MABC: Multi-Agent Blockchain-inspired Collaboration for Root Cause Analysis in Micro-Services Architecture

Wei Zhang[1], Hongcheng Guo[1*], Jian Yang[1*], Zhoujin Tian[1], Yi Zhang[1], Chaoran Yan[1], Zhoujun Li[1*], Tongliang Li[2], Xu Shi[3], Liangfan Zheng[3], Bo Zhang[3]

[1]State Key Laboratory of Complex & Critical Software Environment, Beihang University
[2]Computer School, Beijing Information Science and Technology University
[3]Cloudwise Research

{zwpride,hongchengguo,jiaya,eitbar,zhangyi2021,ycr2345,lizj}@buaa.edu.cn;
tonyliangli@bistu.edu.cn;{tim.shi,leven.zheng,bowen.zhang}@cloudwise.com;*

Abstract

Root cause analysis (RCA) in Micro-services architecture (MSA) with escalating complexity encounters complex challenges in maintaining system stability and efficiency due to fault propagation and circular dependencies among nodes. Diverse root cause analysis faults require multi-agents with diverse expertise. To mitigate the hallucination problem of large language models (LLMs), we design blockchain-inspired voting to ensure the reliability of the analysis by using a decentralized decision-making process. To avoid non-terminating loops led by common circular dependency in MSA, we objectively limit steps and standardize task processing through *Agent Workflow*. We propose a pioneering framework, **m**ulti-**A**gent **B**lockchain-inspired **C**ollaboration for root cause analysis in micro-services architecture (MABC), where multiple agents based on the powerful LLMs follow *Agent Workflow* and collaborate in blockchain-inspired voting. Specifically, seven specialized agents derived from *Agent Workflow* each provide valuable insights towards root cause analysis based on their expertise and the intrinsic software knowledge of LLMs collaborating within a decentralized chain. Our experiments on the AIOps challenge dataset and a newly created Train-Ticket dataset demonstrate superior performance in identifying root causes and generating effective resolutions. The ablation study further highlights *Agent Workflow*, multi-agent, and blockchain-inspired voting is crucial for achieving optimal performance. MABC offers a comprehensive automated root cause analysis and resolution in micro-services architecture and significantly improves the IT Operation domain. The code and dataset are in https://github.com/knediny/mABC.

1 Introduction

Micro-services architecture (MSA) decomposes an application into a series of independent nodes, interacting through lightweight communication mechanisms (Zhang et al., 2021a; Kim et al., 2013; Alquraan et al., 2018). A key component in maintaining MSA is Root cause analysis (RCA), which aims to find the root cause of alert events and enhance system robustness plays a significant role in avoiding data leaks and program failure analysis (Lin et al., 2018; Ma et al., 2020a).

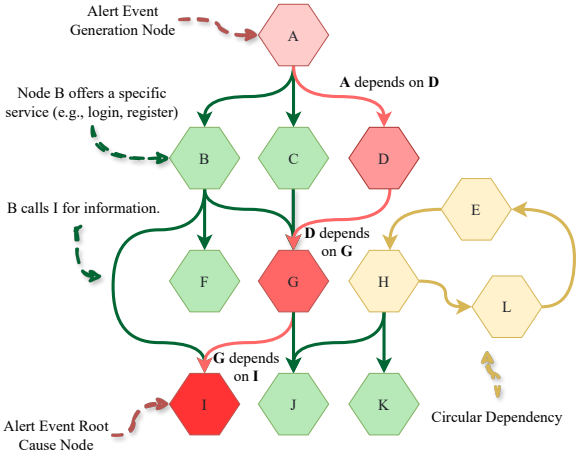

Figure 1: Example of root cause analysis in MSA. Each node corresponds to a specific service in the system (e.g., login, register). Edge $B \rightarrow I$ represents that service I relies on the information provided by service B. Alert event arises on node A while alert event root cause node is I with fault propagation path $I \rightarrow G \rightarrow D \rightarrow A$ where a challenge circular dependency of $H \rightarrow E \rightarrow L \rightarrow H$.

Compared with traditional architectures only containing one central service, RCA in Micro-services architecture (MSA) has become extremely difficult as faults continue to propagate between service nodes and alerts become increasingly complex (Jamshidi et al., 2018; Liu et al., 2020). In Figure 1, alert event arises on A, while the alert event root cause node is I with fault propagation path $I \rightarrow G \rightarrow D \rightarrow A$. RCA identifies the root cause of faults, I here, by tracing back to the origin node and even further analyzing metrics. Existing approaches such as TraceAnomaly (Liu et al., 2020),

*Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2024, pages 4017–4033
November 12-16, 2024 ©2024 Association for Computational Linguistics

and MEPFL (Zhou et al., 2019) with lack of mechanism are unable to handle **circular dependencies** (e.g. $H \rightarrow E \rightarrow L \rightarrow H$) in Figure 1 well and rely heavily on supervised training processes. Large language models (LLMs) like GPT (OpenAI, 2023) and their integration with multi-agent exhibit remarkable analytical and problem-solving capabilities, which are essential for identifying and addressing the root cause of fault in complex MSA (Wei et al., 2022a; Kojima et al., 2022; Wei et al., 2022b; Yao et al., 2023). Although RCA-Copilot (Chen et al., 2023), RCAgent (Wang et al., 2023b), and D-Bot (Zhou et al., 2023) have improved RCA tools with event matching and information aggregation, they struggle with **the hallucination problem** and the common **cross-node fault** (e.g. $I \rightarrow G \rightarrow D \rightarrow A$ in Figure 1) in MSA.

To tackle the above issues, we introduce MABC, a groundbreaking framework designed to revolutionize RCA. To solve diverse cross-node faults, we introduce multi-agents with diverse expertise and extensive software knowledge to analyze a wide range of data and navigate through node dependencies, which fully consider the propagation of failures in dependencies. To mitigate hallucination in LLMs, we integrate a blockchain-inspired voting system in the MABC that uses multi-agent collaboration and community voting for high-quality content assurance. Inspired by blockchain governance (blockchain, 2023; wiki, 2023), this process is transparent and community-driven, enhancing decision-making correctness through a decentralized structure. The MABC employs dynamic weight adaptation for fairness and includes penalties for inactive or inaccurate agents, alongside a cap on weight concentration. This ensures accuracy, fairness, and reliability in content generation through decentralized, professional multi-agent assessments and repeated verifications. To address the non-terminating loop led by circular dependency, we objectively limit the number of steps and standardize task processing through *Agent Workflow* based on task difficulty and dynamic context perception. By harnessing the power of LLMs within multi-agent blockchain-inspired collaboration, MABC conducts RCA and resolution development in MSA, unlike previous methods. Specifically, 1) An alert event arises due to access function blockages or monitoring system alarms in MSA. 2) *Alert Receiver* ($\mathscr{A}_1$) chooses and forwards the alert event with the highest priority. 3) *Process Scheduler* ($\mathscr{A}_2$) divides unfinished RCA into sub-tasks, handled by *Data Detective* ($\mathscr{A}_3$), *Dependency Explorer* ($\mathscr{A}_4$), *Probability Oracle* ($\mathscr{A}_5$), and *Fault Mapper* ($\mathscr{A}_6$) for various requests. 4) *Solution Engineer* ($\mathscr{A}_7$) develops resolutions referencing previous successful cases.

Experimental results on the public AIOps challenge dataset and our created train-ticket dataset demonstrate superior performance in identifying root causes and effective resolution development, compared to existing strong baselines. The ablation study further highlights *Agent Workflow*, multi-agent, and blockchain-inspired voting is crucial for achieving optimal performance. Generally, the main contributions of this work are as follows:

- **Multi-Agent Framework in RCA**: Different from previous works designed for single node fault, we proposed framework MABC driven by LLM and multi-agent collaboration standardized by *Agent Workflow* performs RCA and resolution development in complex MSA scenarios, which will be open-sourced first.

