Rethinking the Evaluation of Microservice RCA with a Fault Propagation-Aware Benchmark

AOYANG FANG, SONGHAN ZHANG, YIFAN YANG, HAOTONG WU, JUNJIELONG XU, XUYANG WANG, RUI WANG, MANYI WANG, QISHENG LU, and PINJIA HE*, The Chinese University of Hong Kong, Shenzhen, China

While cloud-native microservice architectures have revolutionized software development, their inherent operational complexity makes failure Root Cause Analysis (RCA) a critical yet challenging task. Numerous data-driven RCA models have been proposed to address this challenge. However, we find that the benchmarks used to evaluate these models are often too simple to reflect real-world scenarios. Our preliminary study reveals that simple rule-based methods can achieve performance comparable to or even surpassing state-of-the-art (SOTA) models on four widely used public benchmarks. This finding suggests that the oversimplification of existing benchmarks might lead to an overestimation of the performance of RCA methods.

To further investigate the oversimplification issue, we conduct a systematic analysis of popular public RCA benchmarks, identifying key limitations in their fault injection strategies, call graph structures, and telemetry signal patterns. Based on these insights, we propose an automated framework for generating more challenging and comprehensive benchmarks that include complex fault propagation scenarios. Our new dataset contains 1,430 validated failure cases from 9,152 fault injections, covering 25 fault types across 6 categories, dynamic workloads, and hierarchical ground-truth labels that map failures from services down to code-level causes. Crucially, to ensure the failure cases are relevant to IT operations, each case is validated to have a discernible impact on user-facing SLIs.

Our re-evaluation of 11 SOTA models on this new benchmark shows that they achieve low *Top@1* accuracies, averaging 0.21, with the best-performing model reaching merely 0.37, and execution times escalating from seconds to hours. From this analysis, we identify three critical failure patterns common to current RCA models: *scalability issues*, *observability blind spots*, and *modeling bottlenecks*. Based on these findings, we provide actionable guidelines for future RCA research. We emphasize the need for robust algorithms and the co-development of challenging benchmarks. To facilitate further research, we publicly release our benchmark generation framework, the new dataset, and our implementations of the evaluated SOTA models.

CCS Concepts: • **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Additional Key Words and Phrases: Do, Not, Us, This, Code, Put, the, Correct, Terms, for, Your, Paper

ACM Reference Format:
Aoyang Fang, Songhan Zhang, Yifan Yang, Haotong Wu, Junjielong Xu, Xuyang Wang, Rui Wang, Manyi Wang, Qisheng Lu, and Pinjia He. 2018. Rethinking the Evaluation of Microservice RCA with a Fault Propagation-Aware Benchmark. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 20 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The widespread adoption of cloud-native microservice architectures, while enabling substantial business agility, has also introduced significant operational complexity. In such distributed systems, failures are not a matter of if, but when [14]. The complex web of service dependencies means that a single fault can trigger a cascade of failures, propagating rapidly throughout the system. With enterprise downtime costs estimated to exceed

[*] Corresponding author.

Authors' Contact Information: Aoyang Fang, aoyangfang@link.cuhk.edu.cn; Songhan Zhang, 222010549@link.cuhk.edu.cn; Yifan Yang, yifanyang6@link.cuhk.edu.cn; Haotong Wu, haotongwu@link.cuhk.edu.cn; Junjielong Xu, junjielongxu@link.cuhk.edu.cn; Xuyang Wang, xuyangwang@link.cuhk.edu.cn; Rui Wang, 224040299@link.cuhk.edu.cn; Manyi Wang, 225045034@link.cuhk.edu.cn; Qisheng Lu, qishenglu@link.cuhk.edu.cn; Pinjia He, hepinjia@cuhk.edu.cn, The Chinese University of Hong Kong, Shenzhen, Shenzhen, China.

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

$23,000 per minute [15], the ability to perform rapid and accurate Root Cause Analysis (RCA) has evolved from a technical requirement into a critical business imperative. Automating the pinpointing of a fault's origin from vast and varied telemetry data (metrics, logs, and traces), which can reduce the Mean Time To Recovery (MTTR), is therefore essential [48]. The research community has produced a variety of data-driven RCA approaches. Early work often concentrated on a single data modality, such as logs [27, 31, 50], metrics [21, 38], or traces [16, 17, 22, 36, 37, 46]. More recently, a clear trend has emerged towards multi-modal analysis. These approaches integrate heterogeneous data sources to construct a more holistic system view, promising significantly enhanced diagnostic precision [18, 23, 24, 34, 35, 39, 44, 47, 49, 51, 52].

Despite the apparent sophistication of these models, we find their effectiveness is often evaluated against benchmarks that oversimplify real-world failure scenarios. This concern is substantiated by our observation that simple rule-based methods can match or even surpass complex state-of-the-art (SOTA) models on four widely-used public benchmarks. This discrepancy prompts us to ask a fundamental question: Is the reported progress of SOTA algorithms a true reflection of their advanced designs, or is it an artifact of the inherent simplicity of the benchmarks used for their evaluation? We hypothesize that existing public benchmarks lack the complexity and scale required to meaningfully differentiate the capabilities of sophisticated RCA models.

To rigorously investigate this hypothesis, we conduct a systematic empirical study of popular public benchmarks, examining their fault injection strategies, dependency graph structures, and telemetry signal patterns. Our analysis reveals a critical limitation: in the majority of cases in existing datasets, the fault symptom of the root-cause service the most prominent. This characteristic makes the root cause easily identifiable through simple correlation, obviating the need for complex causal inference that advanced models are designed to perform.

Motivated by this clear gap, we argue for the necessity of systematically designed benchmarks that significantly increase the complexity and challenge for RCA algorithms (e.g., introducing more intricate fault propagation patterns). To this end, this paper proposes a framework that automates the generation, collection, and validation of failure scenarios at scale. Specifically, our framework incorporates several key innovations over prior work. (1) **Systematic fault generation**. Instead of relying on a small set of manually specified injections, our framework systematically explores a vast fault space defined by 31 fault types across 6 categories by layered random sampling. This allows for the generation of diverse and complex failure scenarios, including those that are difficult to orchestrate manually. (2) **Impact-driven validation**. Acknowledging that not all injected faults lead to observable issues, we introduce a pragmatic validation oracle that filters for failures causing detectable degradation in user-facing SLIs (e.g., success rate, latency). Through our impact-driven validation process, we identify 25 fault types that consistently produce user-facing degradations, ensuring that our final benchmark contains only operationally relevant failure scenarios. While this approach intentionally focuses on user-impactful failures, which represent a primary concern for SREs, it provides a consistent and automatable standard for dataset quality, a feature absent in prior benchmarks. (3) **Highly dynamic system environment**. We implement our framework on TrainTicket [10], one of the largest open-source microservice systems, which simulates a real-world online ticketing platform with 50 services. To create a more complex and dynamic environment than previous static, low-QPS (Queries Per Second) benchmarks, we subject the system to a dynamic, state-machine-driven workload. Using this approach, we generate a new benchmark dataset comprising 1,430 distinct validated failure cases. This dataset is uniquely annotated with multi-modal telemetry data and a hierarchical root cause labeling scheme. Unlike existing datasets that typically provide one-to-one mappings, our hierarchical labels recognize that a fine-grained failure inherently implies a coarse-grained one. For instance, a function failure necessarily means its containing service is also faulty.

During the dataset construction process, we identify a critical phenomenon often overlooked in existing benchmarks: a large portion of injected faults are silent. That is, they do not produce any user-facing impact. This observation highlights the vastness of the potential fault space and underscores the necessity of an impact-driven validation process to create a benchmark focused on operationally relevant problems. While we initially designed 31 fault types, our validation process revealed that only 25 consistently produced user-facing degradations. Although this set of 25 fault types may not perfectly mirror the distribution of production incidents, it represents a far broader and more structured exploration of failure modes—including code-level, network, and resource faults—than any existing public dataset.

In the re-evaluation stage, we select 11 recent SOTA RCA methods spanning different data modalities and architectural designs, including single-modal (metrics, traces) and multi-modal approaches. Rather than simply re-running existing implementations, we systematically re-engineer these methods into a unified evaluation framework with standardized input interfaces. Our implementation follows modern DevOps practices, containerizing each algorithm and enabling deployment on Kubernetes clusters. This standardized approach not only ensures fair comparison but also establishes a foundation for the broader RCA research ecosystem, where future datasets adhering to our data format can be seamlessly evaluated across all implemented methods. On our benchmark, these methods achieve an average *Top@1* accuracy of only 0.21, with the best-performing method reaching merely 0.37, underscoring considerable room for further improvement. Furthermore, their execution times increased substantially from mere seconds on existing benchmarks to, in some cases, several hours, exposing severe scalability issues. This dramatic performance degradation reveals deep, systemic weaknesses in current data-driven RCA approaches. Through systematic failure mode analysis, we identify three critical patterns that expose fundamental limitations in existing models. (1) **Scalability issues**. Methods struggle with increased telemetry volume, with execution times escalating from seconds to hours. For example, CausalRCA [44] requires over 10 minutes to process a single failure case (8 minutes of data) in our benchmark. (2) **Observability blind spots**. Models fail when critical services lack sufficient telemetry data, a common scenario in production where monitoring coverage is incomplete. (3) **Modeling bottlenecks**. Existing approaches rely on assumptions that break under complex failure conditions, such as stable request rates during incidents.

Based on these findings, we distill actionable guidelines for future RCA research. A critical insight concerns the gap between academic assumptions of complete observability and the reality of production systems with incomplete telemetry coverage. We argue that academic benchmarks must proactively simulate these imperfect conditions to foster the development of robust models prepared for production deployment.

To summarize, this paper makes the following contributions:

- We are the first to systematically reveal the over-simplicity of widely used public RCA benchmarks, demonstrating that their perceived difficulty is an artifact of simplistic evaluation.
- We introduce a novel framework to synthesize a new large-scale, multi-modal benchmark that embodies complex failure phenomena validated by end-user impact.
- Through a large-scale re-evaluation, we identify and analyze a set of critical, previously unreported failure patterns in SOTA models, revealing their fragility to complex causality, observability blind spots, and scalability limitations.
- We establish a standardized and reproducible evaluation ecosystem for RCA research by re-engineering 11 SOTA algorithms into a unified, containerized framework, facilitating fair and scalable evaluation.
- Based on our findings, we distill actionable guidelines for future research and publicly release our entire research artifact, including the benchmark, generation framework, and re-engineered models, to foster community progress.

2 Related Work

2.1 Data-Driven Root Cause Analysis in Microservices

The adoption of microservice architecture has been a paradigm shift in building scalable and resilient cloud applications. However, this architectural style introduces significant complexity for system maintenance. A single user request may traverse dozens of loosely coupled services, and a fault in one service can propagate unpredictably, causing a cascade of failures across the system. This distributed nature makes manual root cause analysis (RCA) a daunting and time-consuming task for engineers [48].

