-of-Memory	Resource exhaustion	Did execution terminate due to memory limits?
	Max Step Exhaustion	Budget depletion	Did the agent run out of steps?

Table 3: Distribution of Identified Intra & Inter Pitfalls (N=1675)

Category	Pitfall	Overall	Claude Sonnet 4	Solar Pro 2	GPT-5 mini	GPT-OSS 120B	Gemini 2.5 Pro
Intra-Agent	**Hallucination in Interpretation**	**1193 (71.2%)**	**267 (79.5%)**	**244 (72.8%)**	**232 (69.3%)**	**226 (67.5%)**	224 (66.9%)
	Incomplete Exploration	1071 (63.9%)	247 (73.5%)	180 (53.7%)	191 (57.0%)	224 (66.9%)	**229 (68.4%)**
	Symptom-as-Cause	669 (39.9%)	115 (34.2%)	117 (34.9%)	152 (45.4%)	151 (45.1%)	134 (40.0%)
	Code generation error	456 (27.2%)	220 (65.5%)	46 (13.7%)	139 (41.5%)	45 (13.4%)	6 (1.8%)
	Limited Telemetry Coverage	451 (26.9%)	132 (39.3%)	41 (12.2%)	105 (31.3%)	54 (16.1%)	119 (35.5%)
	Timestamp Error	390 (23.3%)	114 (33.9%)	98 (29.3%)	83 (24.8%)	47 (14.0%)	48 (14.3%)
	No Cross-Validation	311 (18.6%)	119 (35.4%)	45 (13.4%)	43 (12.8%)	64 (19.1%)	40 (11.9%)
Inter-Agent	Instruction-Code Mismatch	405 (18.8%)	148 (24.8%)	51 (22.5%)	168 (25.5%)	36 (11.8%)	2 (0.5%)
	Meaningless Repetition	179 (8.3%)	45 (7.5%)	0 (0.0%)	116 (17.6%)	18 (5.9%)	0 (0.0%)
	Misattributed Evidence	73 (3.4%)	38 (6.4%)	4 (1.8%)	23 (3.5%)	8 (2.6%)	0 (0.0%)

Manually inspecting all runs at this scale is infeasible, so we constructed an analysis pipeline using Claude Code [1] with Opus 4.5 as the analysis agent [10]. Following the human-in-the-loop principle [6], we verified every classification before inclusion in the final taxonomy.

We restricted the analysis agent to binary diagnostic questions to minimize bias and improve reproducibility. The pipeline operates in two stages. In the first stage, the agent performs free-form exploration with only a JSON output schema, surfacing diverse failure patterns without predefined categories. In the second stage, we consolidated the collected patterns into mutually exclusive categories and repeated the analysis with binary diagnostic questions such as "Did the agent skip a relevant KPI & Component?". The resulting taxonomy of 12 pitfall types is presented in Table 2.

4 Architectural Pitfalls in RCA Agents

We selected five reasoning models to represent the current LLM ecosystem, given their increasing adoption for complex agentic tasks. We also considered both capability and inference cost. Claude Sonnet 4, GPT-5 mini, and Gemini 2.5 Pro cover major providers. Solar Pro 2 represents a compact scale, and GPT-OSS 120B serves as an open-weight alternative.

Table 3 presents the distribution of the identified pitfalls across 1,675 agent executions of five LLM models, sorted by overall frequency. To capture the full spectrum of architectural failure modes, we record every pitfall observed per execution rather than assigning each execution to a single category, as recording only one dominant pitfall would obscure the relative frequency of each failure mode and limit the ability to prioritize architectural improvements. Column percentages therefore sum to more than 100%. The following subsections examine each pitfall category in detail.

4.1 Intra-Agent Pitfalls

We group the seven ***intra-agent*** pitfalls in Table 3 according to where in the diagnostic process the failure occurs.

Hallucination in Interpretation, observed in 71.2% of executions, spans all stages of this process. Rather than faithfully reading the returned data, the Controller imposes an interpretive narrative onto it, assigning meaning that appears coherent but does not reflect what the values indicate, a pattern consistent with the common generative bias of LLMs. The remaining six pitfalls fall into three categories.

Incomplete Exploration and ***Limited Telemetry Coverage*** reflect a common failure in investigation breadth. *Incomplete Exploration* at 63.9% manifests at component and Key Performance Indicator (KPI) level. Although the OpenRCA framework provides lists of candidate components and KPI families in its prompt, agents routinely ignore entire categories, analyzing CPU metrics exclusively while never querying network KPIs, for example. *Limited Telemetry Coverage* at 26.9% operates at the data modality level. Agents predominantly draw conclusions from metric data alone and rarely examine log or trace sources, even though the framework provides all three for every incident.

