Too Many Cooks: Assessing the Need for Multi-Source Data in Microservice Failure Diagnosis

1st Shenglin Zhang
Nankai University
zhangsl@nankai.edu.cn

2nd Xiaoyu Feng
Nankai University
2120230765@mail.nankai.edu.cn

3rd Runzhou Wang
Nankai University
1120240403@mail.nankai.edu.cn

4th Minghua Ma
Microsoft
minghuama@microsoft.com

5th Wenwei Gu
The Chinese University of Hong Kong
wwgu21@cse.cuhk.edu.hk

6th Yongqian Sun*
Nankai University
sunyongqian@nankai.edu.cn

7th Zedong Jia
Nankai University
2110703@nankai.edu.cn

8th Jinrui Sun
Nankai University
2112360@nankai.edu.cn

9th Dan Pei
Tsinghua University
peidan@tsinghua.edu.cn

***Abstract*—Microservice systems, characterized by their distributed nature and dynamic environments, pose significant challenges for failure diagnosis. Traditional failure diagnosis methods based on a single source of observability data, such as logs, metrics, or traces, fall short due to their inability to manage the complexity of inter-service communications. Consequently, in recent years, many methods based on multi-source observability data have been proposed, which promise a comprehensive analysis by integrating logs, metrics, traces. However, despite the promising potential of multi-source methods, we find that existing works often overlook a critical question: whether multi-source data is truly necessary for failure diagnosis. To address this, we conduct a systematic evaluation of eleven representative failure diagnosis methods based on multi-source observability data, using three public datasets and our own curated dataset. Our experiments focus on multiple aspects of multi-source datasets and methods. The results indicate that the quality of existing open-source datasets is inconsistent, and not all studied methods consistently perform well. Surprisingly, we find that adding more data sources does not necessarily improve the performance of microservice failure diagnosis in some cases.**

***Index Terms*—Microservice systems, Multi-Source Data, Failure Diagnosis**

I. INTRODUCTION

The microservices architecture [1] is a modern software development approach that decomposes traditional monolithic applications into a suite of small, loosely coupled services. Each service can be independently developed, deployed, upgraded, and scaled, thereby improving the overall flexibility and maintainability of complex systems. This architecture has seen widespread adoption in large-scale systems due to its operational benefits. For instance, Tencent's WeChat system operates over 3,000 services on 20,000 machines [2], while Alibaba's e-commerce platform runs more than 30,000 microservices [3]. However, the inherent complexity and dynamism of microservice systems can cause a single-instance failure to cascade, undermining system stability, harming user experience, and incurring considerable financial losses. Therefore, prompt and precise failure diagnosis in microservice environments is critical to preventing widespread outages and minimizing system-level damage.

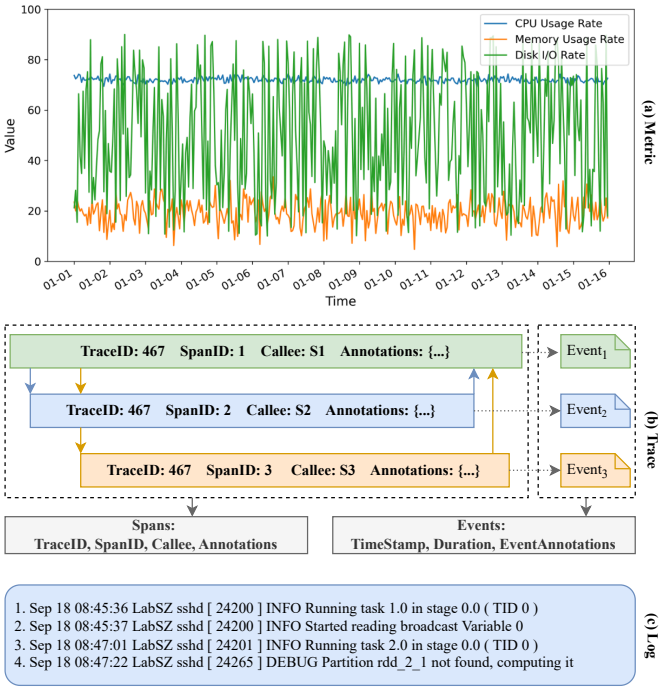

Fig. 1: An example of the multi-source data.

Current research on failure diagnosis, including root cause

*Corresponding author: sunyongqian@nankai.edu.cn

localization and failure classification, predominantly relies on a single source of observability data. For instance, MicroRCA [4] employs metric data, like Fig. 1. (a); TraceRCA [5] is based on trace data, like Fig. 1. (b); and LogCluster [6] utilizes log data, like Fig. 1. (c). However, studies [7], [8] have indicated that relying solely on a single data source is inadequate for accurate failure diagnosis. Firstly, a single failure may impact multiple components of a microservice system, leading to anomalous patterns across various data sources. This implies that exclusive reliance on a single data source may miss critical signals and reduce diagnostic accuracy. Secondly, some failures may not manifest in certain data modalities, rendering methods dependent on them ineffective. Although these studies suggest that multi-source data can improve diagnostic accuracy, others point out that the use of multi-source data may also introduce problems. First, methods based on multi-source data may require offline training and online diagnosis times that are dozens or even hundreds of times greater than those of single-source methods [7]. Additionally, the integration of redundant, noisy, or irrelevant signals can lead to misleading results and incorrect diagnoses. For example, failures stemming from network congestion may not be captured in log data, as they typically occur at the network or infrastructure level, bypassing application-level logging. Blindly integrating logs with other data sources, however, can lead to incorrect failure diagnosis. In recent years, extensive research has been conducted on failure diagnosis in microservice systems using multi-source data, resulting in numerous proposed methods [9], [10], [8], [11], [12], [13], [14], [7], [15], [16], [17], [18], [19]. Although these methods show promising results on their respective datasets, a comprehensive evaluation on public, real-world benchmark datasets is still lacking. Such validation is essential to assess their generalizability across diverse scenarios. Moreover, existing methods often overlook the potential risks of integrating multi-source data, assuming—often blindly—that it will invariably enhance diagnostic accuracy.

To better comprehend the strengths and limitations of existing failure diagnosis studies based on multi-source data and find out whether multi-source data is indispensable for microservice failure diagnosis, we carefully select almost all available multi-source datasets, curate and analyze them, and subsequently conduct comprehensive empirical studies. Our findings reveal that these studies commonly face two issues:

- **Lack of well-curated and publicly usable multi-source observability datasets**: Public multi-source observability datasets often suffer from issues that limit their practical utility in real-world scenarios. One prominent problem is the lack of standardized data organization, including data loss and inconsistencies in collection formats. For instance, the trace data in the Nezha dataset [13] exhibits significant timestamp discrepancies. Moreover, sampling frequency, format, and instrumentation often vary across data sources. Such inconsistencies complicate the integration and unified analysis of heterogeneous data, limiting the generalizability and scalability of downstream diagnostic methods. Another critical challenge arises from inaccurate ground truth annotations. In datasets such as AIOps21[1] and AIOps22[2], Even when failures do occur, their annotated start and end times are often imprecise. This undermines the validity of evaluation and impairs reproducibility in failure diagnosis research. Finally, many failure types appear to be severely underrepresented, resulting in a highly imbalanced class distribution [7]. For example, in the GAIA (Generic AIOps Atlas) dataset[3], most failure cases belong to the "Login failure" type, which restricts the robustness and fairness of models trained on such skewed data distributions.
- **Blindly using multi-source data can lead to misleading results**: While multi-source data theoretically offers a more comprehensive view of system behaviors, improper integration strategies can introduce redundancy, noise, or misleading patterns. First, overlapping events across sources can lead to feature redundancy (e.g., failures being reflected both in logs and traces), which can lead to redundant features. Without proper feature selection, models may overfit to irrelevant correlations instead of capturing true causal signals. Secondly, improper fusion of multi-source data can dilute anomaly signals. For instance, a critical failure may only manifest in logs, while metric or trace data remain normal; when fused indiscriminately, such subtle yet crucial failure logs may be overshadowed. Finally, including unrelated or noisy data sources may degrade model performance. Certain data modalities might be irrelevant to the failure diagnosis task, and incorporating them without discrimination can increase model complexity and reduce generalization. For example, after excluding log data, the F1 score of Diagfusion [7] increases by 10% on the AIOps22 dataset. Additionally, after excluding metric data, the F1 score of CloudRCA [10] increases by 40% on the GAIA dataset. These findings highlight the importance of thoughtful data preprocessing, feature selection, and modality-aware modeling in leveraging multi-source observability data.