- **Blockchain-Inspired Voting**: By employing blockchain-inspired voting with multi-agent collaboration in MABC, dynamically adjusted weights based on *contribution index* and *expertise index* of agents ensure the accuracy and reliability of content generation.

- **Impressive Evaluation**: Superior performance in identifying root causes and resolution development effectively both on the public AIOps challenge dataset and our created Train-Ticket dataset.

2 Methodology

2.1 Overview

In this section, we provide an overview of MABC, specifically engineered to pinpoint the root causes of alert events in a complex MSA. Illustrated in Figure 2, MABC introduces seven agents: *Alert Receiver*, *Process Scheduler*, *Data Detective*, *Dependency Explorer*, *Probability Oracle*, *Fault Mapper*, and *Solution Engineer*. These agents collaborate transparently and equally, invoking each other to address alert events in the MABC pipeline. In MSA, alert events can arise from user-side blocked function access and monitoring system alarms, such as increased login response times and network latency in the login node. The specific case is shown in Figure 5, 6, 7, 8, 9, 10, 11 in Appendix A.

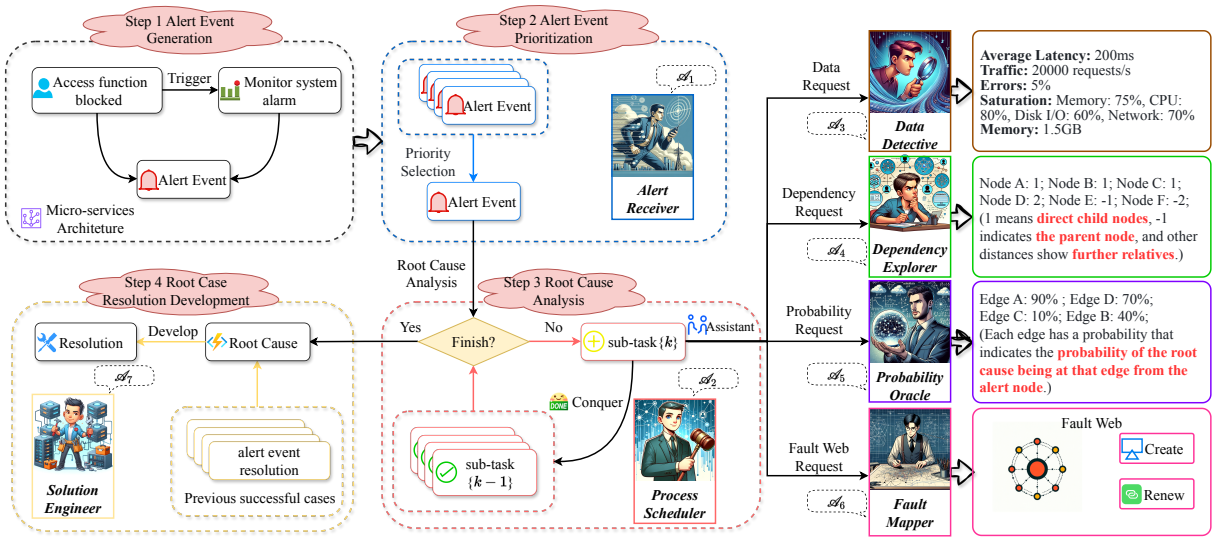

Figure 2: Overview of MABC. Overall pipeline encapsulates the flow from alert inception to root cause analysis within MABC. 1) An alert event arises due to access function blockages or monitoring system alarms in MSA. 2) *Alert Receiver* ($\mathscr{A}_1$) forwards and chooses the alert event with the highest priority. 3) *Process Scheduler* ($\mathscr{A}_2$) divides unfinished root cause analyses into sub-tasks, handled by *Data Detective* ($\mathscr{A}_3$), *Dependency Explorer* ($\mathscr{A}_4$), *Probability Oracle* ($\mathscr{A}_5$), and *Fault Mapper* ($\mathscr{A}_6$) for various requests. 4) *Solution Engineer* ($\mathscr{A}_7$) develops resolutions for the root cause referencing previous successful cases.

2.2 Agent Workflow

In Figure 3, *Agent Workflow* enables all agents to complete their tasks effectively, adhering to a prescribed methodology. For questions that require real-time data or additional information, *Agent Workflow* activates the ReAct answer workflow, which involves an iterative cycle of thought, action, and observation until a satisfactory answer is reached. Conversely, when no external tools are necessary, *Agent Workflow* defaults to the direct answer workflow, where responses are directly formulated based on the prompt provided. It is worth noting that to address the non-terminating loop led by circular dependency, we terminate the process at 20 steps. The prompt example is shown in Figure 13, 14 in Appendix D.

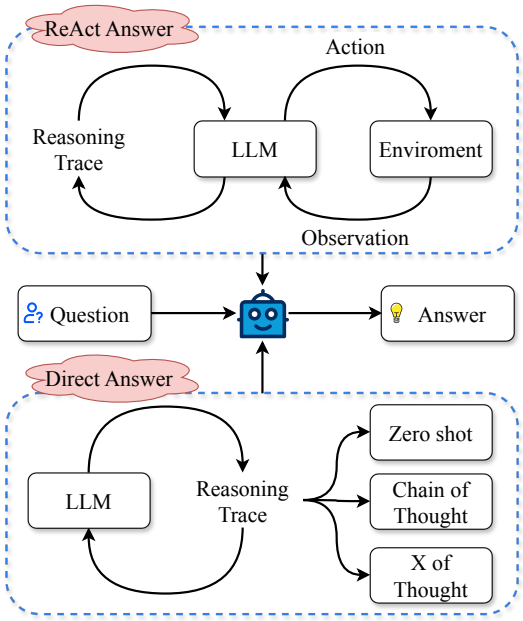

Figure 3: Two distinct workflows of agent.

2.3 Multi-Agent

In this section, we provide a thorough introduction of agents in MABC. The role description are shown in Figure 15, 16, 17, 18, 19, 20, 21 in Appendix E and tools are shown in Table 9 in Appendix H.

2.3.1 Alert Receiver ($\mathscr{A}_1$)

In Figure 2, the responsibility of *Alert Receiver* is to sort the received alert events based on the time, urgency, and scope of impact. After determining the priority of the alert events, *Alert Receiver* dispatches the most urgent and widely impacting alert events to *Process scheduler* further processing following the pipeline.

2.3.2 Process Scheduler ($\mathscr{A}_2$)

In Figure 2, when an alert arrives at *Alert Receiver*, *Process Scheduler* engages specialized agents for tasks like data gathering, fault web updates, dependency analysis, and probability scoring. It forwards critical insights to *Solution Engineer* for resolution.

After each sub-task, it checks for root cause identification. If unresolved, it iterates by generating new sub-tasks and seeking further agent assistance until the root cause is determined. Finally, *Process Scheduler* provides the root cause, an updated fault web, and some resolutions, concluding handling process and preparing for the next alert.

2.3.3 Data Detective ($\mathscr{A}_3$)

In Figure 2, *Data Detective* collects data from designated nodes within specified time windows as directed by the *Process Scheduler*. To ensure thorough analysis and maximize informational value, it excludes non-essential data and processes key metrics like average latency, traffic volume, error rates, resource saturation, and concurrent user counts into charts and reports. This approach simplifies data handling for LLMs, streamlining the task of *Data Detective* and enhancing efficiency in data exploration and analysis within MABC.

2.3.4 Dependency Explorer ($\mathscr{A}_4$)

In Figure 2, *Process Scheduler* sends a *Dependency Request* to *Dependency Explorer* to query dependencies among micro-services nodes, including the specific node and alert time. *Dependency Explorer* identifies direct and indirect dependencies based on global topology and calls within the time window. This is crucial for tracing fault paths, marking impacted nodes, and facilitating further root cause analysis and resolution.