Symptom-as-Cause and ***No Cross-Validation*** reflect the opposite failure in the depth of investigation. Even when agents reach a relevant analysis target, they terminate prematurely rather than pursuing the full causal chain. *Symptom-as-Cause* at 39.9% captures cases where the agent treats the first anomaly it encounters as the root cause without tracing further upstream. *No Cross-Validation* at 18.6% reflects a related tendency to accept a single finding without verifying it against alternative telemetry sources.

Code generation error and ***Timestamp Error*** reflect limitations in the agent's execution capability rather than its analytical reasoning. Code generation error at 27.2% varies sharply, from 1.8% for Gemini 2.5 Pro to 65.5% for Claude Sonnet 4, indicating that code generation reliability remains a differentiating factor. *Timestamp Error* at 23.3% arises mainly from timezone misalignment in the OpenRCA dataset, suggesting that such runtime details are better handled through data preprocessing than through prompt instructions.

Each model shows distinct failure profiles. Specifically, Code generation error ranges from 1.8% for Gemini 2.5 Pro to 65.5% for Claude Sonnet 4, while *Symptom-as-Cause* rates peak above 45% for GPT-5 mini and GPT-OSS 120B. Despite these variations, all five models exhibit *Hallucination in Interpretation* above 66% and *Incomplete Exploration* above 53%.

Since the five models differ in provider, capability, and cost tier but share the same agent framework, these uniformly high rates point to the shared framework as the primary bottleneck rather than to any individual model's limitations.

4.2 Inter-Agent Pitfalls

In the OpenRCA system, the Controller and Executor communicate exclusively through natural language summaries, with neither agent having access to the other's internal context. This opaque interface produces three ***inter-agent*** pitfalls identified through step-level analysis.

Instruction-Code Mismatch is the most pervasive, affecting 20 to 26% of all execution steps for GPT-5 mini, Solar Pro 2, and Claude Sonnet 4. Summarized instructions strip away contextual cues needed to reconstruct the sender's intent [3], causing the Executor to misinterpret the analytical question and generate code that addresses a different one.

Meaningless Repetition, observed at 17.6% for GPT-5 mini, represents an escalated form of the same communication failure. Without shared execution history, the Controller cannot recognize that a previous instruction has already failed and repeats the same directive, entering a loop that consumes the step budget without advancing diagnosis.

Misattributed Evidence at 3.4% originates on the Controller side. Unlike *Hallucination in Interpretation*, where the Controller misreads the data itself, this pitfall stems from the opacity of the Executor's process. The Controller receives only a natural language summary of the Executor's findings without visibility into the code that produced them. When the Executor's implementation deviates from the intended analysis, the Controller has no means to detect the discrepancy and accepts the reported findings at face value, building subsequent reasoning on a flawed evidential basis.

The severity of these pitfalls varies sharply across models. Gemini 2.5 Pro exhibits nearly zero rates across all three categories, while Claude Sonnet 4 and GPT-5 mini record the highest *Instruction-Code Mismatch* at 24.8% and 25.5% respectively, with GPT-5 mini also exhibiting the highest *Meaningless Repetition* at 17.6%. All three pitfalls stem from the information loss inherent to the natural language summary interface, suggesting that enriching the communication channel between agents may reduce these failures.

4.3 Agent-Environment Pitfalls

To enhance resource efficiency and reduce latency, the OpenRCA agent system uses a persistent Python kernel that retains variables and data structures in memory across multiple turns, eliminating the need for redundant data loading. However, agents lack awareness of the kernel's accumulated state, and this disconnect produces two environment-level failures.

Out of Memory (OOM) failures were detected through the memory profiler integrated during our analysis. In the Bank domain, 2 of 41 scenarios terminated during execution when agents reloaded datasets already in memory or failed to release obsolete variables. These failures are categorical rather than partial, as an OOM crash terminates the entire diagnostic session regardless of reasoning quality.

Max Step Exhaustion accounts for 4.1% of executions overall, with GPT-5 mini at 10.4% and Claude Sonnet 4 at 8.3%, while Solar Pro 2 never triggers this pitfall. Whether step exhaustion reflects inefficient execution or more thorough exploration attempts requires a joint analysis of step-level exploration breadth and step consumption.

5 Mitigating Pitfalls in RCA Agent

Building on the pitfall analysis, this section examines whether targeted interventions can mitigate each failure category. Rather than proposing a new end-to-end system, we conduct controlled experiments that isolate the effect of a single modification per category, identifying which pitfalls respond to focused changes and which demand fundamental redesign.