The main contributions of this paper are summarized as follows:

- To the best of our knowledge, this is the first empirical study on microservice failure diagnosis methods based on multi-source data. We evaluate eleven representative methods on three public datasets and our own curated dataset. Additionally, we release the implementations of PDiagnose [8], CloudRCA [10], RMLAD [9], TrinityRCL [11], which were not open-sourced, publicly accessible [4] to facilitate further research.
- We construct a new publicly usable multi-source dataset from a newly released microservice benchmark, MicroServo [20], including observability data from ten microservices. As information shown in Table I, Table II,

[1]Repository: https://www.aiops.cn/gitlab/aiops-nankai/data/trace/aiops2021
[2]Repository: https://github.com/AIOps-Lab-NKU/UniDiag
[3]Repository: https://github.com/CloudWise-OpenSource/GAIA-DataSet
[4]Repository: https://anonymous.4open.science/r/Empirical-Study-on-Multi-source-Failure-Diagnosis-49F0

and Fig. 2, the dataset includes metric data collected at 1-second intervals, a diverse range of logs, and trace records, providing a thorough observability of the system. It features a balanced injection of hundreds of failures across seven different types, covering most failure scenarios without data loss. More importantly, the ground truth is accurately recorded. This dataset is now publicly accessible [5] for broader use.

- By conducting comprehensive experimental analysis across different datasets, we find that the performance of the above methods is not uniformly effective across different types of failures. Additionally, failure classification often achieves optimal performance with less training data than root cause localization. Finally, we offer method selection recommendations tailored to different real-world application scenarios, enabling future engineers to reference and swiftly choose the optimal solution that meets their specific needs.

II. BACKGROUND

A. Multi-Source Data

In microservice systems, observability fundamentally depends on the collection and analysis of diverse data sources. Specifically, metrics, traces, and logs constitute the three pillars of system observability [7], each offering unique yet complementary perspectives on system behaviors and failures.

As shown in Fig. 1.(a), **metrics** are numerical measurements continuously collected at fixed intervals, forming time series data that reflect microservice health and performance [18]. These encompass both system-level indicators (e.g., CPU usage and memory consumption) and application-level metrics (e.g., request latency and error rate). In microservice systems, analyzing such metrics allows operators to monitor service behavior trends, detect performance bottlenecks, and identify underperforming or failed service instances.

Fig. 1.(b) illustrates **traces**, which capture the end-to-end execution paths of user requests across distributed microservices. A trace is typically represented as a tree structure, where each node (called a span) records a single invocation between services, including its start time, duration, and status [13]. Traces reveal the causal relationships among microservices and aid in pinpointing the exact service instance or interface where failures or latency spikes occur. In microservice systems where requests may span dozens of services, traces are crucial for understanding execution dependencies and isolating failure components.

As depicted in Fig. 1.(c), **logs** are time-stamped textual records generated by service instances to document discrete events or state transitions. These include business transactions, system state changes, and error messages. Logs are typically semi-structured, containing a timestamp, severity level (e.g., INFO, WARN, ERROR), and a raw message [19]. Unlike metrics and traces, logs offer fine-grained visibility into the internal logic of each microservice and are often the most direct evidence available for root cause diagnosis. In microservice systems where failures can propagate across components, logs are indispensable for reconstructing the failure context and locating the source of failure.

[5]Repository: https://anonymous.4open.science/r/microservo

B. Tasks of Failure Diagnosis

Following DiagFusion's [7] problem statement, we divide failure diagnosis in large-scale microservice systems into two primary tasks: root cause localization (RCL) and failure classification (FC).

RCL In the context of Artificial Intelligence for IT Operations (AIOps), RCL refers to identifying potential root causes of anomalies and assigning a probability to each. This crucial process pinpoints the problematic microservice upon anomaly detection. The objective is to assess the likelihood of each service or instance as the root cause and rank them accordingly. RCL can be performed at three levels: service-level, instance-level, and component-level. Each level offers a different degree of diagnostic precision, ranging from identifying the failure service or instance to pinpointing the exact failed component.

FC FC aims to determine the specific failure type from a predefined category set, enabling operators to take timely and targeted recovery actions.

III. AN EVALUATION OF MULTI-SOURCE DATASETS

A. Selected Datasets

Our research analyzes eight multi-source datasets, seven of which are open-source (excluding AIOps23). The TT1 dataset is sourced from Eadro [14], and TT2 is from Nezha [13]. Both originate from the Train-Ticket open-source project [21] and are collectively referred to as the TT datasets. The AIOps21, AIOps22, and AIOps23 datasets are collected for competition tasks on RCL and FC. The GAIA dataset is primarily generated from the MicroSS microservice simulation system developed by Cloud Wisdom, containing authorized and rigorously anonymized user data. The features of the eight datasets are presented in Table I.

TABLE I: Key features of eight datasets

Dataset	Failure Instances	Ground Truth Accuracy	Data Loss
TT1	81	Yes	No
TT2	45	No	No
SN	36	Yes	No
GAIA	17155	Yes	Yes
AIOps21	159	No	Yes
AIOps22	241	Yes	No
AIOps23	11	Yes	No
MicroServo	210	Yes	No

Among the eight available datasets, four (TT1, SN, AIOps23, and TT2) are excluded from further analysis due to various limitations.

The TT1 and SN datasets, sourced from the paper by Eadro [14], are excluded due to the relatively low number of failure instances and an unusually high frequency of metric data collection. Additionally, the failure instances in these datasets persist for extended periods, often covering more than two-thirds of the total duration, which deviates from the

typical distribution of normal and abnormal data found in most datasets. The significant imbalance, where anomalous data far exceeding normal data, hinders the model's ability to learn the patterns of normal data. As a result, unsupervised models may misinterpret normal data as anomalies and vice versa.

The AIOps23 dataset is excluded due to the scarcity of failure instances, which makes it difficult to split the data into training and testing sets and limits its usefulness for model evaluation, despite its comprehensive and precise multi-source data.

The TT2 dataset, although providing detailed log data that ensures traceability to specific traces, is excluded due to a timestamp discrepancy within the trace data. We suspect this issue arose from a mix-up between Greenwich Mean Time and Asian Time during data collection, causing a 28,800-second discrepancy between certain trace timestamps and their corresponding log timestamps.

TABLE II: Detailed information of the selected datasets

	AIOps21	AIOps22	GAIA	MicroServo
# Services	14	10	10	9
Time Span	240 h	120 h	708 h	72h
Metrics Types	349	296	681	68
Metric Data	12M	18M	217M	58M
Metric frequency	60 s	60 s	60/30 s	1 s
# Log Templates	391	150	66	353
Log Data	21M	26M	87M	22M
# Call Chains	214M	44M	28M	44M
Failure Types	6	15	6	7
Failure Instances	159	241	17,155	210

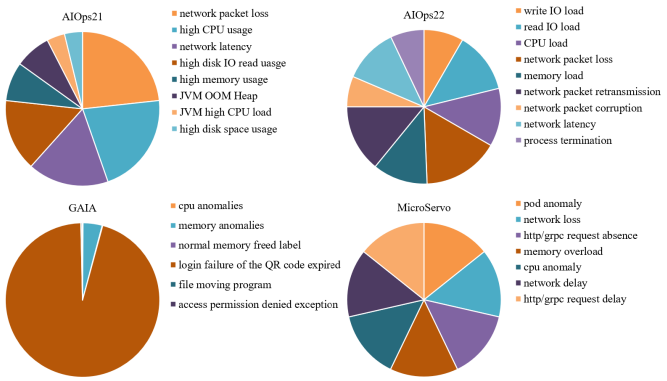

Fig. 2: Proportion of different failure types in the selected datasets ("Failure Type" is the annotated information in the ground truth)

In summary, adhering to the selection criteria mentioned above, we select four (AIOps21, AIOps22, GAIA, and MicroServo) multi-source datasets out of the eight available. Detailed information regarding these four datasets is presented in Table II, and Fig. 2

IV. AN EMPIRICAL STUDY ON FAILURE DIAGNOSIS FOR MULTI-SOURCE DATA

A. Task Classification

According to the task classification outlined in 2.2, the methods selected for this study are categorized into two types: RCL and FC. The features of selected methods are presented in Table III. In the "Data Used" column, "L", "M", and "T" respectively stand for Log, Metric, and Trace data.