To address this challenge, the field has moved towards automated, data-driven RCA. These approaches leverage the vast amount of monitoring data generated by modern cloud systems, which primarily includes metrics, logs, and traces. Initial approaches primarily relied on a single data modality, including metric-based methods [20, 21, 25, 27, 28, 31, 41, 45], log-based methods [22, 30], and trace-based methods [43]. However, recognizing that a holistic system view is essential, researchers started developing methods that fuse metrics, logs, and traces [19, 24, 34, 35, 44, 47, 49]. By correlating information across different data types, these advanced approaches can capture more complex failure patterns, leading to substantial improvements in diagnostic performance.

2.2 Studies in Root Cause Analysis for Microservices

A recent comprehensive study by Pham et al. [32] provides a critical evaluation of the SOTA in causal inference-based RCA. They benchmarked an extensive suite of causal discovery (nine) and RCA methods (twenty-one), concluding that no single algorithm is universally superior, and many struggle to outperform random selection, especially in large-scale systems. Crucially, their work highlights a fundamental challenge that motivates our own:

the performance of methods on existing synthetic datasets does not reliably predict their efficacy on real-world systems. Pham et al. identified this symptom of inconsistent evaluation results stemming from flawed datasets. Our research, therefore, addresses the underlying cause by focusing on the methodological construction of a more faithful and controlled evaluation benchmark. We aim to extend their work by providing a more reliable foundation upon which the true capabilities of RCA algorithms can be more accurately assessed.

3 Preliminary Study

This section presents a preliminary study designed to evaluate the inherent challenges within mainstream public microservices RCA datasets. The primary objective is to scientifically assess the “simplicity” of these datasets and demonstrate a causal link between their structural and behavioral characteristics and the ostensibly superior performance of SOTA RCA models. The findings will serve as a foundational justification for the need to construct a new, more realistic, and challenging benchmark. This study is guided by three core research questions (RQs):

- *RQ1: What are the primary structural and runtime characteristics of failure scenarios in mainstream public RCA datasets?*
- *RQ2: How does a simple heuristic method perform on these datasets compared to SOTA models?*
- *RQ3: What explains the strong performance of SimpleRCA on existing benchmarks?*

3.1 Study Design

3.1.1 Datasets. For a comprehensive evaluation, we selected the most commonly used and publicly available multimodal datasets in the microservice RCA domain. These datasets cover a range of systems, including TrainTicket (TT) [10], OnlineBoutique (OB) [8], SockShop (SS) [9], SocialNetwork (SN) [3], and MicroSS [5].

- **Eadro [4]**: Includes two sub-datasets, Eadro-TT and Eadro-SN, designed with a specific focus on network and resource faults.
- **Nezha [7]**: Includes two sub-datasets, Nezha-TT and Nezha-OB, representing relatively small-scale microservice architectures.
- **RCAEval [33]**: A benchmark suite Spanning three generations (RE1, RE2, RE3), each including three system scenarios. We didn’t include RE1 because it is a unimodal dataset.
- **AIOps-2021 [2]**: The dataset from the AIOps Challenge in 2021, known for its large volume of observational data and various failure types.
- **GAIA [5]**: A dataset with a large number of fault samples generated from real cloud environment traffic.

3.1.2 Heuristic Method Design (SimpleRCA). To address RQ2, we designed **SimpleRCA**, a heuristic method simulating domain experts’ intuitive troubleshooting logic. It takes multi-modal service data as input, uses rule-based anomaly detectors to generate alerts, and identifies the service with the most alerts as the root cause. Tailored rules for each data modality:

- **Metrics**: For each service metric, a threshold-based detector is used. The threshold is dynamically determined using robust statistical methods, such as the 3-sigma rule or the 95th percentile (P95) of its historical data. An alert is triggered if a metric’s current value exceeds this threshold.
- **Traces**: The P95 latency for each service is calculated from its traces. A detector triggers an alert if this latency value is a certain multiple (e.g., 3x) of its normal-period baseline, indicating a significant performance degradation.
- **Logs**: A simple keyword-matching approach is employed. The total count of log entries containing common error keywords (e.g., ERROR, FAIL, EXCEPTION) for each service is tallied. An alert is generated if this count crosses a predefined threshold, assuming a sudden spike in errors.

These rules are all basic and commonly used anomaly detection rules. SimpleRCA is not a practical RCA solution, but a tool to test whether existing datasets capture real-world complex fault-cause relationships.

3.1.3 Experimental Setup. **SOTA baseline models**: We selected the best-performing methods from the respective benchmark papers (e.g., Eadro, Nezha, Baro) and general-purpose SOTA approaches (e.g., ART for AIOps-2021) for fair comparison.

Evaluation metrics: We utilized a set of widely used evaluation metrics in the RCA domain, including:

- $Top@K$: The proportion of correct root cause included in the top K predictions.
- $Avg@K$: The average rank of the correct root cause within the top K predictions.
- *MRR* (Mean Reciprocal Rank): The average reciprocal rank for prediction accuracy.

Table 1. Key Statistics of Existing RCA Benchmark Datasets. We measure scale by the number of failure cases (# Cases), services (# Svcs), total duration, and data volume (logs, traces). Complexity and dynamism are measured by the number of services covered by requests, (# Svc. by Req), Queries Per Second (QPS), max call depth (Max Depth), metric diversity (# Metrics) and metric sampling interval (Spl Interval).

Dataset	System	# Cases	# Svcs	# Svc. by Requests	Max Depth	# Logs (M)	# Metrics	Spl Interval (s)	# Traces (M)	Duration	QPS
Nezha-TT	Train-Ticket	45	46	28	4	0.16	19	60	0.004	8.9hr	0.13
Nezha-OB	Online-Boutique	56	10	10	2	2.20	19	60	0.047	8.1hr	1.59
Eadro-TT	Train-Ticket	81	27	13	3	0.01	7	1	0.016	17.8hr	0.25
Eadro-SN	SocialNetwork	36	12	12	3	0.08	7	1	0.072	2.4hr	8.38
AIOps-2021	E-commerce	159	18	12	4	5.30	483	30	2.94	13.3hr	61.63
GAIA	MicroSS	1097	10	10	2	14.60	949	30	3.1	31days	1.16
RE2-TT	Train-Ticket	90	69	27	5	21.30	8	1	0.56	36hr	4.35
RE3-TT	Train-Ticket	30	69	27	3	1.60	8	1	0.072	15hr	1.34
RE2-OB	Online-Boutique	90	17	7	2	15.00	8	1	2.1M	35.7hr	16.42
RE3-OB	Online-Boutique	30	28	7	2	2.10	8	1	0.28	12hr	6.51
RE2-SS	Sock-Shop	90	16	NA	NA	7.60	8	1	NA	36hr	NA
RE3-SS	Sock-Shop	30	26	NA	NA	2.30	8	1	NA	12hr	NA

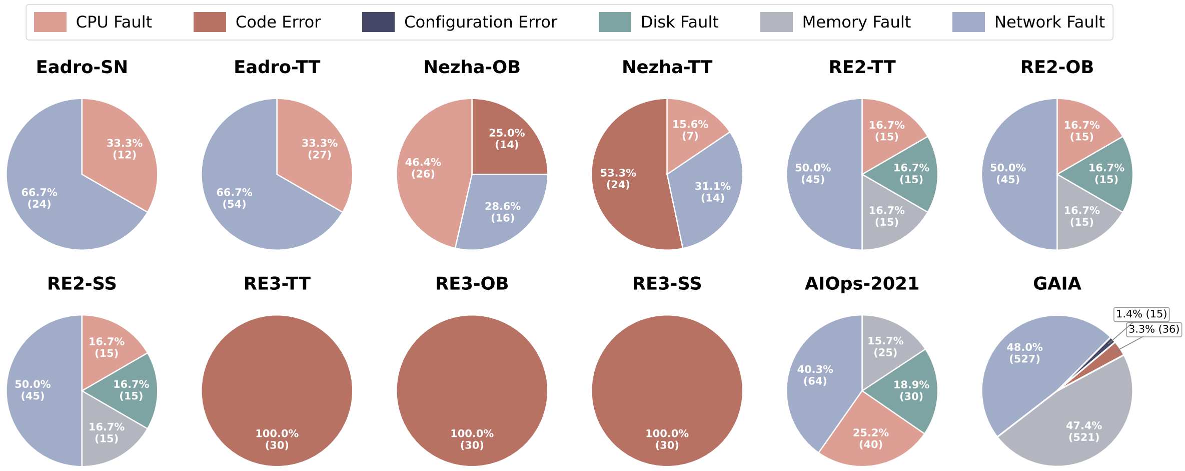

Fig. 1. Fault type distribution of existing benchmark datasets.

- Running Time: The time of model execution, evaluating the efficiency of the algorithm.

3.2 Results and Analysis

3.2.1 Answering RQ1: Dataset Characteristics. We first analyzed the static characteristics of existing RCA benchmarks to uncover their inherent structural and runtime properties. Table 1 summarizes the key statistics, and Fig. 1 shows the fault type distribution.

A review of the static features reveals key limitations that simplify the RCA task. **(1) Limited fault cases.** Except GAIA, all datasets contain fewer than 200 faults (e.g., Eadro and Nezha <100, AIOps-2021 only 159). This scarcity of samples poses a fundamental challenge for complex models to learn robust and generalizable patterns, and the strong performance reported by SOTA methods may largely reflect overfitting to limited scenarios rather than true generalization ability. **(2) Simplified request structures.** Even in datasets with many services (e.g., RE2-TT and RE3-TT with 69 services), all requests collectively cover only 27 services. and has extremely short traces, with seven datasets max out at just 2-3 hops. However, real-world microservice systems typically involve much deeper propagation chains and larger service graphs. These simplified structures reduce the search space and fail to capture the intricate dependencies and cascading failures observed in production environments. **(3) Narrow fault spectrum.** Fault types are few and imbalanced. We categorize them into six classes (e.g., CPU, Code, Configuration, Disk, Memory, and Network) based on the fault definitions, but most datasets contain only one or two types (e.g., Eadro sub-datasets and RE3 series) or exhibit severe skew (e.g., GAIA).

Table 2. Performance Comparison of SimpleRCA and SOTA Methods.