2.3.5 Probability Oracle ($\mathscr{A}_5$)

In Figure 2, *Probability Oracle* assesses the failure probability of nodes. Inaccessible nodes get a high default failure probability, while accessible nodes are evaluated based on performance metrics like response time, error rate using a computational model. By analyzing data correlations, such as a high Pearson correlation coefficient indicating a link between response time and error rate, *Probability Oracle* adjusts failure probabilities of correlated nodes increasingly while decreasing on other nodes. These probabilities are sent to *Process Scheduler*, aiding in updating fault web, root cause analysis, and resolution development.

2.3.6 Fault Mapper ($\mathscr{A}_6$)

In Figure 2, when *Fault Web* needs to be updated, *Process Scheduler* issues a *Fault Web Request*, which includes nodes and their corresponding fault probabilities. *Fault Mapper* creates or renews *Fault Web* based on this information to visually represent the fault probabilities between different nodes. *Fault Web* not only displays the alert source node but also depicts other related nodes and the fault probabilities of their connecting edges. *Fault Mapper* ensures that *Process Scheduler* can make decisions based on the most up-to-date information, thereby guiding *Solution Engineer* to develop appropriate resolutions.

2.3.7 Solution Engineer ($\mathscr{A}_7$)

In Figure 2, *Solution Engineer* receives *Root Cause Analysis and Solution Requests* from *Process Scheduler* and then decides the final root cause analysis and development of solutions based on the available node data. *Solution Engineer* performs node-level analysis to confirm the nodes affected to be repaired by the MCA when node downtime data is unavailable. If node data is available, *Solution Engineer* performs metric-level analysis to find the real problem metric through the correlation between metrics and historical value fluctuations to develop more reasonable solutions like increasing disk throughput for high read and write latency. *Solution Engineer* also references previous successful cases, like in Table 1, to guide the development of the current solution and the conclusion of the process ensuring that the proposed resolution is practical and effective.

2.4 Blockchain-Inspired Voting

2.4.1 Blockchain Communication

To mitigate the hallucination of LLM and avoid falling into non-terminating loops, we have designed *blockchain-inspired voting* as a reflection for any answer to any question from any agent. After the agent answers, all other agents decide whether to initiate a poll and obtain the result through weighted voting. Answers that do not initiate a poll process or pass in the poll are considered to be of high quality due to the majority approval of the agents, while answers that fail to pass will be regenerated by the author agent to improve the quality. The agents in the MABC are transparent and equal to each other, despite their different responsibilities, and compose a decentralized structure *Agent Chain*. Additionally, although *Agent Chain* lacks the implementation of a Byzantine fault-tolerant system, it is still very robust for driven by *Agent Workflow* to avoid the generation of false messages. Inspired by the governance guidelines of decentralized best practice blockchain, we choose on-chain

Alert Events	Description
Tablespace High Utilization	Indicates extensive data occupation in tablespace, potentially degrading database performance.
Database Connectivity Fault	Signifies possible connection issues due to excessive connections, impacting response and transactions.
CPU Resource Insufficiency	High session average CPU time suggest significant CPU occupation, risking performance and stability.
Memory Overflow	Shows memory usage exceeding safe limits, risking performance degradation or crashes.
Disk IO Performance Fault	Abnormal increase in physical read rates, indicating potential disk IO issues.

Table 1: Examples of Alert Events

governance to allow participants to trust each other and leave decision-making power in the hands of decentralized entities. More detailed rules description is shown Figure 22 Appendix F.

2.4.2 Voting Weights

Voting weight is determined by $w_c \cdot w_e$, where *contribution index* (w_c) reflects activity level, and *expertise index* (w_e) reflects professionalism.

The *contribution index* w_c is updated as follows:

$$w_c = \min\left(w_c \cdot (1-\delta) + \Delta w_c,\ w_{c_{\max}}\right) \tag{1}$$

were w_c starts at 1.0. The decay rate δ ranges from 0 to 0.03, applied after each voting event to encourage ongoing contribution and prevent power concentration. Δw_c is an increase of 0.1 from voting participation and proposal submission. $w_{c_{\max}}$ is set to 1.5 to ensure fairness.

The *expertise index* w_e is governed by:

$$w_e = \min\left(w_e + \Delta w_e,\ w_{e_{\max}}\right) \tag{2}$$

where w_e does not decay automatically, reflecting accumulated expertise. Δw_e increases by 0.01 if the agent's vote aligns with the final outcome and decreases by 0.01 otherwise. $w_{e_{\max}}$ is also set to 1.5 to prevent disproportionate influence.

The voting weight system balances activity and expertise to ensure fairness. The contribution index (w_c) starts at 1.0, increases by 0.1 for each vote or proposal, and decays by up to 0.03 after each voting event to encourage ongoing participation, capped at 1.5. The expertise index (w_e) increases by 0.01 if an agent's vote aligns with the outcome and decreases by 0.01 otherwise, capped at 1.5, reflecting professionalism without decay. This system rewards both active engagement and accurate contributions, preventing power hoarding and reckless voting, while maintaining a balanced and fair decision-making process.

2.4.3 Voting Outcome Determination

The support rate (s) and participation rate (p) are defined as:

$$s = \frac{\sum_{i=1}^{n} \mathbf{1}(w_i)}{\sum_{i=1}^{n} w_i} \tag{3}$$

$$p = \frac{\sum_{i=1}^{n} \mathbf{1}'(w_i)}{\sum_{i=1}^{n} w_i} \tag{4}$$

where n is the total number of voting agents, vote_i is the vote of the i-th agent, and w_i is the weight of the i-th vote. A proposal passes if $s \geq \alpha$ & $p \geq \beta$, where α and β are predefined thresholds (e.g., 0.5). The indicator function $\mathbf{1}(\cdot)$ outputs w_i if the i-th agent votes **For**, and 0 otherwise. The indicator function $\mathbf{1}'(\cdot)$ outputs w_i if the i-th agent votes **For** or **Against**, and 0 for **Abstain**.

2.4.4 Voting Process

On *Agent Chain*, every agent is entitled to participate in voting. The voting process works in Figure 4: When $\mathscr{A}_x \in \{\mathscr{A}_i\}_{i=1}^{7}$ gets an answer A for question Q, all agents on the chain will examine A and face the choice of whether to initiate a vote on $X - Q - A$. If no Agent initiates a vote, the answer is accepted. If $\mathscr{A}_y \in \{\mathscr{A}_i\}_{i=1}^{7}$ requires a vote, all agents on the agent chain will vote on $\mathscr{A}_y - \mathscr{A}_x - Q - A$, with the voting options being **For**, **Abstain**, and **Against**. If the vote passes, $\mathscr{A}_x$ will re-answer Question Q to generate a new Answer A'. More detailed description and case are shown in 23, 24, 25 Appendix F.

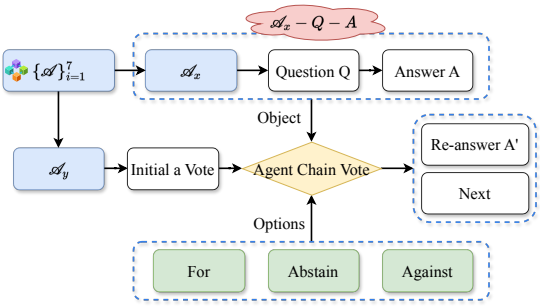

Figure 4: Vote process on *Agent Chain*

Process	Description
Admin Operations	Admin login, site, and route addition, train information addition, user and contact addition, multiple queries (routes, trains, sites, etc.), updates, deletions, repeat queries.
Normal Flow	User registration and login, ticket availability search, ticket booking, order status refresh, order payment, and ticket check-in.
Re-book Flow	Registration and login, availability search, booking, latest order status refresh, re-booking, new order payment (if applicable), check-in.
Re-Book Fail Flow	Registration and login, availability search and booking, order status refresh, successful first re-booking, order payment, failed second re-booking attempt.
Search Fail to Add	User registration and login, failed ticket search (due to missing stations), admin adds missing info, ticket research and booking, latest order status refresh.
Consign Preserve	User registration and login, ticket search and booking, order status refresh, order payment, luggage consignment addition, and check-in.
Preserve Successfully	User registration and login, ticket availability search and booking, order status refresh, payment, and check-in.