TABLE III: Detailed information of selected methods

Method	Data Used	Task	Method Type	Datasets Used	Code & Dataset
RMLAD [9]	L/M	RCL	Unsupervised	TT	No/No
CloudRCA [10]	L/M	FC	Unsupervised	From Alibaba	No/No
PDiagnose [8]	L/M/T	RCL	Unsupervised	AIOps20, AIOps21	No/No
TrinityRCL [11]	L/M/T	RCL	Unsupervised	From Meituan	No/No
MicroCBR [12]	L/M/T	FC	Unsupervised	OB, SS, TT	Yes/No
Nezha [13]	L/M/T	RCL	Unsupervised	OB, TT2	Yes/Yes
Eadro [14]	L/M/T	RCL	Supervised	SN, TT1	Yes/Yes
DiagFusion [7]	L/M/T	RCL/FC	Supervised	GAIA, AIOps21	Yes/Yes
ART [17]	L/M/T	RCL/FC	Selfsupervised	GAIA, AIOps22	Yes/Yes
Medicine [18]	L/M/T	FC	Unsupervised	GAIA, OB, AIOps22s	Yes/Yes
UniDiag [19]	L/M/T	FC	Unsupervised	GAIA, AIOps22	Yes/Yes

*"**Code & Dataset**" indicates whether the method has open-source code or dataset. "**Datasets Used**" refers to datasets reported in the original papers. "**OB**" = Online-Boutique [22], "**SS**" = Sock-Shop [23] "**TT**" = Train-Ticket [21], "**SN**" = Social-Network [24].*

B. RQs On Failure Diagnosis For Multi-source Data

This study aims to evaluate the capabilities of microservice failure diagnosis methods using multi-source data and explore the impact of multi-source data on diagnosis tasks. We design the following research questions:

RQ1: How do the selected methods perform on open-source datasets? While these methods show promising results in experiments, empirical research on their effectiveness with public datasets is lacking. This question aims to validate their performance in real-world scenarios, where open-source datasets often contain noise and incomplete data.

In this RQ, we evaluate Eadro [14], CloudRCA [10], Diagfusion [7], RMLAD [9], MicroCBR [12], ART [17], Medicine [18], UniDiag [19], and PDiagnose [8] on the AIOps21, AIOps22, GAIA, and MicroServo datasets to assess their performance and applicability under diverse conditions. Due to Nezha's [13] specific requirements for multi-source data, it cannot be evaluated on the four selected multi-source datasets. Thus, we only replicate its results on its original open-source datasets: Online-Boutique and Train-ticket (TT2). For TrinityRCL [11], we conduct experiments only on the AIOps22 dataset, as it relies on KPI data for anomaly score calculation, which is only available in this dataset.

RQ2: Does the use of multi-source data enhance performance? It is commonly believed that integrating multi-source data enriches the information set, leading to more accurate diagnostics. However, this assumption lacks empirical validation. This question explores the actual impact of multi-source data on failure diagnosis, providing experimental evidence to support or challenge this belief.

In this RQ, we perform multi-source data ablation experiments on Eadro [14], CloudRCA [10], Diagfusion [7], ART [17], Medicine [18], UniDiag [19], and PDiagnose [8] across the AIOps21, AIOps22, GAIA, and MicroServo datasets. Ablation is not feasible for TrinityRCL [11] and RMLAD [9] due to their reliance on interactions between data sources. MicroCBR [12] is also excluded, as we do not employ all three data types required for complete failure fingerprinting.

RQ3: What is the time efficiency of the selected methods? Time efficiency is crucial for real-time applications.

This question evaluates whether the methods can meet the computational demands of real-time systems and operational speed requirements.

In this RQ, we evaluate the time performance of Eadro [14], CloudRCA [10], Diagfusion [7], RMLAD [9], MicroCBR [12], ART [17], Medicine [18], UniDiag [19], and PDiagnose [8] on the MicroServo dataset. For each method, we measure the total runtime per failure instance, including raw data preprocessing. For deep learning methods, we also report the average training time on fixed training and validation sets. Nezha [13] and TrinityRCL [11] are excluded due to incompatibility with this dataset.

RQ4: How effectively do the selected methods diagnose different types of failures? Assessing the versatility of diagnostic methods under various failure conditions is essential to ensure their broad applicability across different operational environments.

In this RQ, we evaluate the diagnostic performance across different failure types using the MicroServo dataset, where each type of failure is evenly distributed. To ensure fairness, we maintain equal class distributions in both training and testing sets. We conduct experiments on Eadro [14], CloudRCA [10], Diagfusion [7], RMLAD [9], MicroCBR [12], ART [17], Medicine [18], UniDiag [19], and PDiagnose [8]. Nezha [13] and TrinityRCL [11] are excluded due to dataset incompatibility.

RQ5: How do the selected deep learning methods perform under varying training volumes? Understanding how training volume affects performance is crucial for optimizing strategies, ensuring methods remain effective with limited data, which is common in practical scenarios.

In this RQ, we evaluate model performance under varying training data volumes on CloudRCA [10], Diagfusion [7], Eadro [14], ART [17], Medicine [18], UniDiag [19], and RMLAD [9]. To ensure fairness, we use a consistent test set across all training scenarios, enabling uniform and equitable comparison.

C. Evaluation Metrics

Our study on failure diagnosis in microservice systems categorizes the task into two main areas: RCL and FC.

For RCL, we use the Accuracy@k (A@k) metric, which measures the likelihood that the true root cause is within the top-k predicted causes. This metric indicates better diagnostic performance with higher values, with k set at 1, 3, and 5 to reflect real-world diagnostic needs. It is defined as:

$$\text{A@k} = \frac{1}{A}\sum_{a\in A}\begin{cases} 1, & \text{if } V_a \in R_a[k] \\ 0, & \text{otherwise} \end{cases}$$

where A is the set of system failures, $a$is one failure in A; V_a is the real root cause of a; $R_a[k]$is the predicted top-k set of a; FC employs metrics from machine learning to assess effectiveness, focusing on precision (Pre), recall (Rec), and F1-score (F1). These metrics help understand the method's accuracy in predicting different types of failures. Given true positives (TP), false positives (FP), and false negatives (FN), they are defined as follows: Pre $= \frac{TP}{TP+FP}$, Rec $= \frac{TP}{TP+FN}$, F1 $= \frac{2\times\text{Pre}\times\text{Rec}}{(\text{Pre}+\text{Rec})}$.

D. Experimental Setup

In the RCL task, we configure Eadro [14], Diagfusion [7], ART [17], Medicine [18], UniDiag [19], and PDiagnose [8] to operate at the instance-level, while RMALD [9] and TrinityRCL [11] are configured at the service-level. Since Nezha [13] requires the incorporation of trace IDs into log data, and the public datasets used in our experiments do not meet this requirement, we are only able to replicate the experiment using the dataset from the original paper. In addition, TrinityRCL [11] requires KPI data. Our experiments maintain a training and testing set ratio of 7:3. To reduce the impact of randomness, each experiment is repeated five times, and we report the average results. We conduct our experiments on a Linux Server 20.04.1 LTS with two 48C48T Intel(R) Xeon(R) CPU E5-2650 v4@ 2.20GHz, one NVIDIA(R) Tesla(R) M4, and 125 GB RAM.

We build a new publicly usable multi-source dataset from a newly released microservice benchmark, MicroServo [20]. It deploys the open-source Online-Boutique [22] provided by GoogleCloudPlatform, uses three collectors to gather metrics, logs, and trace data, and applies chaos engineering techniques [25] to simulate real failures. In practice, MicroServo [20] uses Prometheus [26] as the metric collector, Filebeat [27] as the log collector, Elastic APM [28] as the trace collector, and ChaosMesh [29] to inject failures into the microservice system.

We orchestrate seven types of failure injections, covering most failure scenarios, into MicroServo [20], collecting data over a three-day period. This dataset comprises metric data collected at 1-second intervals, a diverse range of logs, and trace records, providing comprehensive system observability. Compared to other open-source multi-source datasets, this dataset is free from data loss or timestamp discrepancy issues. More importantly, its ground truth is completely accurate, precisely recording failure instances, start times, and end times. This relieves researchers from the tedious task of data validation. Additionally, it features a balanced injection of hundreds of failures across different types. This ensures that deep learning methods receive fair training across each type of failure.

E. Methodology Analysis

In this chapter, we analyze methodologies for microservice failure diagnosis. We cover a range of methodologies from statistical techniques to advanced machine learning methods, tailored for modern microservice architectures.

1) Eadro: To address task disconnection, Eadro [14] uses all three types of multi-source data for RCL by integrating anomaly detection and root cause localization. It classifies failure cases and treats failure-free cases as a separate class. To better utilize monitoring data, it applies three specialized fusion methods and combines them via a graph attention network. Eadro [14] builds the system structure from trace data

and achieves convergence through dependency graph learning. To ensure fair dataset distribution, failure cases are first split into training and testing sets before sample generation, without degrading performance.