Dataset	Method	$Top@1$	$Top@3$	$Top@5$	$Avg@3$	$Avg@5$	MRR	Time(s)
Re2-OB	BARO	0.14	0.93	0.98	0.64	0.77	0.54	**0.09**
	SimpleRCA	**0.47**	**0.93**	**1.00**	**0.68**	**0.80**	0.67	1.14
RE2-TT	BARO	0.67	0.84	0.89	0.77	0.81	0.76	**0.87**
	SimpleRCA	**0.80**	**0.87**	**0.93**	**0.84**	**0.87**	**0.85**	5.99
RE2-SS	BARO	0.14	0.82	0.94	0.50	0.67	0.47	**0.06**
	SimpleRCA	**0.24**	**0.88**	**0.97**	**0.59**	**0.74**	0.55	0.20
RE3-OB	BARO	0.00	0.90	0.90	0.59	0.71	0.45	**0.06**
	SimpleRCA	**0.30**	**0.90**	**0.90**	**0.62**	**0.73**	**0.56**	0.50
RE3-TT	BARO	0.50	0.93	1.00	0.77	0.86	0.72	**0.26**
	SimpleRCA	**0.83**	**1.00**	**1.00**	**0.93**	**0.96**	**0.91**	1.02
RE3-SS	BARO	0.00	0.83	**0.93**	0.46	0.65	0.40	**0.07**
	SimpleRCA	**0.83**	**0.90**	0.90	**0.88**	**0.89**	**0.87**	0.15
Nezha-TT	Nezha	0.87	0.98	0.98	NA	NA	NA	NA
	SimpleRCA	**0.93**	**0.98**	**0.98**	0.96	0.97	0.96	1.81
Nezha-OB	Nezha	**0.93**	**0.96**	**0.96**	NA	NA	NA	NA
	SimpleRCA	0.91	0.91	0.91	0.91	0.91	0.91	5.05
Eadro-TT	Eadro	**0.99**	**0.99**	**0.99**	NA	NA	NA	NA
	SimpleRCA	0.81	0.83	0.83	0.82	0.83	0.82	0.11
Eadro-SN	Eadro	**0.97**	**0.99**	**0.99**	NA	NA	NA	NA
	SimpleRCA	0.83	0.83	0.83	0.83	0.83	0.83	0.60
AIOps-2021	ART	**0.72**	**0.89**	NA	NA	**0.87**	NA	NA
	SimpleRCA	0.63	0.82	0.91	0.74	0.80	0.74	4.25

Existing RCA datasets lack realism. They feature few fault cases, simplified request structures, and highly imbalanced fault type distributions. These limitations result in benchmarks that hardly approximate real-world complexity, potentially overstating model effectiveness.

3.2.2 Answering RQ2: Performance Comparison. To address RQ2, we compared SimpleRCA with selected SOTA models on their respective datasets (Table 2). Baseline results were taken directly from the original papers to ensure faithful and reproducible comparisons.[1]

Accuracy. SimpleRCA's $Top@1$ accuracy is comparable to or surpasses SOTA methods on multiple datasets. For instance, it achieves 0.93 on Nezha-TT versus Nezha's 0.87, and 0.83 on RE3-TT and RE3-SS versus BARO's 0.50 and 0.00. Even where SOTA models slightly outperform SimpleRCA, the gap is within 0.1, except for Eadro, where the relatively high accuracy is largely due to substantial overlaps in patterns between the training and testing sets. Furthermore, the advantage of SOTA methods does not widen with relaxed $Top@K$ metrics ($Top@3$, $Top@5$), underscoring the robustness of SimpleRCA across ranking thresholds.

Efficiency. SimpleRCA is orders of magnitude faster than SOTA models, except for the metric-based Baro. However, Baro is a unimodal method that performs far worse than SimpleRCA, with an average Top@1 score on the RCAEVAL dataset that is 0.34 lower (0.24 vs. 0.58). This further emphasizes that on these public benchmark datasets, SOTA models do not offer a performance advantage commensurate with their computational cost.

SimpleRCA matches or surpasses SOTA performance at far lower cost. Despite its simplicity, SimpleRCA attains comparable or superior accuracy to SOTA methods while being orders of magnitude faster, indicating that most faults in existing benchmark datasets can be addressed by simple rule-based methods, making it difficult to effectively evaluate SOTA approaches.

[1] We did not include the GAIA dataset in our experiments because it suffers from issues such as inconsistencies in ground-truth annotations (e.g., many injected faults are reflected in the logs, but only a small subset is labeled), which may introduce bias into the evaluation.

Table 3. Statistics on Completeness of Obs ervability and Fault Injection Pattern Classification of Existing Datasets. Note: # Cases: Count of cases; # Incom. Cases: Count of cases with incomplete data; R. of Incom. Cases: Rate of incomplete cases; R. of Type *i*: Rate of cases belonging to Type *i* (i=I,II,III).

Dataset	**Completeness of Observability**			**Fault Injection Pattern Classification**			
	# Cases	**# Incom. Cases**	**R. of Incom. Cases**	**R. of Type I**	**R. of Type II**	**R. of Type III**	**R. of Type I&II**
Nezha-TT	45	45	1.00	0.93	0.07	0.00	1.00
Nezha-OB	56	56	1.00	0.91	0.09	0.00	1.00
Eadro-TT	81	81	1.00	0.75	0.25	0.00	1.00
Eadro-SN	36	36	1.00	0.50	0.44	0.06	0.96
AIOps-2021	159	159	1.00	0.63	0.16	0.21	0.79
RE2-TT	90	90	1.00	0.67	0.17	0.17	0.83
RE3-TT	30	30	1.00	0.83	0.00	0.17	0.83
RE2-OB	90	86	0.96	0.59	0.28	0.13	0.87
RE3-OB	30	30	1.00	0.30	0.00	0.70	0.30
RE2-SS	90	90	1.00	0.60	0.27	0.13	0.87
RE3-SS	30	30	1.00	0.83	0.00	0.17	0.83
All-Dataset	737	733	0.99	0.68	0.18	0.14	0.86

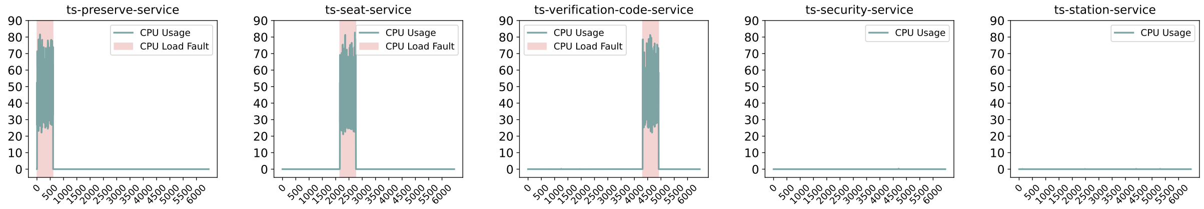

Fig. 2. Case Study: CPU load fault in the Eadro dataset. Fault signatures are concentrated only on the injected service; other services, like ts-security-service and ts-station-service, show no observable fluctuations.

3.2.3 *Answering RQ3: Deficiencies Analysis.*

Based on the fact that SimpleRCA can achieve such superior performance on existing benchmark datasets, we hypothesize that the root cause may lie in deficiencies in the quality of fault injection and observability data within these datasets. To systematically investigate this issue, we assess the quality of existing datasets along two dimensions: **(i) completeness of observability data, i.e., whether the key fault-relevant signals are missing, and (ii) simplicity of fault injection patterns, i.e., whether the faults are overly easy to localize**. These two dimensions evaluate data quality in terms of the sufficiency of observational signals and the authenticity and complexity of fault patterns.

Regarding the completeness of observability, we put forward two guiding principles. First, each fault type must be paired with its corresponding observability data (e.g., network faults necessitate full trace coverage; CPU faults require relevant metrics). Second, from the perspective of user experience, injected faults must propagate to user-facing status codes to reflect real-world disruptions. Otherwise, such faults fail to reflect realistic scenarios, where service disruptions are inevitably observable at the user level.

To assess the simplicity of fault injection patterns, we analyze service-level fault symptoms. A service is deemed to exhibit a *pronounced fault symptoms* if fault-relevant metrics or error counts in logs during the fault period exceed a preset threshold (e.g., twice the average of the preceding normal period); otherwise, it is categorized as showing only an *attenuated fault symptoms*. Based on this rule, each fault case is classified into three types:

(1) **Type I**: Pronounced fault symptoms only in the injected service (i.e., overly localized).
(2) **Type II**: No pronounced fault symptoms in any service (i.e., fault under-expressed).
(3) **Type III**: Stronger fault symptoms in services other than the injected one.

Type I highlights overly localized fault manifestations, while Type II raises concerns about the effectiveness of fault injection. To illustrate, Fig. 2 shows three CPU faults in Eadro classified as Type I. In these faults, only the injected service exhibits abnormal CPU metrics, while all other services maintain flat trends, making the faults easily identifiable but rarely representative of real-world cascading failures.

Table 3 shows that 0.99 of fault cases lack at least one type of observability data, while only four fault cases in the RE2-OB dataset meet our ”comprehensive observability” criterion. In terms of fault injection patterns, Type I

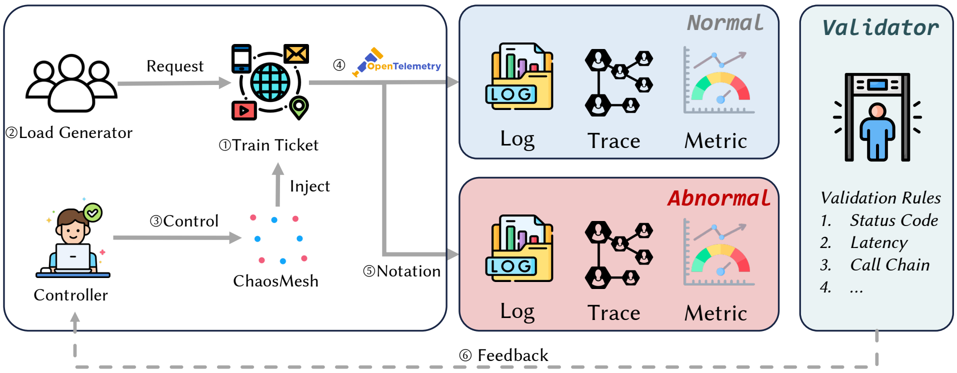

Fig. 3. The six-stage pipeline for our benchmark dataset construction, from system selection to the final validated and labeled failure cases.

and Type II together comprise 0.86 of all cases. The prevalence of Type I cases explains why SimpleRCA achieves surprisingly strong performance-faults confined to a single service are inherently easy to localize. By contrast, the existence of Type II cases sheds light on why many algorithms fail, without any pronounced fault signals, localization becomes nearly impossible. Some datasets (e.g., Nezha-TT, Nezha-OB) contain only Type I/II fault cases, suggesting that injected faults are either overly localized or too weak to manifest. Even in RE3-OB, where Type III cases account for 0.7, fault quality remains questionable because most are code-level issues captured in logs, and our classification rule only counts error entries in a straightforward manner.

Current RCA benchmarks suffer from two systemic shortcomings: (i) incomplete observability depriving models of essential diagnostic signals, and (ii) oversimplified or under-expressed fault patterns making evaluation unreliable. These issues highlight the urgent need for future benchmarks to include diverse, realistic fault scenarios with richer observability signals.