5.1 Intra-Agent: Paradoxes of Prompt Engineering

The most intuitive response to the *intra-agent* pitfalls is to improve the Controller's prompt. We tested two approaches on Claude Sonnet 4, selected for its lowest baseline accuracy, across 70 Bank domain tasks. Hypothesis-driven prompting [11] augments the prompt with a structured template requiring explicit hypotheses for each KPI category, targeting Incomplete Exploration by forcing broader coverage. Pitfall-aware prompting injects descriptions of the frequent pitfalls, instructing the agent to avoid patterns such as skipping network KPIs or accepting the first anomaly as the root cause.

Neither produced a meaningful improvement in root cause identification. Hypothesis-driven prompting did succeed in broadening the exploration scope, with previously ignored KPI categories such as memory utilization appearing as explicit hypotheses. However, the *Hallucination in Interpretation* pitfall persisted at comparable rates. The agent reached the relevant data but continued to impose incorrect interpretations onto it, generating plausible but unfounded narratives regardless of the additional guidance provided. Pitfall-aware prompting exhibited a similar disconnect, with the agent acknowledging pitfall descriptions in its reasoning trace but reproducing the same interpretive failures in practice.

These results demonstrate a clear asymmetry in what prompt engineering can and cannot achieve. While augmented prompts successfully broadened the exploration scope, the Controller continued to fabricate interpretations of the retrieved data comparably, indicating that the hallucination pitfall is not a matter of insufficient guidance but a structural property of the generation process. Mitigating this failure requires architectural intervention, such as external verification modules, rather than further prompt refinement.

5.2 Inter-Agent: Enriched Communication

To mitigate the information loss inherent to the opaque communication interface, we enriched the protocol so that the Executor returns the generated Python code and complete execution output, including exceptions and stack traces, alongside its natural language summary. Symmetrically, the Executor receives the Controller's full diagnostic analysis, a snippet of the previous execution output, and the overall objective, enabling more closely aligned code generation.

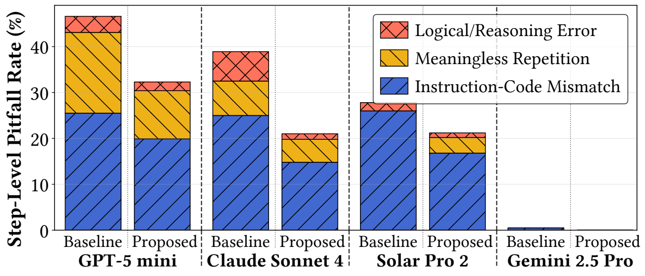

Figure 2: Step-level inter-agent pitfall rates under baseline and enriched communication

Figure 2 compares the pitfall frequency under both protocols. All four models show reduced pitfall rates, with GPT-5 mini and Claude Sonnet 4 exhibiting the largest gains of approximately 14 and 15 percentage points, respectively. Code exposure enables the Controller to directly detect instruction-code discrepancies, while concrete raw error messages prevent the Controller from repeating failed instructions.

These pitfall reductions translate into measurable improvements in RCA performance. In Bank domain experiments, the number of perfect detections increases across all four models under the enriched protocol, with GPT-5 mini rising from 0 to 2 (0.0% to 4.9%), Gemini 2.5 Pro from 1 to 3 (2.4% to 7.3%), and Solar Pro 2 from 2 to 3 (4.9% to 7.3%).

Although the enriched protocol increases token consumption per step by 24.8% on average (from 68K to 85K tokens), the reduced number of steps (−22.1%, from 11.9 to 9.2 per run) yields a net decrease of 1.6% in total tokens and 22.3% in execution time. By exposing code, errors, and diagnostic context between agents, a simple structural modification to the communication interface can simultaneously improve both accuracy and efficiency, achieving gains that prompt engineering alone could not deliver.

5.3 Agent-Environment: Robust State Isolation

As described in Section 4.3, the persistent kernel retains variables across turns, but agents lack awareness of the kernel's accumulated state. When agents reload datasets already in memory or fail to release obsolete variables, the resulting OOM crash terminates the entire diagnostic session regardless of the quality of the reasoning. To prevent this, we integrated a memory watcher that monitors kernel consumption and, upon exceeding a predefined threshold, terminates execution and transmits a structured warning to the Controller, which then generates a more memory-efficient implementation on a restarted kernel. Validation across all models and domains confirmed that this mechanism eliminates all OOM failures observed in the baseline setup.