2) Nezha: Nezha [13] tackles RCL using all three types of multi-source data. It builds event graphs via statistical methods to identify failures within pattern graphs and locate root causes. To address the loss of context in isolated log analysis, Nezha [13] integrates trace IDs into logs and aligns metric data using timestamps, enhancing multi-modal fusion. It defines two pattern types: expected patterns, which occur frequently during failure-free periods but rarely during failures, and actual patterns, which exhibit the opposite trend. By correlating these patterns, Nezha [13] identifies root cause candidates, addressing the challenge of poor interpretability. It further achieves inner-service-level RCL through failure event patterns, overcoming the limitation of coarse-grained localization seen in prior methods.

3) CloudRCA: CloudRCA [10] addresses FC using metric data, log data, and module dependencies from the Configuration Management Database (CMDB). To handle challenges such as manual troubleshooting, real-time requirements, limited training samples, and cross-platform transferability, it applies time-series anomaly detection and log clustering to generate a unified feature matrix. This matrix, combined with module dependencies, feeds into a Knowledge-Informed Hierarchical Bayesian Network (KHBN) for real-time failure inference. Data fusion relies on time windows: upon an alert, metric and log data within the failure window are aligned and input to the KHBN.

4) Diagfusion: Diagfusion [7] uses all three types of multi-source data for FC and RCL, incorporating CMDB data for propagation paths and characteristics. It extracts, serializes, and converts events into vectors, then builds a dependency graph for GNN training using traces and deployment data. A neural network encodes events to learn their representation. Finally, Diagfusion [7] uses a GNN to merge event representations and the system dependency graph for FC and RCL.

5) RMLAD: RMLAD [9] identifies root cause metrics by correlating them with log anomaly scores. It uses DeepLog [30] to detect anomalies in log templates and intervals, then computes scores via polynomial functions. Mutual Information (MI) quantifies the association between each metric and anomaly scores to rank candidates. We modify the method to map metrics to their services for consistent service-level RCL evaluation.

6) TrinityRCL: TrinityRCL [11] uses all three types of multi-source data for RCL at application, service, host, and metric levels. It builds a graph from trace, host, and log data, and merges them with telemetry from 30 minutes before anomalies. Anomaly scores are converted into transfer probabilities and processed with Random Walk with Restart (RWR). Results are ranked by node type and visit count to address multi-level RCL.

7) MicroCBR: MicroCBR [12] uses all three types of multi-source data for FC, with command data to address data loss. It employs failure fingerprints and spatio-temporal knowledge graphs for data fusion, solving challenges of single-sample failure and spatial-temporal diagnosis. Failure fingerprints are extracted from four data sources and integrated with the system topology. MicroCBR [12] performs anomaly detection, constructs failure fingerprints from anomalies, and assigns weights based on frequency and correlation. Its effectiveness depends on the accuracy of failure fingerprints, which vary across datasets.

8) PDiagnose: PDiagnose [8] uses all three types of multi-source data for RCL. It performs anomaly detection on each data type and builds an anomaly microservice queue. For metric data, it uses Kernel Density Estimation (KDE) and Weighted Moving Average (WMA). For call chain data, it structures relationships as $\langle Req, Caller, Callee, Duration \rangle$ and flags traces as anomalous if Duration exceeds a threshold. For log data, suspicious fields are flagged as anomalies. A voting system identifies the root cause.

9) ART: ART [17] unifies all three types of multi-source data into time-series formats and employs self-supervised learning to compute reconstruction errors for FC and RCL tasks. It uses a Transformer encoder to capture channel dependencies, GRU to capture temporal dependencies, and GraphSAGE to capture call dependencies, thereby modeling various system dependencies and predicting future system states to support downstream tasks.

10) Medicine: Medicine [18] uses all three types of multi-source data for FC, employing dedicated encoders to capture complementary information and alleviate performance drops caused by missing or low-quality data. It evaluates data sources to distinguish high- and low-yield data sources, applying gradient suppression to the former and feature enhancement to the latter, dynamically adjusting training to reduce the impact of convergence inconsistency and cross data source interference.

11) UniDiag: UniDiag [19] uses temporal knowledge graphs (TKGs) to integrate all three types of multi-source data for FC. It models structural and temporal dependencies via microservice-oriented graph embedding (MOGE). Offline, it clusters embeddings to identify failure patterns, requiring only cluster centers for labeling. Online, anomalies are matched to existing clusters or used to create new ones [31].

F. Results And Findings

RQ1: How do the selected methods perform on open-source datasets? The RCL experimental results are presented in Table IV, FC Experimental Results are presented in Table VII, Nezha [13] and RMLAD [9] results are presented in Table V and Table VI, TrinityRCL's [11] results on the AIOps22 dataset are 6.30% @1, 27.8% @3, and 50.70% @5.

Only Medicine [18] consistently performs well across all datasets, demonstrating strong effectiveness and robustness. Other methods exhibit varied performance patterns depending on dataset characteristics. As microservice count and interaction complexity increase, the size and intricacy of the generated graphs grow accordingly. This exacerbates the challenges

TABLE IV: A@k (%) of different methods on different datasets in instance-level root cause localization

Dataset	Experiment Type	Eadro [14]			Diagfusion [7]			PDiagnose [8]			ART [17]		
		@1	@3	@5	@1	@3	@5	@1	@3	@5	@1	@3	@5
AIOps21	Normal	**16.52**	**46.39**	**64.04**	47.83	78.26	100.00	27.85	56.96	73.42	8.13	20.33	35.00
	w/o Trace	17.20	32.26	49.68	56.52	86.96	100.00	**34.18**	**58.23**	**72.15**	8.13	20.33	32.25
	w/o Metric	14.04	37.05	47.12	34.78	65.22	100.00	5.06	31.65	48.10	0	0	0
	w/o Log	13.03	44.41	64.58	52.17	82.61	100.00	27.85	56.96	73.42	**12.24**	**27.85**	**47.78**
	Log Only	0	0	0	34.78	78.26	86.96	2.53	25.32	39.24	0	0	0
	Metric Only	16.13	41.29	54.62	**69.57**	**95.65**	**95.65**	**34.18**	**58.23**	**72.15**	10.01	24.23	30.17
	Trace Only	14.97	31.23	45.05	52.17	78.26	95.65	5.06	31.65	48.10	0	0	0
AIOps22	Normal	19.04	37.29	53.01	50.00	81.40	88.37	**17.78**	**51.11**	**57.78**	11.12	20.33	25.00
	w/o Trace	12.42	32.57	47.84	34.88	56.98	76.74	**17.78**	**51.11**	**57.78**	10.01	18.88	23.49
	w/o Metric	0	0	0	47.67	70.93	79.07	0	17.78	33.33	0	0	18.88
	w/o Log	**19.36**	**37.96**	**55.99**	**56.98**	**93.02**	**97.67**	**17.78**	**51.11**	**57.78**	**15.67**	**26.64**	**34.27**
	Log Only	0	0	0	9.30	30.23	53.49	0	17.78	33.33	12.64	21.18	26.36
	Metric Only	12.64	31.99	49.88	38.37	67.44	70.93	**17.78**	**51.11**	**57.78**	10.25	16.67	26.38
	Trace Only	0	0	0	39.53	76.74	87.21	0	17.78	33.33	0	0	0
GAIA	Normal	**30.27**	**56.48**	**61.17**	60.70	89.56	100.00	**29.70**	**54.46**	**67.33**	14.37	25.12	30.08
	w/o Trace	27.38	42.79	49.43	58.89	85.41	100.00	**29.70**	**54.46**	**67.33**	12.10	23.33	27.78
	w/o Metric	0	0	0	58.57	82.64	100.00	6.93	18.81	52.48	0	12.10	14.37
	w/o Log	25.90	42.60	56.37	55.27	88.60	100.00	27.72	53.57	69.31	0	0	0
	Log Only	0	0	0	57.93	78.49	100.00	6.93	18.81	52.48	**26.67**	**27.18**	**35.00**
	Metric Only	22.08	36.12	46.17	**60.28**	**92.44**	**100.00**	27.72	53.47	69.31	12.10	21.11	29.70
	Trace Only	0	0	0	51.86	77.21	100.00	6.93	16.83	47.52	0	0	0
MicroServo	Normal	18.68	24.92	30.46	**43.75**	**65.63**	**81.25**	24.05	56.96	59.49	9.11	13.17	28.00
	w/o Trace	11.48	18.73	15.64	37.50	71.88	87.50	**24.05**	**58.23**	**60.76**	0	0	0
	w/o Metric	0	0	0	37.50	65.63	75.00	11.39	29.11	77.22	0	0	0
	w/o Log	**19.81**	**27.62**	**35.11**	37.50	68.75	78.13	24.05	56.96	59.49	**9.81**	**18.63**	**45.00**
	Log Only	0	0	0	6.25	62.50	71.88	6.33	25.32	54.43	0	0	0
	Metric Only	14.65	21.59	28.41	34.38	78.13	87.50	**24.05**	**58.23**	**60.76**	0	3.11	21.67
	Trace Only	0	0	0	31.25	65.63	75.00	11.39	30.38	77.22	0	3.62	23.33