4 Dataset Construction

4.1 Overview

In this section, we introduce our systematic framework for constructing a comprehensive benchmark dataset for root cause analysis. As illustrated in Fig. 3, our framework operates as a closed-loop process designed to generate a diverse and impactful set of failure scenarios. The process begins with a robust **System Foundation (1)**, the Train-Ticket application, which is subjected to dynamic workloads from our **Load Generator (2)**. Concurrently, a **Controller (3)** orchestrates fault injections into the system using a structured **Fault Space**, which models a wide array of failure types. Throughout this process, a multi-modal observability pipeline collects comprehensive telemetry data, including metrics, logs, and traces, via **OpenTelemetry (4)**. Crucially, the precise parameters of each injected fault are recorded, forming the basis for our **Hierarchical Ground Truth Annotation (5)**. Finally, a **Validator (6)** automatically assesses the collected data against user-impact metrics (e.g., success rate and latency) to filter out silent faults and categorize the anomalies. The insights from this validation provide a **Feedback (7)** mechanism to the controller, enabling a human-in-the-loop strategy to guide subsequent fault injections toward underrepresented and more challenging failure types. This impact-driven, iterative approach ensures that the resulting dataset is not only large but also diverse and relevant for evaluating modern RCA techniques.

4.2 System Foundation and Enhancements

The quality of a microservice benchmark is fundamentally constrained by its underlying application. Most publicly available systems (e.g., SockShop [9], OnlineBoutique [8], SocialNetwork [3], MicroSS [1]) are demonstrators with only about a dozen services, which is difficult to simulate the intricate dependency webs of industrial production environments. To overcome this limitation, we selected Train-Ticket [10], the largest-to-date (as far

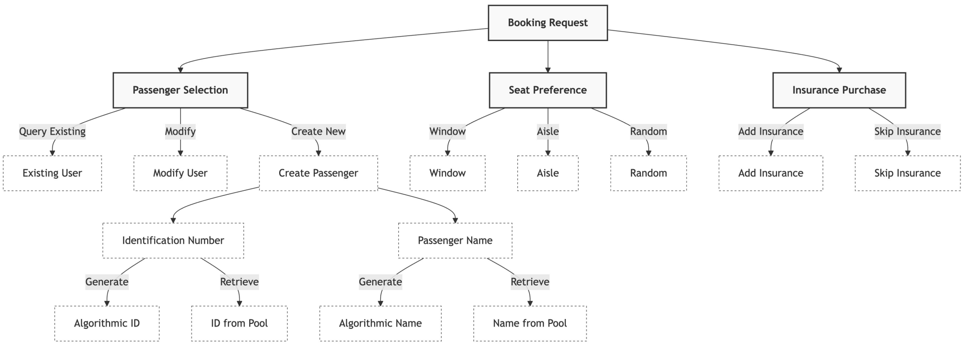

Fig. 4. State-machine-based workflow generation model, the booking request node depends three parameters, i.e., passenger, seat, and insurance. And each of the parameter can be fetched in different ways (either from creating new one, or query existing one). This recursive way leads to the combinational explosion.

as we know) open-source microservice system with 50 distinct services, as our foundation[2]. We further enhanced this foundation in two key ways:

- **Enhanced Observability**: We deployed a comprehensive monitoring stack on Kubernetes [11–13], improving the collection of runtime metrics and network telemetry to provide a more holistic view of the system's state.
- **System Stabilization**: We identified and rectified several inherent bugs and performance bottlenecks within the Train-Ticket application. For instance, we discovered that the default JVM heap size settings for several services were insufficient. Under increased QPS from our dynamic workload generator, these services exhibited non-deterministic latency spikes and instability, contaminating the baseline performance. By tuning these configurations and resolving other similar issues, we established a stable system baseline, which is crucial for isolating the impact of deliberately injected faults.

4.3 Dynamic Workload Generation

Previous benchmarks often use static request patterns, which fail to represent the complexity of real-world user interactions. To address this, we developed a dynamic load generator based on a state-machine model, as illustrated in Fig. 4. Our generator models a user workflow as a directed graph where each node represents a `state` (a required parameter for a request) and each edge represents a `transition` (a method to fulfill that state).

For instance, a `Booking Request` is composed of states like `Passenger Selection` and `Seat Preference`. Each state can be resolved through multiple transitions; e.g., `Passenger Selection` can be done by creating a new user or querying an existing one. Crucially, transitions can trigger sub-workflows to fulfill prerequisite sub-states, creating a recursive dependency model. This nested structure leads to a combinatorial explosion of unique execution paths (e.g., the booking workflow has $6 \times 3 \times 2 = 36$ combinations), ensuring a highly dynamic load that mimics real-world user variability. This approach enables the discovery of defects that static load patterns would miss. To foster reproducibility, our load generator will be made publicly available.

4.4 Fault Space Modeling

The scope and nature of injected faults are critical for a benchmark's ability to challenge RCA models. We used ChaosMesh to implement a structured fault space spanning **31 distinct fault types** across seven categories, as detailed in Table 4. This library includes traditional resource faults as well as more challenging types, such as code-level faults injected via JVM agents and network faults that manipulate HTTP semantics.

We discretize continuous parameters (e.g., CPU cores, latency intervals) to transform the abstract fault space into a concrete, enumerable set of experiments. The resulting fault space exceeds 10^{16} possible configurations, with fine-grained discretization to capture subtle parameter-dependent faults.

[2] Our version is built upon the Nezha [6] release of Train-Ticket, which instrumented application logs with trace IDs, enabling vital log-trace correlation.

Table 4. The 31 Fault Types Supported in Our Benchmark, Organized by Category.

Category	Fault Types
Resource	PodKill, PodFailure, ContainerKill, MemoryStress, CPUStress
Network	Delay, Loss, Duplicate, Corrupt, Bandwidth, Partition
HTTP	ReqAbort, RespAbort, ReqDelay, RespDelay, RespReplaceBody, RespPatchBody, ReqReplacePath, ReqReplaceMethod, RespReplaceCode
Code (JVM)	Latency, Return, Exception, GC, CPUStress, MemoryStress, MySQLLatency, MySQLException
DNS	DNSError, DNSRandom
Time	TimeSkew

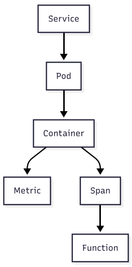

Fig. 5. The hierarchical structure of our ground truth labels. A fault can be localized at different levels of granularity, from the service down to the specific function.

4.5 Multi-modal Data Collection

We employed a standard observability stack based on OpenTelemetry to collect multi-modal data, including Kubernetes-level metrics, trace-correlated logs, distributed traces, and L4/L7 network telemetry. All data is normalized into a unified schema and stored in ClickHouse for efficient querying. This standardized schema, publicly available in our repository, simplifies data consumption for downstream RCA algorithms.

4.6 Hierarchical Ground Truth Annotation

A precise ground truth is essential for evaluating RCA models. In our framework, the ground truth for each failure case is directly derived from the fault injection parameters. This includes the specific fault type, its target (e.g., service, pod), and ensuring unambiguous root cause labels even in cases of complex, cascading failures. Previous studies have used different granularities for root cause labels, ranging from coarse service-level identification to fine-grained function-level pinpointing [39, 44], however, we argue that a more fine-grained label can contain the information of coarse-grained labels, while the reverse is not true. For example, knowing that the root cause is a specific function failure inherently implies that the parent service is also faulty, but not vice versa.

As a result, to support evaluation at varying levels of diagnostic precision, we introduce a hierarchical labeling scheme. This hierarchy is represented by a tree structure, as shown in Fig. 5. Each failure case is annotated with a set of labels corresponding to these levels of granularity: Service, Pod, Container, and where applicable, more specific identifiers for Metrics, Spans, or Functions. This structure provides researchers with the flexibility to evaluate their models at different levels of precision, from coarse service-level localization to fine-grained function-level identification[3].

4.7 Impact-Driven Selection

As discussed in preliminary study 3, a significant methodological flaw in existing benchmarks is the often-unspoken assumption that every injected fault manifests as an observable problem. In practice, many faults are silent or their effects are absorbed by system resilience mechanisms. To construct a benchmark of operationally relevant failures, we implement a strict, impact-driven selection process where a fault is included if and only if it produces a discernible anomaly at the system's entry points, as measured by user-facing Service Level Indicators (SLIs).

Our automated validation logic employs a hybrid approach to identify anomalies. For success rate degradations, it uses Z-tests to detect statistically significant drops. For latency anomalies, it combines hard thresholds (e.g., 10 seconds) with adaptive statistical thresholds based on historical performance patterns. This process classifies experiments into two categories: **Has Anomaly** (experiments with validated user-perceivable impact) and **No Anomaly** (filtered out from the final dataset). We acknowledge this is a trade-off that prioritizes operational relevance and scalable validation over the inclusion of subtle "gray failures," which we discuss further in Section 7.2.

4.8 Fault Injection Strategy

Given the vastness of the fault space (exceeding 10^{16} configurations), an exhaustive exploration is computationally infeasible. For the construction of the current dataset, we adopted a stratified random sampling strategy. We first

[3] In this study, we only use service level labels, since it is the only common granularity across all methods.

partitioned the entire fault space into strata based on the seven high-level fault categories defined in Table 4. Then, we performed random sampling of fault parameters within each stratum. This approach ensures that all major fault categories are represented in the dataset, providing a broad foundation for evaluation.

5 Study Design

Based on our preliminary study (Section 3), which revealed the inherent simplicity of existing RCA benchmarks, this section details the design of our main empirical study. The primary goal is to introduce and validate a more challenging benchmark and re-evaluate the true capabilities of SOTA models against it. This study is guided by two core research questions (RQs), which address the quantitative complexity of our new benchmark and the performance of existing models on it.

5.1 Research Questions

RQ4. How does our newly constructed benchmark quantitatively differ from existing public benchmarks in terms of scale, diversity, and complexity? To answer this, we will first present statistics from our dataset construction process to demonstrate the efficacy of our methodology. We will then introduce a set of indicators to quantify and compare key characteristics across benchmarks, including dependency graph complexity, workload dynamics, and the distribution of failure signatures. This analysis will substantiate our claim that the new benchmark represents a more diverse and challenging set of failure scenarios.

RQ5. How do SOTA RCA models perform on this new benchmark, and what are their primary failure modes when confronted with complex causality and incomplete observability? Through this question, we intend to probe the limitations of current SOTA models using our validated, challenging benchmark. We hypothesize that the increased complexity will lead to a significant performance degradation. By analyzing the failure modes against specific, engineered challenges, we aim to provide clear, actionable directions for future research in the field.

5.2 Evaluation Metrics

To evaluate model performance on our new benchmark, we adopt the same evaluation metrics established in our preliminary study (Section 3), which include effectiveness metrics (*Top@K*, *Avg@K*, *MRR*) and efficiency metrics (*Execution Time*). These metrics together provide a comprehensive and standardized view of model performance, enabling direct comparison between our findings on simple and complex benchmarks.

Additionally, to assess the scalability characteristics of different approaches, we conduct a dedicated scalability experiment that evaluates execution time across varying trace volumes, ranging from 2,000 to 18,000 traces. This experiment specifically focuses on models that utilize trace and log data, as these data types exhibit dramatic volume fluctuations in response to request patterns, unlike metrics data, which typically scales linearly with time. The scalability analysis provides crucial insights into the practical deployment considerations of each model under realistic workload variations.