Table 2: Train-Ticket Process Descriptions

3 Experiments

3.1 Datasets

Train-Ticket Dataset. We curate our dataset on Train-Ticket (Zhou et al., 2018; Li et al., 2022a), an open-source MSA from Fudan University. We designed 7 processes and 100 virtual users to simulate real operations. Table 2 details each process, which includes operations like registration, login, querying, booking, and ticket changes. Users randomly select processes to cover various scenarios. Specifically, we introduce faults by ChaosBlade (alibaba, 2021) into the system as outlined in Table 3. Please refer to Appendix B for more details.

Category	Case Examples
Network	Packet loss, Frequent retransmission, DNS failures, bandwidth saturation, high TCP connection setup delays
Storage	High I/O latency
CPU	High CPU usage by code, CPU frequently grabs
Memory	High frequency of FULL GC, memory frequently grabs
Code	Exceptions thrown by error codes, HTTP requests, returning error codes

Table 3: Types of Faults Injected in the Experiment

AIOps Challenge Dataset. *2020 AIOps International Challenge Dataset* aims to discover alert events and their root causes in micro-service applications, such as cloud platform services, which include containers, service meshes, micro-service, and variable infrastructures. More details are in Appendix C.

3.2 Evaluation Metrics

Root Cause Result Accuracy (RA) : Following the previous work (Liu et al., 2023; Zhou et al., 2023), we use result accuracy (RA) to quantify the precision of MABC in finding the root cause.

$$\text{RA} = \frac{A_c - \sigma \cdot A_i}{A_t} \tag{5}$$

where A_c denotes the number of correct causes, A_t denotes the total number of causes, A_i denotes the number of wrongly detected causes, and σ is a hyper-parameter with 0.1 as the default value because we recognize *redundant causes is less harmful than missing causes*. Therefore, we limit the identification to a maximum of 4 root causes for an anomaly.

Root Cause Path Accuracy (PA) : We use the root cause path accuracy (PA) metric, which aims to measure the effectiveness of MABC in tracing the correct path from the symptoms (alerts) back to the root causes. The formula for PA similar to RA focuses on path accuracy as:

$$\text{PA} = \frac{P_c - \tau \cdot P_i}{P_t} \tag{6}$$

where P_c denotes the number of correctly identified paths leading to the root cause, P_t is the total number of actual root cause paths present, P_i denotes the number of incorrectly inferred paths, which do not align with the actual root cause paths, and τ is a hyper-parameter designed to penalize the inaccuracies in path inference, with a default value of 0.2, reflecting the understanding that inaccurately inferred paths are less detrimental than completely missing the correct paths, but there is a stronger emphasis on precision due to the potential complexity and relevance of paths.

3.3 Baselines

We choose decision tree (Abdallah et al., 2018), TraceAnomaly (Liu et al., 2020) and MEPFL (Zhou et al., 2019) as unsupervised baselines to compare MABC. For ReAct (Yao et al., 2023), we implement by Langchain (langchain ai, 2023). It is worth noting that RCACopilot (Chen et al., 2023) and RCAgent (Wang et al., 2023b) are not open-source, D-Bot (Zhou et al., 2023) is not suitable for MSA.

3.4 Implementation and Configuration

We implement MABC on Ubuntu 22.04, equipped with an Intel Xeon (R) Gold 6348 CPU @2.60GHz, eight NVIDIA H800 GPUs (80 GB), and 528 GB

Model	Base	Train-Ticket			AIOps			Average
		RA	PA	Average	RA	PA	Average	
Decision Tree	-	36.8	34.7	35.8	28.3	26.7	27.5	31.6
TraceAnomaly	-	25.3	23.5	24.4	20.1	18.9	19.5	22.0
MEPFL	-	30.3	29.1	29.7	33.7	29.7	31.7	30.7
ReAct	GPT-3.5-Turbo	31.8	26.8	29.3	25.1	22.7	23.9	26.6
ReAct	GPT-4-Turbo	43.0	38.9	41.0	37.5	34.4	36.0	38.5
MABC	Llama-3-8B-Instruct	46.1	40.9	43.5	43.0	39.9	41.5	42.5
MABC	GPT-3.5-Turbo	48.1	42.8	45.5	41.1	36.7	38.9	42.2
MABC	GPT-4-Turbo	**54.4**	**48.2**	**51.3**	**45.5**	**39.3**	**42.4**	**46.9**

Table 4: Main Results On Train-Ticket Dataset and AIOps challenge Dataset

of memory. The software setup includes NVIDIA-SMI version 535.104.05 and CUDA 12.3. We set temperature as 0.6 for LLMs.

3.5 Main Results

Based on the results in Table 4, we can see that baselines such as Decision Tree(Abdallah et al., 2018), TraceAnomaly(Liu et al., 2020), and MEPFL(Zhou et al., 2019) achieved average performance scores ranging from 16.0 to 26.7 on the Train-Ticket dataset and AIOps Challenge dataset. In comparison, ReAct(Yao et al., 2023) with GPT-3.5-Turbo and GPT-4-Turbo showed improvements with average scores of 21.6 and 27.5, respectively. However, our proposed MABC significantly outperformed all the baseline models and ReAct with GPT-4-Turbo, achieving an impressive average score of 64.9. This indicates that our framework MABC has a strong predictive capability and robustness in detecting faults and anomalies in both datasets. The substantial improvement over the baselines demonstrates the effectiveness and superiority of our approach in this context.

4 Analysis

4.1 Decision Efficiency.

Following RCAgent (Wang et al., 2023b), we use pass rate (PR) and average path length (APL) to evaluate the thinking trajectory steps of MABC in accomplishing the task, considering the validness of action trajectories and stability of the autonomous agent. PR calculated by $\frac{N_p}{N_t}$, where N_p denotes the number of trajectories completed within θ steps, θ typically set to 15, and N_t denotes the total number of trajectories. Besides, APL is denoted by $\frac{\sum_{k=1}^{N_p} L_k}{N_p}$, where L_k denotes the path length of the k-th successful trajectory.

Model	Base	Train-Ticket		AIOps	
		PR	APL	PR	APL
Decision Tree	-	62.4	12.1	53.8	13.4
TraceAnomaly	-	25.3	20.3	31.1	19.1
MEPFL	-	33.3	19.2	37.1	18.7
ReAct	GPT-3.5-Turbo	41.7	15.9	38.0	16.2
ReAct	GPT-4-Turbo	47.1	13.9	44.2	14.3
MABC	Llama-3-8B-Instruct	56.1	14.8	46.1	17.7
MABC	GPT-3.5-Turbo	58.1	13.8	51.1	14.7
MABC	GPT-4-Turbo	**73.0**	**10.4**	**68.8**	**11.7**

Table 5: Decision Efficiency Evaluation

Model	Base	R-Useful (Train)	R-Useful (AIOps)
Decision Tree	-	-	-
TraceAnomaly	-	-	-
MEPFL	-	-	-
ReAct	GPT-3.5-Turbo	2.1	2.1
ReAct	GPT-4-Turbo	2.4	2.3
MABC	Llama-3-8B-Instruct	3.3	2.7
MABC	GPT-3.5-Turbo	3.1	3.2
MABC	GPT-4-Turbo	**4.2**	**3.6**

Table 6: Human Evaluation

In Table 5, the results confirm that MABC exhibits superior decision efficiency, as evidenced by its high pass rate (PR) and low average path length (APL) across both datasets, which demonstrates that MABC not only completes tasks with a higher probability but also does so with fewer steps, indicating a more efficient and stable decision-making process. The GPT-4-Turbo variant, in particular, showcases the most effective decision trajectories, suggesting that MABC is highly capable of generating efficient actions in the context of root cause analysis in a MSA.