Bold *indicates the best score of a method on the current dataset, while* underline *highlights the best overall score among all methods for that dataset.*

TABLE V: A@k(%) of Nezha [13] in root cause localization

Dataset	Level	@1	@3	@5
Online-Boutique	service-level	92.86	96.43	96.43
	inner-service-level	92.86	96.43	96.43
Train-ticket	service-level	86.67	97.78	97.78
	inner-service-level	86.67	97.78	97.78

TABLE VI: A@k(%) of RMLAD [9] in service-level root cause localization

Dataset	@1	@3	@5
AIOps22	26.09	47.20	96.43
AIOps21	20.09	42.86	63.74
Gaia	23.19	46.38	65.22
MicroServo	12.31	29.69	47.75

in graph construction, traversal, and updating, leading to performance degradation in graph-based methods like Eadro [14], ART [17], UniDiag [19], and Diagfusion [7], which perform best on GAIA and worst on AIOps21.

Nezha [13] replicates its original results successfully. CloudRCA [10] suffers when its anomaly detection module fails to capture patterns, particularly when wavelet decomposition is unsuitable due to lack of clear seasonality. RMLAD [9], focusing solely on metrics rather than services, struggles to generalize when log-indicated services are unclear or collection intervals vary. TrinityRCL [11] relies on precisely labeled entry-point anomalies, limiting its localization capability on datasets like AIOps22 where such labels are missing. MicroCBR [12] may misclassify when failure fingerprints are ambiguous, though it performs well under ideal conditions. PDiagnose [8] exhibits unstable performance due to its dependence on expert-driven parameter tuning, which varies across datasets.

Finding1: The performance of various methods across different datasets is inconsistent, indicating a lack of broad applicability. Their effectiveness largely depends on the appropriateness of data fusion techniques and the quality of multi-source datasets.

RQ2: Does the use of multi-source data enhance performance? The ablation experiment results for the RCL method are presented in Table IV. The ablation experiment results for the FC method are presented in Table VII

Ablation studies conducted across multiple datasets (AIOps21, AIOps22, MicroServo, and GAIA) reveal that the effectiveness of multi-source data integration is highly context-dependent. For example, Eadro [14] exhibits strong performance on the AIOps21 dataset in the *Normal* setting, highlighting the advantages of leveraging multi-source data. However, its performance significantly deteriorates in *Log*

TABLE VII: Precision(%), Recall(%), F1-Score(%) of different methods on different datasets in failure classification

Dataset	Experiment Type	Diagfusion [7]			CloudRCA [10]			MicroCBR [12]			ART [17]			Medicine [18]			UniDiag [19]		
		Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1
AIOps21	Normal	26.67	22.50	24.41	**35.14**	**27.83**	**24.56**	35.65	40.00	37.56	11.67	9.28	10.33	**67.69**	**62.50**	**63.30**	20.21	18.50	19.31
	w/o Trace	25.00	20.38	22.45	/	/	/	/	/	/	10.01	8.33	9.09	40.76	43.33	38.40	18.88	15.85	17.23
	w/o Metric	15.00	11.25	12.86	6.59	14.91	9.14	/	/	/	0.56	4.17	0.98	17.02	19.24	18.00	10.11	8.50	9.24
	w/o Log	**26.67**	**23.33**	**24.89**	22.82	26.21	23.78	/	/	/	**12.33**	**15.00**	**13.53**	51.75	45.27	41.54	**22.50**	**21.33**	**21.90**
	Log Only	18.33	12.22	14.67	/	/	/	/	/	/	0	0	0	36.38	46.67	38.84	4.17	5.22	4.64
	Metric Only	23.33	19.05	20.97	/	/	/	/	/	/	9.33	7.11	8.07	60.90	69.40	63.28	19.88	16.67	18.14
	Trace Only	11.67	9.33	10.37	/	/	/	/	/	/	0	0	0	18.89	16.33	16.87	9.24	8.11	8.64
AIOps22	Normal	42.88	39.88	41.13	14.76	13.62	8.74	39.05	46.67	40.00	18.87	16.67	17.70	**90.05**	**82.14**	**80.40**	38.13	35.33	36.64
	w/o Trace	38.26	36.20	35.84	/	/	/	/	/	/	19.98	17.12	18.45	49.27	42.86	44.36	39.33	36.11	37.62
	w/o Metric	30.07	30.12	29.81	8.01	9.28	7.70	/	/	/	4.00	20.00	6.67	20.83	25.33	21.19	20.09	18.88	19.45
	w/o Log	**53.15**	**49.39**	**49.18**	**15.60**	**14.35**	**13.49**	/	/	/	**21.07**	**18.87**	**19.89**	75.33	85.74	77.57	**42.67**	**38.34**	**40.36**
	Log Only	29.33	33.57	29.67	/	/	/	/	/	/	5.12	15.25	7.66	33.95	33.50	32.35	23.34	20.07	21.56
	Metric Only	34.88	36.09	35.09	/	/	/	/	/	/	20.00	17.33	18.57	56.92	53.75	54.83	37.34	36.09	36.70
	Trace Only	32.62	35.34	31.21	/	/	/	/	/	/	0	0	0	35.71	37.24	35.13	34.67	31.08	32.78
GAIA	Normal	44.10	51.11	46.30	11.25	33.33	15.48	86.62	80.65	83.53	**21.33**	**28.77**	**24.51**	**84.87**	**78.85**	**81.41**	41.33	45.37	43.26
	w/o Trace	44.85	44.47	44.62	/	/	/	/	/	/	18.67	25.56	21.54	59.05	62.30	60.58	43.47	45.11	44.34
	w/o Metric	**53.75**	**52.44**	**52.98**	**55.48**	**58.33**	**55.34**	/	/	/	10.11	13.33	11.50	61.17	60.92	60.90	**56.61**	**53.21**	**54.86**
	w/o Log	28.61	29.84	27.53	8.85	25.00	12.50	/	/	/	0	0	0	44.51	47.37	44.80	36.67	38.67	37.64
	Log Only	53.10	53.21	52.65	/	/	/	/	/	/	20.25	27.12	23.23	58.25	61.45	59.11	53.75	50.11	52.00
	Metric Only	26.29	26.65	25.56	/	/	/	/	/	/	16.29	23.10	19.14	40.64	41.52	41.06	20.10	22.65	21.32
	Trace Only	32.51	29.66	29.69	/	/	/	/	/	/	0	0	0	27.90	24.78	25.53	34.62	35.57	35.14
MicroServo	Normal	28.81	24.34	25.29	**31.51**	**28.23**	**24.72**	49.85	59.49	51.70	15.56	13.33	14.37	**69.10**	**80.00**	**69.94**	**30.08**	**28.41**	**29.22**
	w/o Trace	23.81	11.14	14.29	/	/	/	/	/	/	13.11	10.01	11.35	58.38	62.22	57.61	25.57	22.11	23.66
	w/o Metric	2.86	33.90	0.14	24.41	18.17	15.62	/	/	/	0.33	10.25	0.64	29.38	37.05	31.99	3.01	30.34	5.46
	w/o Log	26.67	19.69	21.79	31.76	28.08	24.71	/	/	/	**25.57**	**26.31**	**25.86**	58.86	65.27	56.46	28.67	26.21	27.42
	Log Only	**31.43**	**42.30**	**36.32**	/	/	/	/	/	/	0	0	0	42.61	41.67	40.70	20.09	19.47	19.76
	Metric Only	13.57	10.32	11.39	/	/	/	/	/	/	15.56	11.39	13.12	64.88	66.67	65.17	25.57	24.46	25.02
	Trace Only	26.19	19.47	20.03	/	/	/	/	/	/	0	0	0	12.33	17.33	13.49	26.19	24.32	25.22

Bold *indicates the best score of a method on the current dataset, while* *underline* *highlights the best overall score among all methods for that dataset.*

Only and other single data source settings, suggesting that Eadro [14] is highly reliant on comprehensive data integration. Similarly, CloudRCA [10] and Diagfusion [7] sometimes yield better results when certain data source are excluded, indicating that fusing heterogeneous data sources does not universally improve diagnostic accuracy.