5.3 Evaluation Setup

Our evaluation setup is designed to provide a rigorous and reproducible assessment of RCA models.

Baselines. To ensure a thorough and representative comparison, we evaluate against eleven recent SOTA RCA models that have been published in top-tier software engineering and systems venues. These models were selected to cover a wide spectrum of underlying techniques in the field. Specifically, our selection includes:

- (1) Graph-based models, such as *MicroRank* [43], *MicroRCA* [40], *MicroHECL* [29], and *MicroDig* [36], which leverage service dependency graphs and random walk algorithms to pinpoint faulty services.
- (2) Machine learning and deep learning approaches, such as *DiagFusion* [47], *Eadro* [24], and *Art* [35], which learn complex failure patterns from historical and multimodal data using techniques like GNNs and Transformers.
- (3) Causal inference and statistical methods, including *Shapleyiq* [26], *Nezha* [44], *CausalRCA* [42], and *Baro* [31], which aim to identify the causal contributors to a failure through techniques like Shapley values, event pattern mining, and hypothesis testing.

By comparing against this diverse set of baselines, we can draw more robust conclusions about the true state of the art and its limitations when facing complex failure scenarios. To ensure a fair comparison, we conducted a systematic, best-effort hyperparameter tuning for each model to adapt it to the unique characteristics of our new,

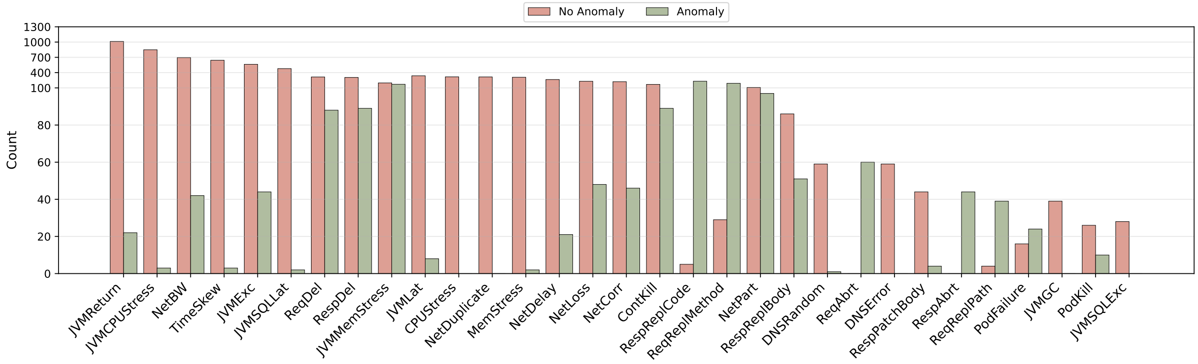

Fig. 6. Three types of the datasets collected and detected in the dataset generation process in terms of the injected fault types. The labels are labeled by the detector introduced in Section 4.7, which detects the user-aware anomalies.

more complex benchmark, rather than simply adopting the default configurations provided in the original papers, which were optimized for simpler, smaller-scale datasets.

Dataset. We conduct our evaluation on the new benchmark dataset proposed in this paper (detailed in Section 4). This benchmark is designed to challenge SOTA models through its dynamic workloads, diverse fault types, and complex causal propagation chains. Our dataset construction process follows a rigorous protocol: each data collection cycle consists of 4 minutes of normal operation data followed by 4 minutes of failure data. After each fault injection, the entire service environment is redeployed to ensure complete isolation and prevent interference from previous experiments. Before each normal data collection phase begins, the system undergoes a 4-minute warm-up period to reach a stable operational state, ensuring the reliability and consistency of our baseline measurements. For models that require training, such as Art [35], DiagFusion [47], and Eadro [24], we partition the dataset by fault type, allocating 80% for training and the remaining 20% for testing.

Environments. All experiments are conducted on a server equipped with two EPYC 9754 CPUs and 1.48TiB of DDR5 memory. For the dataset construction phase, we deploy a Kubernetes cluster consisting of 1 master node and 6 worker nodes, where each worker node is equipped with 128 CPU cores and 128GB of memory. For algorithm evaluation, each algorithm is executed within a controlled environment, constrained to 128 CPU cores and 128GB of memory. This setup guarantees that performance differences are attributable to the models' intrinsic capabilities rather than variations in computational resources.

6 Study Results

This section presents the results of our main empirical study, guided by the research questions established in Section 5. We first answer *RQ4* by quantitatively demonstrating our benchmark's increased scale and complexity compared to prior work. We then address *RQ5* by re-evaluating eleven SOTA models on this new benchmark, analyzing their performance degradation and systematically identifying three primary failure modes that reveal systemic weaknesses in current RCA approaches. This analysis provides a foundation for the actionable insights discussed in Section 7.

6.1 RQ4: New Dataset Characteristics

To quantitatively demonstrate the increased complexity and scale of our new benchmark, we first show the effectiveness of our impact-driven data generation process and then conduct a multi-faceted comparative analysis against prior work. Our evaluation centers on three critical dimensions of a benchmark's complexity and diagnostic challenge: **Workload Intensity**, **System Interaction Coverage**, and **Causal Complexity**.

6.1.1 Effectiveness of the Impact-Driven Generation Process. To construct a benchmark with genuine diagnostic challenges, we filtered out inconsequential fault injections through a rigorous, impact-driven selection process described in Section 4.7. Fig. 6 presents the outcome of this process across over 9,152 initial fault injection experiments spanning 31 distinct fault types, from which we ultimately derived our 1,430 validated failure cases. The distribution of these experiments across fault types is intentionally non-uniform, reflecting the intrinsic differences in the "injection space" for each fault category. For instance, JVM-related faults encompass a vast number of code methods as potential injection targets, whereas `PodKill` faults are limited to the much smaller number of available pods. To preserve the uniqueness and integrity of each experiment, we did not artificially balance these counts through duplicate injections; each experiment represents a distinct fault configuration.

> Effective fault injection requires careful calibration of both injection intensity and injection location to avoid undetectable weak faults, unreachable code paths, and trivially obvious strong faults.

Our analysis reveals that a vast majority of initial fault injections (84.4%) were classified as “No Anomaly”, failing to produce any user-perceivable impact. This high failure rate is not random but stems from two orthogonal and equally critical dimensions: injection intensity and injection location.

First, **resource-based fault injection requires delicate calibration within a narrow effective range**. ChaosMesh implements resource stress faults by spawning independent processes that compete for system resources. For `CPUStress` and `MemoryStress` faults, near 100% of our initial injections produced no impact due to intensity miscalibration. When stress levels are too low, the target application continues operating normally, rendering the injection ineffective. Conversely, when stress levels are too high, the system’s out-of-memory killer or scheduler intervenes to terminate the stress process, again nullifying the injection. This narrow effective window makes resource-based faults particularly challenging to calibrate and explains their high failure rate in producing meaningful diagnostic scenarios.

Second, **when injection location is poorly chosen, even high-intensity faults may never manifest**. This issue was most prominent in JVM-level manipulations, such as `JVMReturn` and `JVMCPUStress`. The primary reason for their failure was that the targeted methods were never invoked by our workload generator, rendering the fault injection inert regardless of its configured intensity. This dual failure mode validates our premise that many existing benchmarks, which often use minimal intensities and naive location selection, fail to generate meaningful diagnostic challenges.

In contrast, our process also identified faults that consistently produced clear and severe failures. Manipulations of core HTTP semantics, such as `HTTPResponseReplaceCode`(98%) and `HTTPRequestAbort` (100%), predictably led to “Has Anomaly” classifications. This is because these faults directly disrupt the fundamental communication protocols that microservices rely on, leading to immediate and obvious user-facing errors. After filtering out the inconsequential injections, 6 fault types are filtered out, since they never produced any impactful failures in our experiments. The final benchmark thus consists of 1,430 unique fault configurations across 25 fault types, each validated to produce meaningful user-perceivable impacts.

6.1.2 Comparative Analysis of Benchmark Characteristics. Building on our curated set of high-impact failures, we now compare our benchmark against existing datasets across three key dimensions of complexity: workload intensity, system interaction coverage, and causal complexity. Fig. 7 provides a summary of this comparison.

Workload Intensity and Data Volume. A benchmark’s diagnostic challenge is fundamentally tied to its workload intensity, which dictates the volume and velocity of telemetry data generated. We measure this intensity using Queries Per Second (QPS), derived from the total traces and duration. Our benchmark operates at an average of **16.47 QPS** (max is 37.6), with fluctuations inherent to its dynamic request generation; we did not increase the QPS further to avoid system instability that would impact data generation efficiency. This process generated a massive dataset of **154.7 million log lines** and **11.2 million traces** over 188.1 hours. This stands in contrast to existing benchmarks, whose intensity is often at a toy level. For instance, Nezha-TT [7], Eadro-TT [4], RE2-TT and RE3-TT [33], operate at a mere **0.14 QPS**, **0.25 QPS**, **4.35 QPS**, and **1.34 QPS**, respectively. This low intensity fails to simulate the high-concurrency conditions where failures often emerge from complex interactions under load. The large data volume in our benchmark is a direct consequence of this higher workload intensity, making it essential for assessing the scalability and noise-resistance of data-driven RCA models.

Breadth and Depth of System Interaction. Beyond data volume, a benchmark’s value is determined by its ability to exercise the system’s architectural complexity. The ability to trigger complex, cascading failures depends on how thoroughly the workload explores the microservice architecture. We assess this using two key indicators: the service count, representing the system’s potential scale, and the service coverage rate (the proportion of services exercised by a single trace), representing the interaction’s breadth. While some benchmarks like RE2-TT, Nezha-TT, and Eadro-TT span relatively large systems but still exhibit shallow workloads and low service coverage

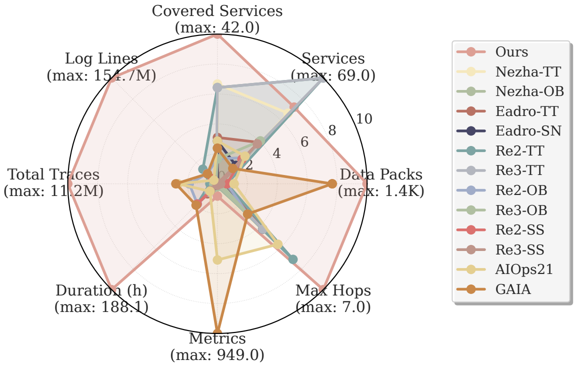

Fig. 7. Dataset characteristics in terms of eight key attributes. Our newly constructed benchmark demonstrates a balanced and challenging profile across all dimensions, including request covered services, log numbers, request numbers, durations, and data-packs (distinct injected faults).

Table 5. Algorithm performance comparison on our newly constructed benchmark dataset. The best results are highlighted in **bold**.