4.2 Human evaluation.

In Appendix G, we evaluated 200 randomly selected cases, focusing on root causes, pathways, and resolutions. Ten AIOps experts rated each case on a scale of 1 (very useless) to 5 (very useful), and

Model	Train-Ticket					AIOps				
	RA	PA	PR	APL	R-Useful	RA	PA	PR	APL	R-Useful
MABC	**54.4**	**48.2**	**73.0**	**10.4**	**4.2**	**45.5**	**39.3**	**68.8**	**11.7**	**3.6**
MABC w/o *Agent Workflow*	46.2	38.7	67.7	11.8	3.5	36.6	34.3	61.3	11.7	3.3
MABC w/o *Multi-Agent*	38.4	33.0	52.9	13.7	2.8	32.4	28.8	50.1	13.7	2.7
MABC w/o Voting	44.8	39.9	65.7	10.9	3.3	40.1	36.7	68.0	10.2	3.4

Table 7: Component Impact Evaluation

we averaged these ratings to derive the *Resolution Evaluation Metrics (R-Useful)* score.

Table 6 highlights a clear preference among experts for the resolutions generated by the MABC model, especially when enhanced with GPT-4-Turbo. The higher R-Useful scores for MABC with GPT-4-Turbo across both datasets underscore its ability to produce highly useful solutions that align well with expert expectations in the AIOps domain. In contrast, the moderate R-Useful scores for ReAct indicate its limited effectiveness in meeting the nuanced needs of AIOps experts. Decision tree, TraceAnomaly, and MEPFL are not evaluated for R-Useful due to their inability to generate resolutions. Overall, the human evaluation confirms that MABC with GPT-4-Turbo excels in creating expert-aligned solutions, showing significant potential for improving decision-making and productivity in AIOps.

4.3 Component Impact.

In this section, we verify the impact of three components in MABC with GPT-4-Turbo, i.e., MABC without *Agent Workflow* (based on ReAct rather than *Agent Workflow*), MABC without *Multi-Agent* (*Agent Workflow*), and MABC without Blockchain-Inspired Voting.

Table 7 shows that MABC in its complete form excels across all metrics on both datasets, highlighting the necessity of integrating all components. Removing *Agent Workflow* significantly reduces performance, indicating its crucial role. Limiting the framework to a Single Agent results in the lowest scores, severely diminishing its capability for AIOps tasks. Excluding the Blockchain-Inspired Voting component also decreases performance, though less critically, underscoring its role in refining and validating resolutions. The evaluation underscores the importance of each component: *Agent Workflow* provides a structured approach, the *Multi-Agent* architecture captures diverse perspectives, and the Blockchain-Inspired Voting mechanism ensures consensus and reliability. Together, these components synergize to enhance performance of MABC, making it a robust tool for root cause analysis in MSA.

5 Related Work

5.1 Root Cause Analysis in Micro-Services Architecture

Root cause analysis (RCA) in large systems, particularly within MSA, is crucial in AIOps (Alquraan et al., 2018; Guo et al., 2024; Zhang et al., 2021b; Liu et al., 2019; Lou et al., 2020). RCA tasks focus on logs, metrics, and traces, with various studies proposing methods for each data source (Guo et al., 2021b; Leesatapornwongsa et al., 2017; Liu et al., 2023). Techniques include identifying failure patterns (Ma et al., 2020b; Zhang et al., 2021a) and exploring service dependency graphs using metrics and traces (Ma et al., 2020a; Li et al., 2022b). Advanced methodologies, particularly NLP for log analysis and anomaly detection, are emphasized (Ghosh et al., 2022; Guo et al., 2023a). Machine learning has been leveraged for log analysis (Locke et al., 2021; Guo et al., 2021a; Gao et al., 2018), and LLMs are used to enhance RCA performance (Zhang et al., 2024b).

5.2 LLM in Micro-Services Architecture

The rapid advancements in language modeling, particularly through Transformer-based architectures and LLMs like GPT-4 and PaLM (OpenAI, 2023; Anil et al., 2023; Guo et al., 2023b), have significantly impacted natural language processing and facilitated their use in complex MSA (Aghajanyan et al., 2023; Kaplan et al., 2020; Park et al., 2023). Integrating LLMs with external tools and APIs enhances their functionality in cloud RCA (Qin et al., 2023), improving log analysis and anomaly detection. LLMs are being explored as core intelligence in autonomous multi-agent systems (Wang et al., 2024; Zhang et al., 2024a), enabling effective environment interaction (Wei et al., 2022b; Ouyang

et al., 2022). This spans tasks from toy examples to real-world cloud RCA, highlighting LLM versatility in dynamic environments (Wang et al., 2023a). The shift towards LLMs in MSA promotes greater autonomy, intelligence, and efficiency (Chen et al., 2019, 2023), driving the growth of end-to-end intelligent operation and maintenance with tasks like database diagnosis, event processing, and RCA (Wang et al., 2023b; Zhou et al., 2023).

6 Conclusion

In this paper, we introduce MABC, a framework that improves alert events resolution in complex MSA by combining multi-agent systems, LLMs, and blockchain voting. We also develop the train-ticket benchmark, an open-source dataset for RCA in MSA. Experimental results on the AIOps challenge and train-ticket datasets show MABC's effectiveness in identifying root causes and providing solutions, with the *Agent Workflow* and voting mechanism being crucial. MABC enhances root cause analysis, boosting system reliability and operational efficiency. Future work will focus on enhancing components, incorporating more data sources, and improving agent collaboration, aiming to make MABC essential for IT operations.

7 Limitations

MABC faces challenges in complexity and scalability as the number of agents and alert events increase, leading to higher computational overhead and longer processing times, necessitating more efficient algorithms and optimization techniques. Its effectiveness also heavily relies on the accuracy and reliability of the data and models used, requiring regular updates and validations to prevent erroneous analyses. Additionally, the blockchain-inspired voting mechanism, while innovative, can be cumbersome and time-consuming, especially with frequent alert events and numerous agents, potentially delaying decision-making. Future iterations should refine the voting process and incorporate mechanisms to detect and mitigate potential biases among agents. Addressing these limitations will be crucial for enhancing MABC's overall performance and reliability.

8 Ethical Considerations

In our study focused on the MABC, we exclusively utilized publicly available data and adhered strictly to ethical and legal standards. Sensitive terminology and methodologies were carefully managed to ensure no breach of privacy or confidentiality. Our commitment to transparency and integrity in handling data ensured that our research remained within ethical boundaries without compromising the effectiveness of our findings.

References

Imad Abdallah, V Dertimanis, H Mylonas, Konstantinos Tatsis, Eleni Chatzi, N Dervili, K Worden, and Eoghan Maguire. 2018. Fault diagnosis of wind turbine structures using decision tree learning algorithms with big data. In *Safety and Reliability–Safe Societies in a Changing World*, pages 3053–3061. CRC Press.

Armen Aghajanyan, Lili Yu, Alexis Conneau, Wei-Ning Hsu, Karen Hambardzumyan, Susan Zhang, Stephen Roller, Naman Goyal, Omer Levy, and Luke Zettlemoyer. 2023. Scaling laws for generative mixed-modal language models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 265–279. PMLR.

alibaba. 2021. https://github.com/chaosblade-io/chaosblade.

Ahmed Alquraan, Hatem Takruri, Mohammed Alfatafta, and Samer Al-Kiswany. 2018. An analysis of network-partitioning failures in cloud systems. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation (OSDI'18)*.

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernández Ábrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan A. Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vladimir Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, and et al. 2023. Palm 2 technical report. *CoRR*, abs/2305.10403.

blockchain. 2023. https://www.blockchain.com/.

Haicheng Chen, Wensheng Dou, Yanyan Jiang, and Feng Qin. 2019. Understanding exception-related bugs in large-scale cloud systems. In *2019 34th*

IEEE/ACM International Conference on Automated Software Engineering (ASE'19).

Yi Chen et al. 2023. Empowering cloud rca with augmented large language models. *arXiv preprint arXiv:2311.00000*.