6 Conclusion and Future Work

This paper presented a process-level failure analysis of LLM-based RCA agents that moves beyond outcome-based evaluation to diagnose why agents fail rather than merely how often. Through 1,675 agent runs across five LLM models on the full OpenRCA benchmark, we classified failures into 12 pitfall types spanning *intra-agent* reasoning, *inter-agent* communication, and *agent-environment* interaction. The analysis revealed that *Hallucination in Interpretation* (71.2%) and *Incomplete Exploration* (63.9%) persist across all tested models at comparable rates regardless of capability tier, establishing that these dominant failures originate from the shared agent framework rather than from individual model limitations.

Mitigation experiments reinforced this architectural diagnosis. Prompt engineering broadened the scope of investigation but could not suppress the interpretive errors that constitute the most prevalent pitfall. In contrast, enriching the *inter-agent* communication protocol with code and execution output reduced communication-related pitfalls by up to 15 percentage points, improved detection scores across all models, and simultaneously reduced execution time by 22.3% through more efficient step utilization. A memory watcher that intercepts resource exhaustion before kernel crashes further eliminated all OOM failures observed in the baseline. Together, these results demonstrate that structural modifications to the agent framework are both necessary and effective where prompt-level interventions are not.

This study has several limitations. The mitigation experiments were conducted on the Bank domain subset, the diagnostic pipeline relied on semi-automated classification with human verification that limits scalability, and the generalizability of the pitfall taxonomy to other multi-agent RCA frameworks remains to be validated. Future work will pursue two complementary directions. First, the pitfall taxonomy and binary diagnostic criteria can serve as the basis for a continuous monitoring pipeline that identifies failure patterns across agent runs without manual inspection. Second, the persistent dominance of *Hallucination in Interpretation* despite prompt-level intervention indicates the need for more fundamental architectural changes such as verification modules that cross-check agent interpretations against raw data, structured state sharing, and adaptive task decomposition.

Acknowledgments

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2025-25441560 & RS-2022-00144309) & Upstage-AWS AI Initiative Program.

References

[1] Anthropic. 2026. *Claude Code.* https://code.claude.com/

[2] Yinfang Chen, Manish Shetty, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Jonathan Mace, Chetan Bansal, Rujia Wang, and Saravan Rajmohan. 2025. AIOpsLab: A Holistic Framework for Evaluating AI Agents for Enabling Autonomous Cloud. In *MLSys '25.* https://openreview.net/forum?id=3EXBLwGxtq

[3] Ahmed M. Hussain, Salahuddin Salahuddin, and Panos Papadimitratos. 2025. Beyond Context: Large Language Models Failure to Grasp Users Intent. arXiv:2512.21110 [cs.AI] https://arxiv.org/abs/2512.21110

[4] Changhua Pei, Zexin Wang, Fengrui Liu, Zeyan Li, Yang Liu, Xiao He, Rong Kang, Tieying Zhang, Jianjun Chen, Jianhui Li, Gaogang Xie, and Dan Pei. 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 (WWW '25).* Association for Computing Machinery, 422–431. doi:10.1145/3701716.3715225

[5] Luan Pham, Hongyu Zhang, Huong Ha, Flora Salim, and Xiuzhen Zhang. 2025. RCAEval: A Benchmark for Root Cause Analysis of Microservice Systems with Telemetry Data. In *Companion Proceedings of the ACM on Web Conference 2025 (WWW '25).* Association for Computing Machinery, 777–780. doi:10.1145/3701716.3715290

[6] Wannita Takerngsaksiri, Jirat Pasuksmit, Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Ruixiong Zhang, Fan Jiang, Jing Li, Evan Cook, Kun Chen, and Ming Wu. 2025. Human-In-The-Loop Software Development Agents. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP).* 342–352. doi:10.1109/ICSE-SEIP66354.2025.00036

[7] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (NIPS '22).* Curran Associates Inc., Article 1800, 14 pages.

[8] 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.* https://openreview.net/forum?id=M4qNIzQYpd

[9] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations.* https://openreview.net/forum?id=WE_vluYUL-X

[10] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-bench and Chatbot Arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23).* Curran Associates Inc., Red Hook, NY, USA, Article 2020, 29 pages.

[11] Yangqiaoyu Zhou, Haokun Liu, Tejes Srivastava, Hongyuan Mei, and Chenhao Tan. 2024. Hypothesis Generation with Large Language Models. In *Proceedings of the 1st Workshop on NLP for Science (NLP4Science)*, Lotem Peled-Cohen, Nitay Calderon, Shir Lissak, and Roi Reichart (Eds.). Association for Computational Linguistics, Miami, FL, USA, 117–139. doi:10.18653/v1/2024.nlp4science-1.10

[12] Mingchen Zhuge, Changsheng Zhao, Dylan R. Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. 2025. Agent-as-a-Judge: Evaluate Agents with Agents. In *Forty-second International Conference on Machine Learning.* https://openreview.net/forum?id=Nn9POI9Ekt