Algorithm	*Top*@1	*Top*@3	*Top*@5	Avg3	Avg5	MRR	Time (s)
			Full Dataset				
Baro	0.36	0.50	0.58	0.43	0.48	0.44	0.99
CausalRCA	0.22	0.40	0.42	0.33	0.36	0.31	927.11
MicroDig	0.35	**0.61**	0.67	**0.49**	0.56	**0.48**	19.03
MicroHECL	0.34	0.34	0.34	0.34	0.34	0.34	34.82
MicroRank	0.04	0.18	0.25	0.12	0.17	0.12	36.29
MicroRCA	**0.37**	0.50	0.64	0.43	0.51	0.45	35.64
Nezha	0.04	0.08	0.08	0.06	0.07	0.05	94.62
Shapleyiq	0.22	0.36	0.52	0.29	0.37	0.31	42.74
SimpleRCA	0.28	0.60	**0.80**	0.46	**0.58**	0.47	**0.90**
			Test Set				
DiagFusion	0.13	0.17	0.22	0.15	0.17	0.16	3.03
Eadro	0.16	0.16	0.16	0.16	0.16	0.16	32.75
Art	0.04	0.08	0.10	0.06	0.07	0.06	8.10

rates. In contrast, our dynamic workload generator achieves a service coverage rate of **84%** (42 out of 50 services), far exceeding RE2-TT's **39%** (27/69), Nezha-TT's **61%** (28/46) and Eadro-TT's **48%** (13/27). This high coverage ensures that injected faults are not isolated but can propagate widely through the system's dependency graph, creating complex diagnostic challenges that require a holistic system view.

Complexity of Causal Propagation. Finally, the ultimate test of an RCA benchmark is its ability to generate failures with long and complex causal chains. We use the **maximum service call depth** as a direct proxy for a benchmark's potential causal complexity. A deeper call chain allows for more intricate fault propagation paths and increases the likelihood of challenging phenomena like *symptom drift*, where the most severe symptoms manifest far downstream from the actual root cause. Our dataset achieves a maximum depth of **7** (corresponding trace length is 20), creating long and complex causal chains. This is substantially deeper than most other benchmarks, which typically have depths between 2 and 4 (e.g., Nezha-OB: 2, Eadro-TT: 3). The shallow propagation paths in datasets like GAIA (Max Depth of 2) mean the root cause is often coincident with the most obvious symptom, making it trivially identifiable and thus unsuitable for evaluating advanced causal inference capabilities. Furthermore, our curated set of **78 unique metric types** provides the rich, multi-faceted signals necessary for complex diagnosis, balance between the sparse instrumentation of benchmarks like Eadro (7 types) and the potentially overwhelming noise of GAIA (949 types, most of the which are not relevant for RCA).

> Our benchmark is qualitatively superior by design, creating complex diagnostic scenarios by combining high-impact failures, intense workloads, deep system interactions, and rich telemetry.

6.2 RQ5: Performance Evaluation and Failure Mode Analysis

Given the differences in characteristics observed between our benchmark and existing datasets, we first present the overall performance of eleven SOTA RCA models on our new benchmark. From Fig. 6, we see that the distribution of fault types in our benchmark is not balanced, simply inspecting the overall performance metrics can obscure important nuances in how different models handle various fault categories. We then discuss the underlying reasons for their widespread failure, moving beyond simple metrics to a qualitative and quantitative analysis of their core limitations.

6.2.1 Overall Performance. We first evaluated all models on our entire benchmark and found overall low performance: the mean *Top*@1 accuracy across all models is only **0.21**, with the best-performing model, MicroRCA, reaching merely **0.37**. Our heuristic baseline, SimpleRCA, can also achieve only **0.28** *Top*@1 accuracy, further underscoring that the complex failure patterns in our dataset cannot be captured by simple rules and demand more sophisticated approaches.

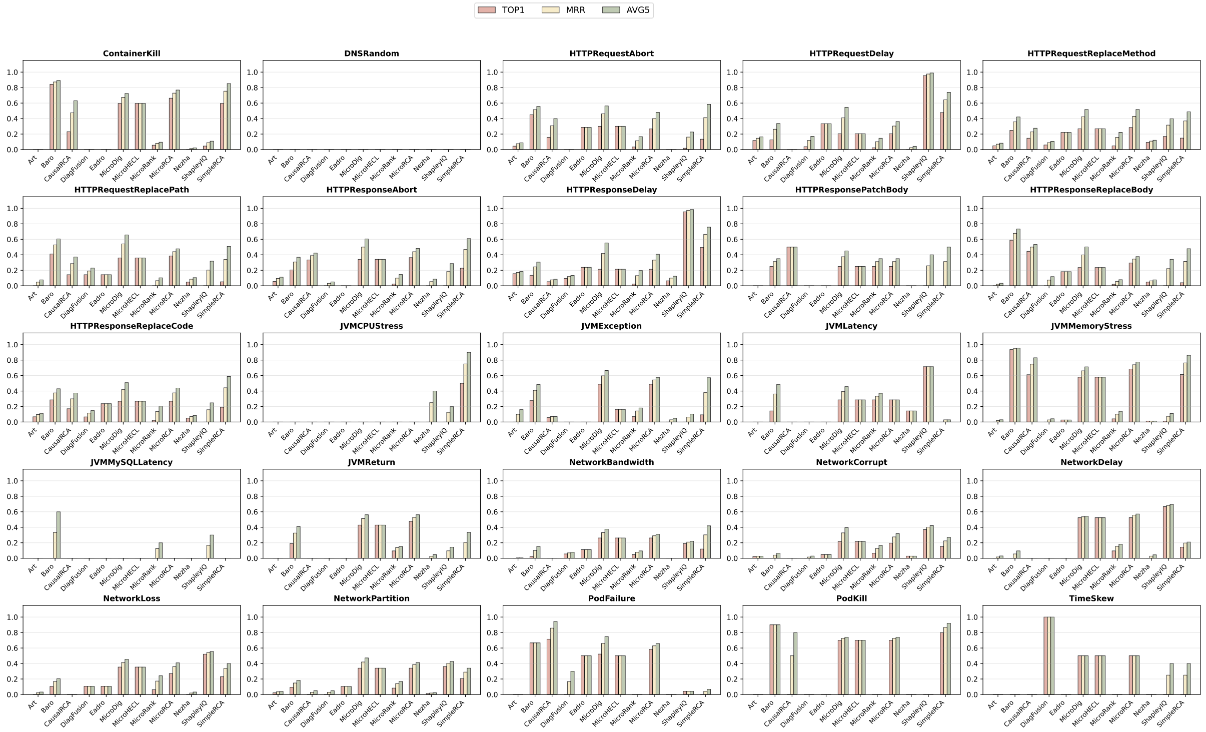

Fig. 8. The performance distribution of each algorithm in terms of fault type, *Top*@1(red), MRR(yellow), *Avg*@5(green), respectively. Most of the algorithms perform poorly on the DNS failure, TimeSkew failure, CPU and Memory stress failure, as well as MySQL related failures.

6.2.2 Performance Breakdown by Fault Type. To understand this performance drop, we first break down the results by fault type, revealing that model effectiveness is highly dependent on the nature of the failure. Fig. 8 presents a detailed breakdown of *Top*@1, MRR, and *Avg*@5 for each model across various fault categories. We choose these three metrics because *Top*@1 reflects the strictest accuracy requirement, MRR provides a balanced view of ranking quality, and *Avg*@5 offers a more stable and holistic view of ranking quality across a broader range of candidates. From the figure, we observe several important patterns: (1) **Performance varies significantly across different fault types, revealing the specific strengths and weaknesses of current RCA models.** For instance, many models demonstrate high proficiency in diagnosing faults related to container lifecycle and resource stress, such as `ContainerKill`, `PodKill`, and `JVMMemoryStress`. In these scenarios, models like Baro, SimpleRCA, and MicroRCA consistently achieve high scores (e.g., Baro's MRR is 0.87 for `ContainerKill` and 0.95 for `JVMMemoryStress`), likely because these faults produce clear, unambiguous signals (e.g., component crashes, resource usage spikes) that are easily captured by telemetry data. (2) **Network-related and semantically complex faults pose a significant challenge to almost all models.** Performance drops sharply for most network faults (`NetworkCorrupt`, `NetworkLoss`, etc.) and logic-related faults (`JVMReturn`). For example, in `NetworkCorrupt` scenarios, even the best-performing model, Shapleyiq, only achieves an MRR of 0.40. This suggests that current models struggle to diagnose problems that are subtle, manifest intermittently, or require a deep understanding of system interactions rather than just simple signal anomalies. (3) **Certain fault types lead to near-total failure across all models.** For faults like `DNSRandom` and `TimeSkew`, almost every model fails completely, with performance metrics at or near zero. This highlights the existence of "blind spots" in current observability and modeling approaches, where the generated telemetry data may not contain sufficient information for diagnosis. Interestingly, Shapleyiq shows exceptional performance on delay-based faults (`HTTPRequestDelay` and `HTTPResponseDelay`), achieving near-perfect scores (MRR > 0.97), indicating its specialized capability in handling latency-induced issues.

6.2.3 Performance Breakdown by Algorithm. Next, we analyze the results from the perspective of algorithmic families, which reveals distinct behavioral patterns in how different approaches rank potential root causes. Fig. 9

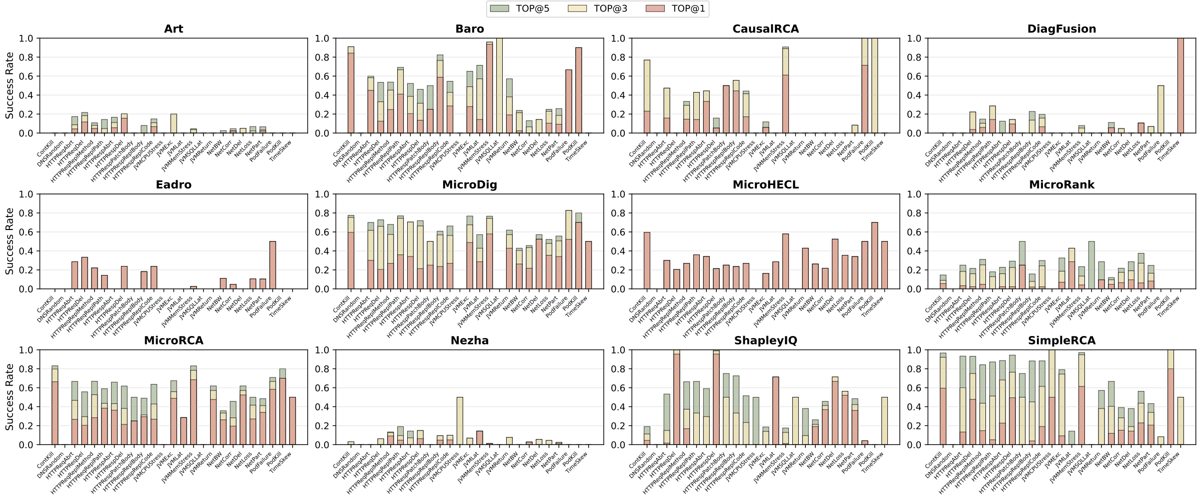

Fig. 9. Failure rate distribution of each algorithm across different fault types. The failure rate is measured at $Top@1$ (red), $Top@3$ (yellow), and $Top@5$ (green) accuracy levels. $Top@1$ shows the highest failure rate due to its strict accuracy requirement, while $Top@5$ demonstrates the lowest failure rate with relaxed criteria. When all bars appear green, it indicates that $Top@5$ and $Top@1$ achieve identical performance.