This trend is particularly evident in the AIOps22 dataset. In the *w/o Log* setting, Eadro [14], CloudRCA [10], and Diagfusion [7] all outperform their respective results in the *Normal* setting. In contrast, PDiagnose [8] produces identical results across both settings. Upon further inspection of the log data, we find that no actual failure-related information is present in the logs for this dataset. As a result, large volumes of large volumes of irrelevant logs introduces noise that misleads most models. In contrast, PDiagnose [8], which exclusively utilizes anomalous log data, avoids this issue and maintains stable performance.

To further investigate, we conduct additional experiments on Diagfusion [7] using the AIOps22 dataset, supplying only the relevant data sources for each failure case. Specifically, we remove all log data and selectively prune trace data for certain cases. With this simplified data provisioning strategy, Diagfusion [7] achieves an approximately 10% improvement in @1 for RCL and a nearly 15% gain in F1 score for FC. Additionally, Medicine [18] achieves the best results under the *Normal* setting across all datasets, fully demonstrating the effectiveness and robustness of its data fusion strategy.

These findings suggest that while multi-source data holds significant potential to enhance failure diagnosis, its actual utility hinges on well-designed data preprocessing and fusion strategies tailored to the characteristics of the dataset and the task. Blindly integrating multi-source data does not necessarily lead to improved outcomes in failure diagnosis.

Finding2: The effectiveness of multi-source data integration relies on aligning data sources with methodological capabilities. Dynamically adjusting data source utilization based on their performance can enhance method accuracy.

RQ3: What is the time efficiency of the selected methods? The results of the time performance experiment are presented in Table VIII.

TABLE VIII: Efficiency of different methods

Method	Training Time		Runtime
	Per Epoch	Total	
Eadro [14]	33.83s	2266.45s	31.05s
Diagfusion [7]	14.18s	1300.17s	13.96s
CloudRCA [10]	**1.82s**	**1.82s**	0.31s
RMLAD [9]	/	/	1.19s
MicroCBR [12]	/	/	**0.05s**
PDiagnose [8]	/	/	80.21s
ART [17]	24.17	1208.40	1.25s
Medicine [18]	2.55	255.47	0.17s
UniDiag [19]	25.52	2041.53	0.32s

The results reveal significant disparities. Due to the utilization of Graph Neural Networks (GNNs), Eadro [14], UniDiag [19], and Diagfusion [7] requires considerably more per epoch and total training time, than several tens of times or even hundreds more compared to CloudRCA [10], presenting

a substantial overhead that may hinder practical deployment. In contrast, PDiagnose [8] suffers from extended runtime due to computationally intensive processes like Kernel Density Estimation (KDE) and Weighted Moving Average (WMA). On the other hand, MicroCBR [12] boasts the shortest runtime at just 0.12 seconds, demonstrating exceptional efficiency. CloudRCA [10] shows relatively low training and runtime durations, indicating a balanced method between performance and efficiency.

Finding3: Eadro [14], ART [17], UniDiag [19], and DiagFusion [7] suffer from prolonged training time. Most of the methods exhibit runtimes of less than 30 seconds, with notable exceptions being PDiagnose [8] and Eadro [14].

RQ4: How effectively do the selected methods diagnose different types of failures? The different failure types experiment results are presented in Fig. 3.

The diagnostic capabilities of different methods across datasets exhibit variability, particularly in their ability to accurately identify specific types of failures. MicroCBR [12] struggles with diagnosing *Request Delay*, *Request Absence*, and *Network Loss failures*, resulting in zero diagnostic outcomes due to its dependency on failure fingerprints. Similarly, Diagfusion [7], and Medicine [18] face significant challenges in distinguishing between failures in the MicroServo dataset, which lacks network-related metrics and has low failure distinguishability, preventing it from effectively learning the characteristics of each failure type, thus significantly reducing the FC effectiveness.

Finding4: Existing methods exhibit inconsistent performance across different failure types. Future methods should be optimized for a wide range of failure types to ensure reliable performance in diverse failure cases.

RQ5: How do the selected deep learning methods perform under varying training volumes?

The different data volumes experiment results are presented in Table IX and Table X.

In the realm of RCL, RMLAD [9], ART [17], and Diagfusion [7] achieve their optimal performance with full training data. Conversely, Eadro [14] peaks at 80% training volume. For FC, all methods reach their best performance with relatively low training volumes; notably, CloudRCA [10] peaks at just 20% training volume. However, post-peak, performance trends for both methods demonstrate a decline followed by a resurgence, indicating a non-linear response to increasing training data.

This discrepancy in training data requirements suggests that RCL tasks are inherently more data-intensive than FC. One plausible explanation lies in the higher level of feature redundancy and noise involved in root cause localization. Since RCL models must evaluate a broader range of candidate components or services, they inevitably process large volumes of non-informative or misleading signals. Without sufficient training data, the model is less capable of learning effective

TABLE IX: A@k (%) of Eadro [14], RMLAD [9], Diagfusion [7], and ART [17] with varying training data size (compared to the whole training dataset) in root cause localization

Method	Training Data Size	@1	@3	@5
Eadro [14]	5/5	38.90	52.45	56.43
	4/5	**43.52**	**60.07**	**64.70**
	3/5	26.70	32.19	33.52
	2/5	24.54	41.23	48.41
	1/5	22.43	36.55	39.56
RMLAD [9]	**5/5**	**11.44**	**27.03**	**47.27**
	4/5	10.32	25.26	45.56
	3/5	11.25	25.53	47.73
	2/5	10.01	26.19	43.14
	1/5	10.21	24.51	46.61
Diagfusion [7]	**5/5**	**43.75**	**65.63**	**84.38**
	4/5	28.13	68.75	81.25
	3/5	21.88	43.75	65.63
	2/5	31.25	46.88	75.00
	1/5	25.00	46.86	62.50
ART [17]	**5/5**	**9.11**	**13.17**	**28.00**
	4/5	7.13	9.33	24.54
	3/5	3.11	6.18	13.88
	2/5	0	1.28	8.67
	1/5	0	0	0

TABLE X: Precisions (%), Recalls (%), F1-Scores (%) of CloudRCA [10], Diagfusion [7], ART [17], Medicine [18], and UniDiag [19] with varying training data size (compared to the whole training dataset) in failure classification.

Method	Training Data Size	Pre	Rec	F1
CloudRCA [10]	5/5	31.76	28.08	24.71
	4/5	28.75	27.32	24.53
	3/5	29.97	30.96	26.61
	2/5	45.25	38.96	36.38
	1/5	**50.70**	**54.37**	**49.38**
Diagfusion [7]	5/5	25.71	29.17	23.27
	4/5	24.76	34.48	25.28
	3/5	28.10	29.64	25.73
	2/5	**30.00**	**27.38**	**26.84**
	1/5	13.33	8.16	9.47
ART [17]	5/5	15.56	13.33	14.37
	4/5	8.15	9.85	8.90
	3/5	17.48	14.56	15.91
	2/5	**19.98**	**15.15**	**17.18**
	1/5	10.12	13.33	11.47
Medicine [18]	5/5	69.10	80.00	69.94
	4/5	71.29	82.50	74.72
	3/5	61.10	73.33	63.26
	2/5	**77.14**	**85.00**	**80.91**
	1/5	32.00	28.33	27.91
UniDiag [19]	5/5	30.08	28.41	29.22
	4/5	26.76	27.18	26.96
	3/5	24.10	26.44	25.18
	2/5	30.11	27.38	28.70
	1/5	**33.33**	**31.67**	**32.48**