Yu Gao, Wensheng Dou, Feng Qin, Chushu Gao, Dong Wang, Jun Wei, Ruirui Huang, Li Zhou, and Yongming Wu. 2018. An empirical study on crash recovery bugs in large-scale distributed systems. In *Proceedings of the 26th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering (ESEC/FSE'18)*.

Supriyo Ghosh, Manish Shetty, Chetan Bansal, and Suman Nath. 2022. How to fight production incidents? an empirical study on a large-scale cloud service. In *Symposium on Cloud Computing*, pages 126–141.

Haixuan Guo, Shuhan Yuan, and Xintao Wu. 2021a. Logbert: Log anomaly detection via bert. In *2021 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE.

Hongcheng Guo, Yuhui Guo, Renjie Chen, Jian Yang, Jiaheng Liu, Zhoujun Li, Tieqiao Zheng, Weichao Hou, Liangfan Zheng, and Bo Zhang. 2023a. Loglg: Weakly supervised log anomaly detection via log-event graph construction. *Preprint*, arXiv:2208.10833.

Hongcheng Guo, Xingyu Lin, Jian Yang, Yi Zhuang, Jiaqi Bai, Tieqiao Zheng, Bo Zhang, and Zhoujun Li. 2021b. Translog: A unified transformer-based framework for log anomaly detection. *arXiv preprint arXiv:2201.00016*.

Hongcheng Guo, Jian Yang, Jiaheng Liu, Jiaqi Bai, Boyang Wang, Zhoujun Li, Tieqiao Zheng, Bo Zhang, Junran Peng, and Qi Tian. 2024. Logformer: A pre-train and tuning pipeline for log anomaly detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 135–143.

Hongcheng Guo, Jian Yang, Jiaheng Liu, Liqun Yang, Linzheng Chai, Jiaqi Bai, Junran Peng, Xiaorong Hu, Chao Chen, Dongfeng Zhang, et al. 2023b. Owl: A large language model for it operations. *arXiv preprint arXiv:2309.09298*.

Pooyan Jamshidi, Claus Pahl, Nabor C Mendonça, James Lewis, and Stefan Tilkov. 2018. Microservices: The journey so far and challenges ahead. *IEEE Software*, 35(3):24–35.

Jared Kaplan et al. 2020. Scaling laws for neural language models. *ArXiv*.

Myunghwan Kim, Roshan Sumbaly, and Sam Shah. 2013. Root cause detection in a service-oriented architecture. *ACM SIGMETRICS Performance Evaluation Review*, 41(1):93–104.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*.

langchain ai. 2023. https://github.com/langchain-ai/langchain.

Tanakorn Leesatapornwongsa, Cesar A Stuardo, Riza O Suminto, Huan Ke, Jeffrey F Lukman, and Haryadi S Gunawi. 2017. Scalability bugs: When 100-node testing is not enough. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS'17)*.

Bowen Li, Xin Peng, Qilin Xiang, Hanzhang Wang, Tao Xie, Jun Sun, and Xuanzhe Liu. 2022a. Enjoy your observability: an industrial survey of microservice tracing and analysis. *Empirical Software Engineering*, 27:1–28.

Mingjie Li, Minghua Ma, Xiaohui Nie, Kanglin Yin, Li Cao, Xidao Wen, Zhiyun Yuan, Duogang Wu, Guoying Li, Wei Liu, et al. 2022b. Mining fluctuation propagation graph among time series with active learning. In *Database and Expert Systems Applications: 33rd International Conference*.

JinJin Lin, Pengfei Chen, and Zibin Zheng. 2018. Microscope: Pinpoint performance issues with causal graphs in micro-service environments. In *Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16*, pages 3–20. Springer.

Haopeng Liu, Shan Lu, Madan Musuvathi, and Suman Nath. 2019. What bugs cause production cloud incidents? In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS'19)*.

Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, and Dan Pei. 2020. Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 48–58.

Yuhe Liu, Changhua Pei, Longlong Xu, Bohan Chen, Mingze Sun, Zhirui Zhang, Yongqian Sun, Shenglin Zhang, Kun Wang, Haiming Zhang, et al. 2023. Opseval: A comprehensive task-oriented aiops benchmark for large language models. *arXiv preprint arXiv:2310.07637*.

Steven Locke, Heng Li, Tse-Hsun Peter Chen, Weiyi Shang, and Wei Liu. 2021. Logassist: Assisting log analysis through log summarization. *IEEE Transactions on Software Engineering*, 48(9):3227–3241.

Chang Lou, Peng Huang, and Scott Smith. 2020. Understanding, detecting and localizing partial failures in large system software. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI'20)*.

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

Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, et al. 2020b. Diagnosing root causes of intermittent slow queries in cloud databases. *Proceedings of the VLDB Endowment (VLDB'20)*.

OpenAI. 2023. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Long Ouyang et al. 2022. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*.

Jae Hyun Park et al. 2023. Generative models as multi-agent systems. *Journal of Artificial Intelligence Research*.

Luna Qin et al. 2023. Toolllm: Enhancing large language models with external tools for advanced problem solving. *arXiv preprint arXiv:2305.00000*.

Alex Wang et al. 2023a. Interactive learning with autonomous agents and large language models. *arXiv preprint arXiv:2303.00000*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better llm agents. *Preprint*, arXiv:2402.01030.

Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Lunting Fan, Lingfei Wu, and Qingsong Wen. 2023b. Rcagent: Cloud root cause analysis by autonomous agents with tool-augmented large language models. *Preprint*, arXiv:2310.16340.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022a. Finetuned language models are zero-shot learners. *Preprint*, arXiv:2109.01652.

Jason Wei et al. 2022b. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

wiki. 2023. https://en.wikipedia.org/wiki/blockchain.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Wei Zhang, Xianfu Cheng, Yi Zhang, Jian Yang, Hongcheng Guo, Zhoujun Li, Xiaolin Yin, Xiangyuan Guan, Xu Shi, Liangfan Zheng, and Bo Zhang. 2024a. Eclipse: Semantic entropy-lcs for cross-lingual industrial log parsing. *Preprint*, arXiv:2405.13548.

Wei Zhang, Hongcheng Guo, Anjie Le, Jian Yang, Jiaheng Liu, Zhoujun Li, Tieqiao Zheng, Shi Xu, Runqiang Zang, Liangfan Zheng, and Bo Zhang. 2024b. Lemur: Log parsing with entropy sampling and chain-of-thought merging. *Preprint*, arXiv:2402.18205.

Yingying Zhang, Zhengxiong Guan, Huajie Qian, Leili Xu, Hengbo Liu, Qingsong Wen, Liang Sun, Junwei Jiang, Lunting Fan, and Min Ke. 2021a. Cloudrca: a root cause analysis framework for cloud computing platforms. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*.

Yongle Zhang, Junwen Yang, Zhuqi Jin, Utsav Sethi, Kirk Rodrigues, Shan Lu, and Ding Yuan. 2021b. Understanding and detecting software upgrade failures in distributed systems. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP'21)*.

Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. 2018. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Transactions on Software Engineering*, 47(2):243–260.

Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. 2019. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 683–694.

Xuanhe Zhou, Guoliang Li, Zhaoyan Sun, Zhiyuan Liu, Weize Chen, Jianming Wu, Jiesi Liu, Ruohang Feng, and Guoyang Zeng. 2023. D-bot: Database diagnosis system using large language models. *Preprint*, arXiv:2312.01454.

A Alert Event Case

To clearly and directly demonstrate the process in MABC and intuitively and distinctly display the root cause analysis in a micro-services architecture, we will show how MABC works to handle the case in Figure 1.

MABC uses the alert event arising on node *A* and trace it back to its root cause *I* with fault propagation path $I \rightarrow G \rightarrow D \rightarrow A$. In this simple case, we only demonstrate the summary question and answer for length limitation when an alert arises on node A will be sent to Alert Receiver ($\mathscr{A}_1$).