illustrates the performance distribution of each algorithm in terms of fault type, using $Top@1$ (red), $Top@3$ (yellow), and $Top@5$ (green) accuracy. This perspective reveals distinct behavioral characteristics of different algorithmic approaches. (1) **Some models exhibit an "all-or-nothing" diagnostic behavior.** For algorithms like Eadro, MicroHecl, and to some extent MicroDig and MicroRCA, the bars for $Top@1$, $Top@3$, and $Top@5$ are often of the same height. This indicates that if the model does not identify the correct root cause as its top-ranked candidate, it is highly unlikely to find it in the subsequent four results. This suggests a lack of a nuanced ranking mechanism; the models' features either point directly to the correct cause or miss it entirely. (2) **Other models demonstrate more effective ranking capabilities.** In contrast, models like SimpleRCA, Shapleyiq, and Baro frequently show a significant gap between their $Top@1$ and $Top@5$ scores. For example, in `HTTPRequestAbort` cases, SimpleRCA's $Top@1$ is only 0.13, but its $Top@5$ accuracy jumps to 0.93. This pattern suggests that while these models may not always place the true root cause at the very top, their ranking logic is effective enough to include it within a small set of candidates, which is valuable in practical diagnostic scenarios. (3) **Model performance is highly dependent on the fault category.** No single algorithm excels across all fault types. For example, Baro and SimpleRCA perform exceptionally well on lifecycle faults like `ContainerKill` and `PodKill`, but struggle with network issues. Conversely, Shapleyiq dominates latency-based faults but performs poorly on many others. This lack of a universally superior model underscores that the effectiveness of an RCA algorithm is tightly coupled to its underlying assumptions and the specific characteristics of the failure it is designed to detect.

The performance of SimpleRCA, in particular, serves as a crucial barometer for the complexity of our benchmark. On our new, more challenging benchmark, its performance profile validates our design intentions. The algorithm excels on faults like `ContainerKill` (MRR 0.75) and `JVMMemoryStress` (MRR 0.76) precisely because its design (aggregating simple, threshold-based anomaly counts) is perfectly suited for failures that produce a large volume of obvious signals (e.g., crashes, resource spikes) localized to a single component. However, its effectiveness plummets on network-related and other subtle faults, where the root cause does not necessarily generate the most "noise". **This demonstrates that our benchmark successfully creates complex scenarios where the true cause is decoupled from the most obvious symptoms, a condition that invalidates simple heuristic-based approaches.** Therefore, SimpleRCA's performance pattern confirms that our benchmark introduces the kind of diagnostic ambiguity that necessitates the more sophisticated causal reasoning SOTA models claim to provide.

6.2.4 Failure Mode Distribution. To understand the root causes of this widespread performance degradation, we manually analyzed the 262 most challenging cases where at least 10 of the 12 models failed (including SimpleRCA). Our inductive coding process revealed three primary failure modes. Since a single case could cause different

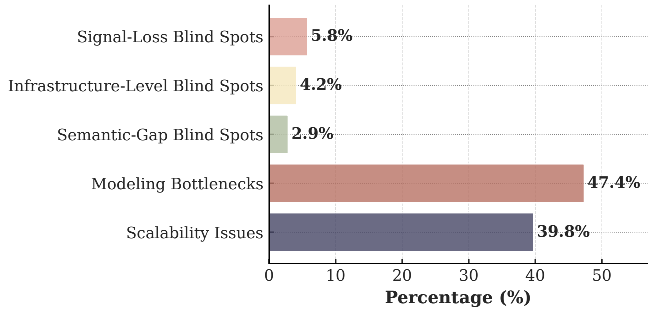

Fig. 10. Distribution of failure causes across SOTA RCA algorithms in failure cases. Failures are categorized into three main types: (1) observability blind spots where telemetry data is missing or incomplete, (2) modeling limitations in capturing system behaviors, and (3) scalability issues that prevent models from handling complex scenarios.

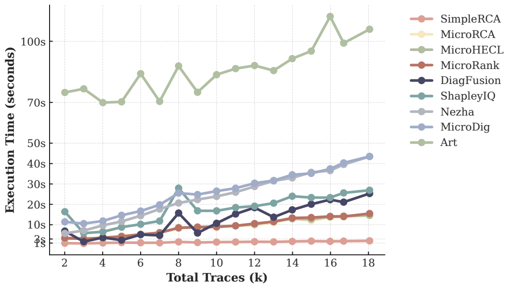

Fig. 11. Scalability analysis of trace/log-based algorithms under 4-core CPU allocation. Most models exhibit at least linear time complexity, limiting real-time applicability.

algorithms to fail for different reasons, we allowed each case to be assigned multiple failure mode labels; the proportions shown in Fig. 10 are calculated from the total sum of these labels. These modes represent a hierarchy of challenges, from practical viability to algorithmic soundness.

The most immediate barrier to adoption is **Scalability Issues**, which render many SOTA models impractical for production environments. Without resource limit, CausalRCA and Nezha need 927.11s and 94.62s on average to process our 8-minute data window (present in Table 5), respectively. In our controlled scalability tests, we constrained all models to a realistic resource limit of 4 CPU cores. As shown in Fig. 11, the execution time of most models increases at least linearly with data volume except for SimpleRCA. The time cost of Eadro and Art is very high, even with a small data volume, since it needs parallelism to prepare the data format required by their deep learning models.

Beyond raw performance, even scalable models are frequently defeated by **Observability Blind Spots**, where necessary diagnostic signals are either absent or uninterpretable. We identified three distinct types: *Signal-Loss Blind Spots*, where models fail to interpret the cessation of telemetry (e.g., from a `PodKill` fault) as a signal; *Infrastructure-Level Blind Spots*, where faults occur in unmonitored components like databases, which we deliberately included to test robustness; and *Semantic-Gap Blind Spots*, where contradictions between telemetry sources (e.g., an error log entry alongside a "success" trace status) create a paradox. These blind spots often compound, creating a web of contradictory signals that defeats models lacking the ability to reason under data uncertainty and incompleteness.

Ultimately, even with perfect scalability and complete observability, models fail due to inherent **Modeling Bottlenecks**, where their core assumptions are invalidated by complex failure patterns. A prime example is frequency-based models like Nezha [44], which assume faults increase event frequency. This assumption is violated by faults like `PodKill` that cause signals to cease, leading the model to misinterpret the absence of events as normal operation. This highlights the most profound limitation: many models' rigid, assumption-laden approaches cannot accommodate the diverse and often subtractive nature of real-world failures, revealing a critical gap in their core diagnostic logic.

7 Discussion

7.1 Threats to Validity

We analyze threats to our study's internal, external, and construct validity and our mitigation strategies.

Internal Validity. This concerns the confidence in our causal conclusions. A key threat is the configuration of baseline models; we mitigated this by using official implementations and prescribed setups. Another threat is potential bugs in our experimental framework. We addressed this through extensive testing and a feedback-guided data curation process, which provided continuous end-to-end validation of our toolchain. Finally, to counter human error in our qualitative failure analysis, we established explicit classification criteria and had three authors independently review and cross-validate the results to ensure consensus.

External Validity. This relates to the generalizability of our findings. Our study relies on a single microservice system, Train-Ticket. Although it is the largest open-source system of its kind, its architecture may not represent all production environments. We mitigated this by injecting 31 diverse fault types and simulating dynamic workloads to capture challenges common to most microservice systems. Our selection of eleven SOTA models may not cover the entire rapidly evolving field. However, our analysis focuses on fundamental failure modes (e.g., Observability Blind Spots) likely shared by entire classes of models, making the findings broadly relevant.

Construct Validity. This addresses whether our metrics accurately measure the intended concepts. Our proxy metrics for benchmark complexity (e.g., QPS, service coverage) may not fully capture diagnostic difficulty. We mitigated this by complementing quantitative metrics with a qualitative, in-depth failure analysis (RQ5). Furthermore, standard evaluation metrics (e.g., Top@K, MRR) do not fully reflect the practical needs of an operator, such as the interpretability of results. We acknowledge this limitation and scope our work to automated localization accuracy, a prerequisite for adoption, noting that full utility evaluation requires human studies.

7.2 Implications for Future Research

7.2.1 Benchmark Design. Our findings call for a significant evolution in benchmark design, moving beyond current limitations. A primary challenge is the **oracle problem**, where reliance on user-facing SLIs creates a flawed ground truth. This approach often mislabels subtle but genuine degradations as "ineffective," which unfairly penalizes advanced models and underestimates their true diagnostic capabilities. To address this, future benchmarks must broaden their **scope of failure** to include critical system-level issues, such as metastable states, rather than focusing only on user-perceivable incidents. Furthermore, evaluation must evolve **beyond diagnostic accuracy**. Standard metrics like *Top@K* accuracy are insufficient because they ignore the practical costs of investigating ranked lists. We advocate for new metrics that measure a model's **diagnostic coherence**, better reflecting its utility in real-world operations.

7.2.2 Model Design. Our failure analysis reveals critical directions for the next generation of RCA models. The evidence shows that a "one-size-fits-all" approach is ineffective, highlighting a clear need for **fault-aware algorithm design** that tailors models to the distinct characteristics of different fault types. In parallel, models must adopt a **holistic system modeling** perspective. Instead of focusing on local, "loudest noise" anomalies, algorithms must be able to reason about cascading effects and systemic dependencies. Finally, the **scalability-efficiency tradeoff** remains a major barrier to adoption. Future work must prioritize lightweight architectures that balance high precision with the near-real-time diagnostic speeds required in production environments.

8 Conclusion

In this paper, we examined the recent progress in data-driven RCA, questioning whether performance on existing benchmarks reflects true capability. Our work suggests that the success of many SOTA models may be closely tied to the characteristics of current public benchmarks. We supported this by analyzing these benchmarks, which revealed common limitations in workload intensity, interaction coverage, and causal complexity. To address this gap, we developed an automated framework to generate a more comprehensive benchmark with dynamic workloads and a wider array of validated fault types. Our re-evaluation of 11 SOTA models on this new benchmark showed that they achieve low *Top@1* accuracies, averaging 0.21 with the best-performing model reaching merely 0.37. This analysis helped us identify three common challenges for current methods: *Scalability Issues*, *Observability Blind Spots*, and *Modeling Bottlenecks*. Our findings suggest that many current RCA methods may face significant hurdles in complex production environments. We propose that the research community could benefit from shifting focus from purely model-centric innovation to a co-design approach that also emphasizes the creation of more realistic and challenging benchmarks. To facilitate this direction, we publicly release our benchmark, generation framework, and evaluation suite, providing a foundation for building the next generation of more robust diagnostic systems.