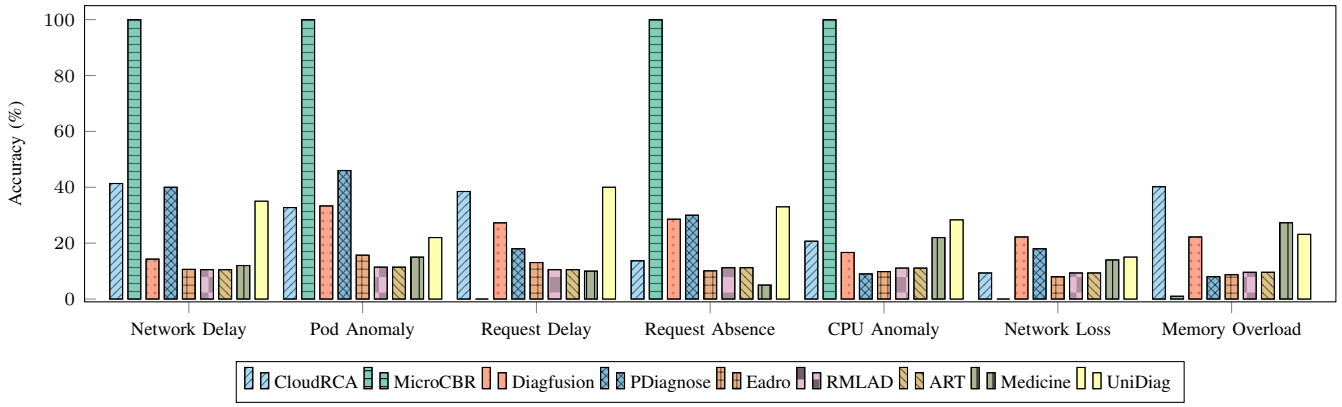

Fig. 3: Accuracy (%) of different methods with different failure types

filtering mechanisms, which impairs its ability to distinguish between relevant and irrelevant information.

Another critical factor is the strong dependence of RCL on contextual and structural system information. Unlike FC, which often relies on globally discriminative patterns linked to failure types, RCL requires the model to capture fine-grained interactions across logs, metrics, and traces. Modeling such complex interdependencies demands a richer training set to support generalization across diverse and often subtle failure scenarios.

Finding5: More training data does not necessarily lead to better performance for RCL and FC. Notably, FC often achieves optimal results with less data, underscoring the need for adaptive learning systems that can dynamically determine the optimal training size.

G. Summary:

Based on our experimental findings and analysis, it is clear that, except for Medicine [18], no single method excels across every dataset. Medicine [18] exhibit outstanding effectiveness and robustness, other methods typically exhibit different limitations. Importantly, the results of these methods on synthetic datasets may not truly represent how they perform in actual real-world systems. It is crucial to make assumptions about the working environment based on prior knowledge and select the algorithm most suitable for the current scenario. Based on the experimental results, we provide several high-reliability recommendations for SREs:

For the FC task: 1) Medicine [18] employs a dedicated encoding mechanism for each data source and integrates an adaptive optimization module to adjust the convergence speed across multi-source data. This design effectively captures the unique characteristics of individual data sources and their complementary information, mitigating the performance degradation commonly caused by incomplete or low-quality multi-source data. These strengths make Medicine [18] particularly well-suited for real-world scenarios, where data heterogeneity, noise, and missing data sources are common challenges. 2) In scenarios requiring both high processing speed and diagnostic effectiveness, MicroCBR [12] is a suitable choice, provided that detailed representations of failure types are available. It offers the shortest runtime among all methods, and, with high-quality failure fingerprints, it can achieve outstanding diagnostic performance. 3) When using deep learning methods, avoid excessive training data, as models may actually perform better with a smaller amount of training data.

For the RCL task: 1) Diagfusion [7] performs well in most scenarios. More importantly, it incorporates data augmentation during training, effectively mitigating the issue of data imbalance across different failure types. This feature makes it especially suitable for real-world applications, where the distribution of failure types is often highly skewed, some failures occur much less frequently than others. In microservice systems, such imbalances are further exacerbated by failure-tolerance mechanisms that mask or recover from certain failures, making failure-related data even more challenging to capture. 2) When using deep learning methods, more training data generally leads to better results.

Furthermore, since the optimal amount of training data varies significantly between the FC and RCL tasks, we do not recommend adopting a multi-task learning approach that trains a single model to handle both tasks simultaneously. This discrepancy may lead to suboptimal convergence and degraded performance in one or both tasks. Therefore, training separate models tailored to the specific data and requirements of each task is generally a more effective and reliable strategy.

For dataset: 1) If the dataset does not include complete trace data, mathematical statistical methods that do not rely on this data modality may significantly outperform learning methods that depend on trace data as part of their structure. 2) For methods utilizing logs, the contribution of logs within multimodal data heavily depends on the context. It is essential to carefully assess whether the dataset provides high-quality data before deciding whether to use models that rely on log data. 3) The issue of misleading results from multi-source data cannot be overlooked; blindly using multi-source data is likely to degrade performance. To avoid misleading outcomes, we recommend selecting methods that focus solely on failure-relevant data source features. 4) If you are working with the AIOps21 or AIOps22 datasets, we recommend omitting the log data. These logs contain very little information useful for

failure diagnosis, so most methods actually perform better on these two datasets when the log data is removed.

V. THREATS TO VALIDITY

During our study, we identify the following major threats to the validity:

Limited methods: We evaluate only eight microservice failure diagnosis methods based on multi-source data. This limited selection may not fully represent the range of methods that could be applied to this domain. In the future, we plan to expand our evaluations to include a wider variety of methods to better assess the domain's capabilities.

Reimplementation: Among the eleven failure diagnosis methods we evaluate, only seven have publicly available code. We replicate the remaining methods based on their respective papers' descriptions, which may introduce discrepancies due to potential undocumented details. We will engage with the original authors to clarify ambiguous details and mitigate these risks in future replications.

VI. RELATED WORK

Failure diagnosis. Previous studies of failure diagnosis mostly rely on single source of data such as logs, metrics, and traces [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46]. Among them, Amar et al. [32], He et al. [33], Ikeuchi et al. [34], Jia et al. [35] and Li et al. [36] utilize log data; Lin et al. [37], FluxRank [38], FacGraph [39], MS-Rank [40] and CauseInfer [41] employs metric data; Zhou et al. [42], Marwede et al. [43], Mi et al. [44], Li et al. [45] and Pham et al. [46] are based on trace data. To overcome the limitations of a single source of data, researchers have proposed numerous methods based on multi-source data to achieve more effective failure diagnosis [9], [10], [8], [11], [12], [13], [14], [7], [19], [18], [17].

Empirical study in the failure management domain. In recent years, there have been numerous empirical studies focusing on failure management based on a single source of data [47], [48], [49], [50], [51], [52]. Among them, Li et al. [47] conduct an empirical study on anomaly detection methods using multivariate time series (MTS) data, providing tailored model selection suggestions based on typical data characteristics and anomaly types. Zhao et al. [50] propose a generic log anomaly detection system named LogAD, which outperforms all baselines and represents the first in-depth study of log anomaly detection in real-world settings. He et al. [52] conduct a comprehensive survey study on log analysis at Microsoft and uncovers the real needs of industrial practitioners and the unnoticed yet significant gap between industry and academia. Fu et al. [48] explore the impact of log parsers on log-based anomaly detection methods. Their experiments demonstrate that all anomaly detection methods perform more effectively and efficiently when using heuristic-based log parsers. Le et al. [49] analyze five state-of-the-art deep learning models for log-based anomaly detection and found that factors like training data selection and data noise significantly impact performance, indicating that the problem remains unsolved. Chuah et al. [51] propose a method for diagnosing major page faults and evaluate the LASSO, Ridge, and Elastic Net regression methods on real resource use data and system logs.

VII. CONCLUSION

This empirical study provides a comprehensive analysis and evaluation of eleven methods designed for microservice failure diagnosis using multi-source data. Initially, we conduct an in-depth evaluation of existing open-source multi-source datasets, uncovering a significant gap in the availability of usable datasets. As a result, we collect and construct a new multi-source dataset, MicroServo, which addresses this limitation. Subsequently, we carry out extensive experiments on the eleven selected methods across three open-source datasets as well as the newly developed MicroServo dataset.

Our findings lead to two pivotal conclusions regarding the use of multi-source data in failure diagnosis tasks:

1) The integration of multi-source data does not always yield positive outcomes.
2) For deep learning-based methods, a small amount of training data is sufficient to achieve optimal performance on the FC task, whereas the RCL task requires significantly more training data to reach comparable performance levels.

Moreover, we summarize the experimental results and provide recommendations for method selection tailored to different real-world application scenarios from three key perspectives: RCL task, FC task, and dataset characteristics.

In conclusion, our work represents a pioneering empirical study in this field, shedding light on both the potential and limitations of multi-source data for failure diagnosis. The critical insights gained from this research will serve as a valuable foundation for future studies and innovations in this area, providing guidance for the development of more effective and robust methods. We hope that these findings will stimulate further exploration and advancements in applying multi-source data to complex, real-world diagnostic challenges.