Alert Receiver ($\mathscr{A}_1$)
Question: An alert has been triggered on node A. What is the priority of this alert based on time, urgency, and scope of impact?
Answer: The alert on node A should be chosen to handle immediately, which has the highest priority due to its immediate impact on critical system functions and multiple nodes compared to other alerts although the latest triggered time.

Figure 5: Determining the priority of an alert on node A based on time, urgency, and impact scope for Alert Receiver ($\mathscr{A}_1$).

Process Scheduler ($\mathscr{A}_2$)
Question: What sub-tasks need to be initiated to analyze the root cause of the alert on node A now based on sub-tasks conquered?
Answer: The following sub-tasks need to be initiated: {call function ask for help to {sub_task_agent} conquer {sub_task_content} subtask.}

Figure 6: Initiating sub-tasks to analyze the root cause of the alert on node A for Process Scheduler ($\mathscr{A}_2$).

Data Detective ($\mathscr{A}_3$)
Question: What are the performance metrics for node A at {yyyy-mm-dd hh-mm-ss} across 3 hours windows?
Answer: The average latency is 300ms, traffic volume is 5000 requests/min, error rate is 5%, resource saturation levels are at 85% for memory, 80% for cpu, 60% for disk I/O, and concurrent user count is 1200. These are some details as follows: {detail_information}

Figure 7: Asking for performance metric of specific node and data collection for Data Detective ($\mathscr{A}_3$).

Dependency Explorer ($\mathscr{A}_4$)
Question: What are the direct dependent nodes of node A and the indirect nodes with a distance of 2?
Answer: In the graph provided: direct dependent nodes of node A: These are nodes {B, C, D} that are directly connected to node A by an outgoing edge. Indirect nodes with a distance of 2 from node A: These are nodes {I, F, G} that can be reached from node A by traversing two edges.
From node B: node I, F, G.
From node C: node G.
From node D: node G.
Thus, the indirect dependent nodes of node A with a distance of 2 are nodes I, F, and G.

Figure 8: Dependency analysis on specific node for Dependency Explorer ($\mathscr{A}_4$).

Probability Oracle ($\mathscr{A}_5$)
Question: What are the fault probabilities for nodes A, D, G, and I based on {information_from_xxx}?
Answer: The fault probabilities are as follows: node A - 10%, node D - 20%, node G - 30%, and node I - 70%.

Figure 9: Fault probabilities analysis on specific nodes for Probability Oracle ($\mathscr{A}_5$).

Fault Mapper ($\mathscr{A}_6$)
Question: Can you update the fault web with the latest fault probabilities for nodes A, D, G, and I follow the information provided by Probability Oracle and Dependency Explorer? {information_from_other_agent}
Answer: The fault web has been updated to show the fault propagation path $I(70\%) \rightarrow G(30\%) \rightarrow D(20\%) \rightarrow A(10\%)$ with respective fault probabilities.

Figure 10: Summarizing information from other agent and update fault web for Fault Mapper ($\mathscr{A}_6$).

Solution Engineer ($\mathscr{A}_7$)
Question: Based on the root cause analysis, alert arises on node A with dependency chain $I \rightarrow G \rightarrow D \rightarrow A$. what resolution strategy should be implemented to resolve the issue on node I?
Answer: The recommended resolution strategy is to increase the resource allocation for node I and optimize its performance metrics. Historical data suggests that similar issues were resolved by upgrading the node's CPU and memory capacity.

Figure 11: Summarizing the process of root cause analysis and develop a solution to handle the fault for Solution Engineer ($\mathscr{A}_7$).

B More Details on Train-Ticket

Train-Ticket is a Kubernetes-deployed train booking system with integrated monitoring and analysis tools, comprising 41 micro-services for high-concurrency functions like ticket query, reservation, payment, changes, and notifications. We create a dataset of 233,111 call chains with 800,656 spans across 112 time periods and 53 nodes, including 900 direct alert events and 294 induced by external nodes.

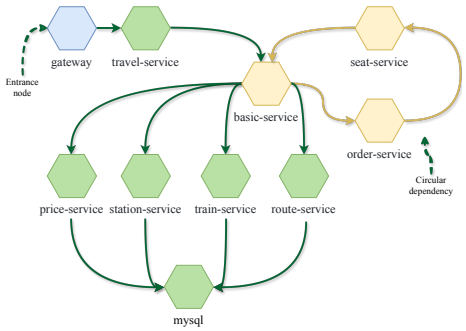

Figure 12: An example of Train-Ticket architecture (query remaining tickets). *basic service* relies on *seat service*, *order service*, and others. A circular dependency of *basic-service→order-service→seat service→basic-service* brings a new challenge for root cause analysis.

C More Details on AIOps Challenge Dataset

In Table 8, the types of alert events mainly include container CPU utilization, container memory utilization, database connection limit, database close, host network delay, and container network loss. All types of alert events are distributed across various nodes of the system. The dataset includes 14 days of system logs totaling 145,907,050 entries.

Alert Node	Alert Events Type	Count
os_021	CPU	8434
docker_006	Database Connectivity	130
docker_006	Database Local Method	7174
os_021	Operate System	6352
docker_008	Database Connectivity	233
docker_008	Database Local Method	7552
docker_005	Database Connectivity	211
docker_005	Database Local Method	9684
os_022	CPU	15099
docker_001	Network	2
os_022	Operate System	109
docker_004	Network	124

Table 8: Summary of Alert Events in 2020 International AIOps Challenge Dataset

D Prompts for Agent Workflow

In this section, we will introduce the workflow prompt for two distinct agent workflows to demonstrate how agents work in their own tasks.

Example Prompt for Direct Answer: {Agent Role Description}. The answer MUST contain a sequence of bullet points that explain how you arrived at the answer. This can include aspects of the previous conversation history. Answer the following question as best you can. So, let's get started!

Figure 13: Example Prompt for Direct Answer.

Example Prompt for ReAct Answer: {Agent Role Description}. You have access to the following tools: {tools}.

Conversations will go on in following format:
Question: the input question you must answer.
Thought: you should always think about what to do.
Action Tool Name: the action tool name to take, should be one of {tool name}.
Action Tool Input: the input to the action tool, should be argument of [Action Tool Name], such as "a=1, b=2" for Action Tool Name "add(a,b)".
Observation: the result of the action.
[Thought / Action Tool Name / Action Tool Input / Observation can be repeated MORE or ZERO times.]
Thought: I now know the final answer.
Final Answer: the final answer to the original input question.

At the point, your answer MUST start with a "Thought".
The answer MUST contain a sequence of bullet points that explain how you arrived at the answer. This can include aspects of the previous conversation history.
Answer the following question as best you can. So, let's get started!

Figure 14: Example Prompt for ReAct Answer.

E Agent Role Description Prompt for Multi-Agent

In this section, we will introduce the agent role description prompt for each agent in multi-agent.

Prompt for Alert Receiver ($\mathscr{A}_1$) Role Description: You are an Alert Receiver. You prioritize incoming alerts based on time, urgency, and scope of impact and dispatch the most urgent and impacting alerts to the Process Scheduler for further processing.

Figure 15: Prompt for Alert Receiver ($\mathscr{A}_1$) Role Description.

Prompt for Process Scheduler ($\mathscr{A}_2$) Role Description: You are a Process Scheduler. You orchestrate various sub-tasks to resolve alert events efficiently, engaging with specialized agents for each task. You ensure that the root cause analysis is iterated and finalized.

Figure 16: Prompt for Process Scheduler ($\mathscr{A}_2$) Role Description.

Prompt for Data Detective ($\mathscr{A}_3$) Role Description: You are a Data Detective. You are adept at collecting and analyzing data from various nodes within a specific time window, and you use tools like the Data Collection Tool and Data Analysis Tool to exclude non-essential data and apply fuzzy matching to focus on critical parameters.

Figure 17: Prompt for Data Detective ($\mathscr{A}_3$) Role Description.