9 Data Availability

We promise that we will release all the code, baseline implementations, and datasets as open source.

References

[1] [n. d.]. Gaia-dataset. https://github.com/CloudWise-OpenSource/GAIA-DataSet. Accessed: 2024-09-14.

[2] 2024. AIOps Competition Dataset. https://www.aiops.cn/gitlab/aiops-nankai/data/trace/aiops2021. Accessed: 2025-07-01.

[3] 2024. DeathStarBench. https://github.com/delimitrou/DeathStarBench/tree/master. Accessed: 2025-07-01.

[4] 2024. Eadro Dataset. https://zenodo.org/records/7615394. Accessed: 2025-07-01.

[5] 2024. GAIA Dataset. https://github.com/CloudWise-OpenSource/GAIA-DataSet. Accessed: 2025-07-01.

[6] 2024. GitHub - IntelligentDDS/Nezha: The implementation of multimodal observability data root cause analysis approach Nezha in FSE 2023. https://github.com/IntelligentDDS/Nezha. Accessed: 2024-10-27.

[7] 2024. Nezha Dataset. https://github.com/IntelligentDDS/Nezha/tree/main. Accessed: 2025-07-01.

[8] 2024. Online Boutique. https://github.com/GoogleCloudPlatform/microservices-demo. Accessed: 2025-07-01.

[9] 2024. Sock Shop. https://github.com/microservices-demo/microservices-demo. Accessed: 2024-08-21.

[10] 2024. Train Ticket. https://github.com/FudanSELab/train-ticket. Accessed: 2025-07-01.

[11] 2025. Hubble. https://github.com/cilium/hubble. Accessed: 2025-07-01.

[12] 2025. Kube-Prometheus. https://github.com/prometheus-community/helm-charts/tree/main/charts/kube-prometheus-stack. Accessed: 2025-07-01.

[13] 2025. OpenTelemetry Collector. https://opentelemetry.io/docs/collector/. Accessed: 2025-07-01.

[14] Hajar Hameed Addeen. 2019. *A dynamic fault tolerance model for microservices architecture*. South Dakota State University.

[15] CloudZero. 2025. *Cloud Computing Statistics For 2025 And Beyond*. https://www.cloudzero.com/blog/cloud-computing-statistics/ Accessed: 2025-07-01.

[16] Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Xiaomin Wu, Meng Zhang, Qingjun Chen, Xin Gao, Xuedong Gao, et al. 2023. TraceDiag: Adaptive, Interpretable, and Efficient Root Cause Analysis on Large-Scale Microservice Systems. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1762–1773.

[17] Yu Gan, Guiyang Liu, Xin Zhang, Qi Zhou, Jiesheng Wu, and Jiangwei Jiang. 2023. Sleuth: A Trace-Based Root Cause Analysis System for Large-Scale Microservices with Graph Neural Networks. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*. 324–337.

[18] Shenghui Gu, Guoping Rong, Tian Ren, He Zhang, Haifeng Shen, Yongda Yu, Xian Li, Jian Ouyang, and Chunan Chen. 2023. TrinityRCL: Multi-Granular and Code-Level Root Cause Localization Using Multiple Types of Telemetry Data in Microservice Systems. *IEEE Transactions on Software Engineering* 49, 5 (2023), 3071–3088.

[19] Yongqi Han, Qingfeng Du, Ying Huang, Jiaqi Wu, Fulong Tian, and Cheng He. 2024. The Potential of One-Shot Failure Root Cause Analysis: Collaboration of the Large Language Model and Small Classifier. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 931–943.

[20] Zilong He, Pengfei Chen, Yu Luo, Qiuyu Yan, Hongyang Chen, Guangba Yu, and Fangyuan Li. 2022. Graph based incident extraction and diagnosis in large-scale online systems. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–13.

[21] Azam Ikram, Sarthak Chakraborty, Subrata Mitra, Shiv Saini, Saurabh Bagchi, and Murat Kocaoglu. 2022. Root cause analysis of failures in microservices through causal discovery. *Advances in Neural Information Processing Systems* 35 (2022), 31158–31170.

[22] Xinrui Jiang, Yicheng Pan, Meng Ma, and Ping Wang. 2023. Look Deep into the Microservice System Anomaly through Very Sparse Logs. In *Proceedings of the ACM Web Conference 2023*. 2970–2978.

[23] He Kong, Tong Li, Jingguo Ge, Lei Zhang, and Liangxiong Li. 2024. Enhancing fault localization in microservices systems through span-level using graph convolutional networks. *Automated Software Engineering* 31, 2 (2024), 46.

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

[25] Mingjie Li, Zeyan Li, Kanglin Yin, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2022. Causal inference-based root cause analysis for online service systems with intervention recognition. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3230–3240.

[26] Ye Li, Jian Tan, Bin Wu, Xiao He, and Feifei Li. 2023. Shapleyiq: Influence quantification by shapley values for performance debugging of microservices. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*. 287–323.

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

[28] Cheng-Ming Lin, Ching Chang, Wei-Yao Wang, Kuang-Da Wang, and Wen-Chih Peng. 2024. Root cause analysis in microservice using neural granger causal discovery. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 206–213.

[29] Dewei Liu, Chuan He, Xin Peng, Fan Lin, Chenxi Zhang, Shengfang Gong, Ziang Li, Jiayu Ou, and Zheshun Wu. 2021. 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)*. IEEE, 338–347.

[30] Yicheng Pan, Meng Ma, Xinrui Jiang, and Ping Wang. 2021. Faster, deeper, easier: crowdsourcing diagnosis of microservice kernel failure from user space. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 646–657.

[31] Luan Pham, Huong Ha, and Hongyu Zhang. 2024. BARO: Robust Root Cause Analysis for Microservices via Multivariate Bayesian Online Change Point Detection. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 2214–2237.

[32] Luan Pham, Huong Ha, and Hongyu Zhang. 2024. Root Cause Analysis for Microservices based on Causal Inference: How Far Are We?. In *2024 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 706–718.

[33] Luan Pham, Hongyu Zhang, Huong Ha, Flora Salim, and Xiuzhen Zhang. 2025. RCAEval: A Benchmark for Root Cause Analysis of Microservice Systems with Telemetry Data. In *Companion Proceedings of the ACM on Web Conference 2025*. 777–780.

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

[35] Yongqian Sun, Binpeng Shi, Mingyu Mao, Minghua Ma, Sibo Xia, Shenglin Zhang, and Dan Pei. 2024. ART: A Unified Unsupervised Framework for Incident Management in Microservice Systems. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 1183–1194.

[36] Lei Tao, Xianglin Lu, Shenglin Zhang, Jiaqi Luan, Yingke Li, Mingjie Li, Zeyan Li, Qingyang Yu, Hucheng Xie, Ruijie Xu, et al. 2024. Diagnosing performance issues for large-scale microservice systems with heterogeneous graph. *IEEE Transactions on Services Computing* (2024).

[37] Dongjie Wang, Zhengzhang Chen, Yanjie Fu, Yanchi Liu, and Haifeng Chen. 2023. Incremental causal graph learning for online root cause analysis. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2269–2278.

[38] Dongjie Wang, Zhengzhang Chen, Jingchao Ni, Liang Tong, Zheng Wang, Yanjie Fu, and Haifeng Chen. 2023. Interdependent causal networks for root cause localization. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5051–5060.

[39] Yidan Wang, Zhouruixing Zhu, Qiuai Fu, Yuchi Ma, and Pinjia He. 2024. MRCA: Metric-level Root Cause Analysis for Microservices via Multi-Modal Data. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 1057–1068.

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

[41] Zhe Xie, Shenglin Zhang, Yitong Geng, Yao Zhang, Minghua Ma, Xiaohui Nie, Zhenhe Yao, Longlong Xu, Yongqian Sun, Wentao Li, et al. 2024. Microservice Root Cause Analysis With Limited Observability Through Intervention Recognition in the Latent Space. (2024).

[42] Ruyue Xin, Peng Chen, and Zhiming Zhao. 2023. Causalrca: Causal inference based precise fine-grained root cause localization for microservice applications. *Journal of Systems and Software* 203 (2023), 111724.

[43] Guangba Yu, Pengfei Chen, Hongyang Chen, Zijie Guan, Zicheng Huang, Linxiao Jing, Tianjun Weng, Xinmeng Sun, and Xiaoyun Li. 2021. Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments. In *Proceedings of the Web Conference 2021*. 3087–3098.

[44] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. 2023. Nezha: Interpretable Fine-Grained Root Causes Analysis for Microservices on Multi-modal Observability Data. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 553–565.

[45] Qingyang Yu, Changhua Pei, Bowen Hao, Mingjie Li, Zeyan Li, Shenglin Zhang, Xianglin Lu, Rui Wang, Jiaqi Li, Zhenyu Wu, et al. 2023. CMDiagnostor: An Ambiguity-Aware Root Cause Localization Approach Based on Call Metric Data. In *Proceedings of the ACM Web Conference 2023*. 2937–2947.

[46] Chenxi Zhang, Zhen Dong, Xin Peng, Bicheng Zhang, and Miao Chen. 2024. Trace-based Multi-Dimensional Root Cause Localization of Performance Issues in Microservice Systems. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–12.

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

[48] Shenglin Zhang, Sibo Xia, Wenzhao Fan, Binpeng Shi, Xiao Xiong, Zhenyu Zhong, Minghua Ma, Yongqian Sun, and Dan Pei. 2024. Failure Diagnosis in Microservice Systems: A Comprehensive Survey and Analysis. *arXiv preprint arXiv:2407.01710* (2024).

[49] Shenglin Zhang, Yongxin Zhao, Sibo Xia, Shirui Wei, Yongqian Sun, Chenyu Zhao, Shiyu Ma, Junhua Kuang, Bolin Zhu, Lemeng Pan, et al. 2024. No More Data Silos: Unified Microservice Failure Diagnosis with Temporal Knowledge Graph. *IEEE Transactions on Services Computing* (2024).

[50] Shenglin Zhang, Yongxin Zhao, Xiao Xiong, Yongqian Sun, Xiaohui Nie, Jiacheng Zhang, Fenglai Wang, Xian Zheng, Yuzhi Zhang, and Dan Pei. 2024. Illuminating the Gray Zone: Non-intrusive Gray Failure Localization in Server Operating Systems. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 126–137.

[51] Lecheng Zheng, Zhengzhang Chen, Jingrui He, and Haifeng Chen. 2024. MULAN: Multi-modal Causal Structure Learning and Root Cause Analysis for Microservice Systems. In *Proceedings of the ACM on Web Conference 2024*. 4107–4116.

[52] Zhouruixing Zhu, Cheryl Lee, Xiaoying Tang, and Pinjia He. 2024. HeMiRCA: Fine-grained root cause analysis for microservices with heterogeneous data sources. *ACM Transactions on Software Engineering and Methodology* 33, 8 (2024), 1–25.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009