REFERENCES

[1] J. Lewis and M. Fowler. Microservices a definition of this newarchitectural term, 2014. [Online].Available:http://martinfowler.com/articles/microservices.html.

[2] Hao Zhou, Ming Chen, Qian Lin, Yong Wang, Xiaobin She, Sifan Liu, Rui Gu, Beng Chin Ooi, and Junfeng Yang. Overload control for scaling wechat microservices. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 149–161, 2018.

[3] Dewei Liu, Chuan He, Xin Peng, Fan Lin, Chenxi Zhang, Shengfang Gong, Ziang Li, Jiayu Ou, and Zheshun Wu. Microhecl: High-efficient root cause localization in large-scale microservice systems. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 338–347. IEEE, 2021.

[4] Li Wu, Johan Tordsson, Erik Elmroth, and Odej Kao. Microrca: Root cause localization of performance issues in microservices. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2020.

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

[6] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 102–111, 2016.

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

[8] Chuanjia Hou, Tong Jia, Yifan Wu, Ying Li, and Jing Han. Diagnosing performance issues in microservices with heterogeneous data source. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 493–500. IEEE, 2021.

[9] Lingzhi Wang, Nengwen Zhao, Junjie Chen, Pinnong Li, Wenchi Zhang, and Kaixin Sui. Root-cause metric location for microservice systems via log anomaly detection. In *2020 IEEE international conference on web services (ICWS)*, pages 142–150. IEEE, 2020.

[10] Yingying Zhang, Zhengxiong Guan, Huajie Qian, Leili Xu, Hengbo Liu, Qingsong Wen, Liang Sun, Junwei Jiang, Lunting Fan, and Min Ke. Cloudrca: A root cause analysis framework for cloud computing platforms. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 4373–4382, 2021.

[11] Shenghui Gu, Guoping Rong, Tian Ren, He Zhang, Haifeng Shen, Yongda Yu, Xian Li, Jian Ouyang, and Chunan Chen. Trinityrcl: Multi-granular and code-level root cause localization using multiple types of telemetry data in microservice systems. *IEEE Transactions on Software Engineering*, 2023.

[12] Fengrui Liu, Yang Wang, Zhenyu Li, Rui Ren, Hongtao Guan, Xian Yu, Xiaofan Chen, and Gaogang Xie. Microcbr: Case-based reasoning on spatio-temporal fault knowledge graph for microservices troubleshooting. In *International Conference on Case-Based Reasoning*, pages 224–239. Springer, 2022.

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

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

[15] Hanzhang Wang, Zhengkai Wu, Huai Jiang, Yichao Huang, Jiamu Wang, Selçuk Köprü, and Tao Xie. Groot: An event-graph-based approach for root cause analysis in industrial settings. *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 419–429, 2021.

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

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

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

[19] Shenglin Zhang, Yongxin Zhao, Sibo Xia, Shirui Wei, Yongqian Sun, Chenyu Zhao, Shiyu Ma, Junhua Kuang, Bolin Zhu, Lemeng Pan, et al. No more data silos: Unified microservice failure diagnosis with temporal knowledge graph. *IEEE Transactions on Services Computing*, 2024.

[20] https://anonymous.4open.science/r/microservo.

[21] https://github.com/FudanSELab/serverless-trainticket.

[22] https://github.com/GoogleCloudPlatform/microservices-demo.

[23] https://github.com/microservices-demo/microservices-demo.

[24] https://github.com/delimitrou/DeathStarBench/tree/master/socialNetwork.

[25] https://principlesofchaos.org/.

[26] https://github.com/prometheus/prometheus.

[27] https://github.com/elastic/beats.

[28] https://github.com/elastic/apm.

[29] https://github.com/chaos-mesh/chaos-mesh.

[30] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1285–1298, New York, NY, USA, 2017. Association for Computing Machinery.

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

[32] Anunay Amar and Peter C. Rigby. Mining historical test logs to predict bugs and localize faults in the test logs. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 140–151, 2019.

[33] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R. Lyu, and Dongmei Zhang. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, page 60–70, New York, NY, USA, 2018. Association for Computing Machinery.

[34] Hiroki Ikeuchi, Akio Watanabe, Takehiro Kawata, and Ryoichi Kawahara. Root-cause diagnosis using logs generated by user actions. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7, 2018.

[35] Tong Jia, Pengfei Chen, Lin Yang, Ying Li, Fanjing Meng, and Jingmin Xu. An approach for anomaly diagnosis based on hybrid graph model with logs for distributed services. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 25–32, 2017.

[36] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. Swisslog: Robust anomaly detection and localization for interleaved unstructured logs. *IEEE Transactions on Dependable and Secure Computing*, 20(4):2762–2780, 2023.

[37] Jieyu Lin, Qi Zhang, Hadi Bannazadeh, and Alberto Leon-Garcia. Automated anomaly detection and root cause analysis in virtualized cloud infrastructures. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 550–556, 2016.

[38] Ping Liu, Yu Chen, Xiaohui Nie, Jing Zhu, Shenglin Zhang, Kaixin Sui, Ming Zhang, and Dan Pei. Fluxrank: A widely-deployable framework to automatically localizing root cause machines for software service failure mitigation. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pages 35–46, 2019.

[39] Weilan Lin, Meng Ma, Disheng Pan, and Ping Wang. Facgraph: Frequent anomaly correlation graph mining for root cause diagnose in micro-service architecture. *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8, 2018.

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

[41] Pengfei Chen, Yong Qi, Pengfei Zheng, and Di Hou. Causeinfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 1887–1895, 2014.

[42] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, page 683–694, New York, NY, USA, 2019. Association for Computing Machinery.

[43] Nina Marwede, Matthias Rohr, André van Hoorn, and Wilhelm Hasselbring. Automatic failure diagnosis support in distributed large-scale software systems based on timing behavior anomaly correlation. In *2009 13th European Conference on Software Maintenance and Reengineering*, pages 47–58, 2009.

[44] Haibo Mi, Huaimin Wang, Yangfan Zhou, Michael Rung-Tsong Lyu, and Hua Cai. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1245–1255, 2013.

[45] Yufeng Li, Guangba Yu, Pengfei Chen, Chuanfu Zhang, and Zibin Zheng. Microsketch: Lightweight and adaptive sketch based perfor-

mance issue detection and localization in microservice systems. In *Service-Oriented Computing: 20th International Conference, ICSOC 2022, Seville, Spain, November 29 – December 2, 2022, Proceedings*, page 219–236, Berlin, Heidelberg, 2022. Springer-Verlag.

[46] Cuong Pham, Long Wang, Byung Chul Tak, Salman Baset, Chunqiang Tang, Zbigniew Kalbarczyk, and Ravishankar K. Iyer. Failure diagnosis for distributed systems using targeted fault injection. *IEEE Transactions on Parallel and Distributed Systems*, 28(2):503–516, 2017.

[47] D. Li, S. Zhang, Y. Sun, Y. Guo, Z. Che, S. Chen, Z. Zhong, M. Liang, M. Shao, M. Li, S. Liu, Y. Zhang, and D. Pei. An empirical analysis of anomaly detection methods for multivariate time series. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pages 57–68, Los Alamitos, CA, USA, oct 2023. IEEE Computer Society.

[48] Ying Fu, Meng Yan, Zhou Xu, Xin Xia, Xiaohong Zhang, and Dan Yang. An empirical study of the impact of log parsers on the performance of log-based anomaly detection. *Empirical Software Engineering*, 28(1):6, 2023.

[49] Van-Hoang Le and Hongyu Zhang. Log-based anomaly detection with deep learning: How far are we? In *Proceedings of the 44th international conference on software engineering*, pages 1356–1367, 2022.

[50] Nengwen Zhao, Honglin Wang, Zeyan Li, Xiao Peng, Gang Wang, Zhu Pan, Yong Wu, Zhen Feng, Xidao Wen, Wenchi Zhang, et al. An empirical investigation of practical log anomaly detection for online service systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1404–1415, 2021.

[51] Edward Chuah, Arshad Jhumka, and Sai Narasimhamurthy. An empirical study of major page faults for failure diagnosis in cluster systems. *The Journal of Supercomputing*, 79(16):18445–18479, 2023.

[52] Shilin He, Xu Zhang, Pinjia He, Yong Xu, Liqun Li, Yu Kang, Minghua Ma, Yining Wei, Yingnong Dang, Saravanakumar Rajmohan, et al. An empirical study of log analysis at microsoft. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1465–1476, 2022.