Prompt for Dependency Explorer ($\mathscr{A}_4$) Role Description: You are a Dependency Explorer. You specialize in analyzing the dependencies among internal nodes of the micro-services architecture. You use tools to identify direct and indirect dependent nodes for a specific node, which is vital for identifying fault paths and impacted nodes.

Figure 18: Prompt for Dependency Explorer ($\mathscr{A}_4$) Role Description.

Prompt for Probability Oracle ($\mathscr{A}_5$) Role Description: You are a Probability Oracle. You assess the probability of faults across different nodes within the micro-services architecture. You use computational models to evaluate fault probabilities based on performance metrics and data correlations.

Figure 19: Prompt for Probability Oracle ($\mathscr{A}_5$) Role Description.

Prompt for Fault Mapper ($\mathscr{A}_6$) Role Description: You are a Fault Mapper. You are responsible for visualizing and updating the Fault Web with fault probability information. You create or renew the Fault Web to visually represent the fault probabilities between different nodes.

Figure 20: Prompt for Fault Mapper ($\mathscr{A}_6$) Role Description.

Prompt for Solution Engineer ($\mathscr{A}_7$) Role Description: You are a Solution Engineer. You conduct the final root cause analysis and develop solutions. You perform metric-level or node-level analysis and reference previous successful cases to guide the development of current solutions.

Figure 21: Prompt for Solution Engineer ($\mathscr{A}_7$) Role Description.

F Blockchain Voting

Prompt for Rule Description in Blockchain Voting: Welcome to the governance system of a P2P organization.

Here are the rules for participating in polls:
1. Anonymity: Your identity will be anonymized to protect your privacy.
2. Encrypted Voting: All content will be encrypted when voting or initiating a poll.
3. Anonymous Records: Any records will be anonymized to prevent tracking.
4. Reason Anonymity: They will also be anonymized if you provide reasons when voting or initiating a poll.
5. Result Publication: Only aggregated results will be published without revealing individual voter identities.

You have the right to vote on polls initiated by any member and to initiate a poll to challenge any existing answers. Please follow the provided formats strictly when participating.

Figure 22: Prompt for Rule Description in Blockchain Voting.

Prompt for Initiating a Poll in Blockchain Voting: Welcome to the governance system of a P2P organization. As a member of a P2P organization, you have the right to initiate a vote to challenge everyone's answers. Please Think before you act!!!

Face to {poll_problem}, the answer of {poll_role} expert is as follows: {poll_content}.
Now, you need to decide whether to initiate a poll to challenge the answer of the {poll_role} expert. Please and Must answer in the format:
Poll: Yes/No
Reason: Why you Initiate POLL or NOT

Note: Remember that your decision and reason will be anonymized.

Figure 23: Prompt for Initiating a Poll in Blockchain Voting.

Prompt for Voting in Blockchain Voting: Welcome to the governance system of a P2P organization. As a member of a P2P organization, you have the right to vote for the poll from any member.

The answer of {poll_role} expert facing to {poll_problem} is as follows: {poll_content}. However, somebody challenges the answer and initiates a poll because {anonymous_reasons_for_initiating_poll}. Now you need to answer which options you vote to. Please and Must answer in the format and DO NOT give any reason:
Option: For/Against/Abstain

Note: Remember that your decision and reason will be anonymized.

Figure 24: Prompt for Voting in Blockchain Voting.

Introduction
Scenario: The Probability Oracle determines a high fault probability of 90% for node I.

Poll Initiation Case
Poll Question: "{scenario}. Now, you need to decide whether to initiate a poll to challenge the answer of the Probability Oracle".
Poll Answer Collection: [Yes, No, No, No, No, No] (Probability Oracle does not need to answer)
Poll Reason in Answer Collection: [{yes_reason}]

Voting Case
Voting Question: {scenario}. {poll initiation case and reason anonymous_reasons}. Now you need to answer which options you vote to.
Voting Options: For (agree with the fault probability), Against (disagree and request a re-evaluation), Abstain (choose not to vote)
Voting Answer: [For, For, For, For, Abstain, Against, For]
Voting Weight (only visible to agent self and compute unit):
- contribution index: [1.3, 1.2, 1.3, 0.9, 1.3, 1.2, 0.7]
- expertise index: [1.0, 1.2, 1.2, 1.4, 1.1, 1.2, 1.1]
- voting weight: [1.3, 1.44, 1.56, 1.26, 1.43, 1.44, 0.77]
Voting Outcome:
- support rate: 6.33 / 9.2 = 0.688 >= 0.5
- participation rate: 7.77 / 9.2 = 0.845 >= 0.5
- voting outcome: pass, the fault probability for node I is accepted.

Figure 25: Case in Blockchain Voting by MABC.

G Human Evaluation

Introduction

Thank you for participating in this evaluation. Your expertise is invaluable in assessing the usefulness of the solutions generated by different models for root cause analysis in a micro-services architecture. Please rate each provided solution on a scale of 1 (very useless) to 5 (very useful).

Root Cause Analysis by MABC

Scenario: An application built on a micro-services architecture is experiencing frequent timeouts and slow response times affecting services such as user management and order processing. The monitoring system has flagged high latency in the order processing service, which should be handled now.
Root Cause Node: order database node.
Root Cause: The database connection pool is overwhelmed due to a sudden spike in user activity from the order processing service node.
Pathways: order submit node => order information node => price node => count node => order database node.

Resolutions by MABC

Solution 1: Scale the database vertically to handle more queries per second.
Solution 2: Implement caching for frequently accessed data to reduce the load on the database.
Solution 3: Introduce a queueing mechanism to handle the spike in requests more efficiently.

Evaluation
Root Cause Node Rating: _____ (1: very useless, 2: useless, 3: neutral, 4: useful, 5: very useful)
Root Cause Rating: _____ (1: very useless, 2: useless, 3: neutral, 4: useful, 5: very useful)
Root Cause Pathways: _____ (1: very useless, 2: useless, 3: neutral, 4: useful, 5: very useful)
Root Cause Solution 1: _____ (1: very useless, 2: useless, 3: neutral, 4: useful, 5: very useful)
Root Cause Solution 2: _____ (1: very useless, 2: useless, 3: neutral, 4: useful, 5: very useful)
Root Cause Solution 3: _____ (1: very useless, 2: useless, 3: neutral, 4: useful, 5: very useful)

Figure 26: Human Evaluation Case.

H Tools for Multi-Agent

Agent	Tool Name	Description
Alert Receive	Receive Alert Tool	Receive an alert from the micro-services system and add it to the scheduled queue
	Prioritize Highest Alert Tool	Choose the alert with the highest priority from the scheduled queue based on the trigger time, urgency provided by self, and the number of impact nodes
Process Scheduler	Call For Help Tool	Call for help from other agents by asking a domain question and get a summary of the answer but not a detailed answer efficiently
	Judge Sub-task Tool	Divide the task and conquer it by anything until no sub-task to do
Data Detective	Data Collection Tool	Collects data from nodes within a specific time window
	Data Cleaning Tool	Cleans and preprocesses collected data for analysis by providing useful information but not a list of data
Dependency Explorer	Dependency Query Tool	Identifies direct and indirect dependencies of a node
	Dependency Visualization Tool	Visualizes the dependencies among nodes for caching the result
Probability Oracle	Fault Probability Tool	Evaluates fault probabilities of nodes
	Correlation Analysis Tool	Analyzes correlation between performance metrics of different nodes
Fault Mapper	Fault Web Tool	Visualizes and updates the fault web based on fault probabilities
	Impact Analysis Tool	Analyze the impact of faults on different parts of the system
Solution Engineer	Solution Development Tool	Develops resolutions based on root cause analysis
	Case Reference Tool	References previous successful cases to guide current solution development
General Tools	Metric Explorer	Retrieves node statistics like CPU and disk I/O and memory and so on for a specific time or over a time range
	Alert Aggregation Tool	Aggregates alerts from various sources for unified processing
	JSON Tool	Provide JSON file reading and writing

Table 9: Overview of the tools for agents.