AIOpsArena: Scenario-Oriented Evaluation and Leaderboard for AIOps Algorithms in Microservices

Yongqian Sun*, Jiaju Wang*, Zhengdan Li*, Xiaohui Nie†, Minghua Ma¶, Shenglin Zhang*‖, Yuhe Ji*
Lu Zhang*, Wen Long*, Hengmao Chen§, Yongnan Luo*, Dan Pei‡
*Nankai University,{sunyongqian, zhangsl, lzd}@nankai.edu.cn
{wangjiaju,luzhang,YuheJi,luoyongnan,longwen}@mail.nankai.edu.cn
†CNIC, CAS, xhnie@cnic.cn ¶Microsoft, minghuama@microsoft.com
§BizSeer, chenhengmao@bizseer.com ‡Tsinghua University, peidan@tsinghua.edu.cn
‖Corresponding Author

***Abstract*—AIOps algorithms play a crucial role in the maintenance of microservice systems. Many previous benchmarks' performance leaderboard provides valuable guidance for selecting appropriate algorithms. However, existing AIOps benchmarks mainly utilize offline static datasets to evaluate algorithms. They cannot consistently evaluate the performance of algorithms using real-time datasets, and the operation scenarios for evaluation are static, which is insufficient for effective algorithm selection. To address these issues, we propose an evaluation-consistent and scenario-oriented evaluation framework named *AIOpsArena*. The core idea is to build a live microservice benchmark to generate real-time datasets and consistently simulate the specific operation scenarios on it. *AIOpsArena* supports different leaderboards by selecting specific algorithms and datasets according to the operation scenarios. It also supports the deployment of various types of algorithms, enabling algorithms hot-plugging. At last, we test *AIOpsArena* with typical microservice operation scenarios to demonstrate its efficiency and usability. Platform and a video demonstrating the functioning of *AIOpsArena* is available from https://github.com/AIOpsArena/aiopsarena.**

***Index Terms*—microservice system, AIOps benchmark, algorithm hot-plugging**

I. Introduction

Microservice architecture, which utilizes loosely coupled and independently scalable services, has revolutionized the development and deployment of web applications [1]. This architectural model is widely adopted due to its ability to enhance agility, improve scalability, and facilitate continuous deployment, making it ideal for businesses that need to update and scale components frequently. However, due to the complex structure of microservice systems, the volume of monitoring data has grown exponentially, making it unrealistic to rely solely on manual efforts to detect or identify system failures. Therefore, researchers have utilized three types of data that reflect system states: logs, metrics, and traces, to develop many AIOps algorithms [2]–[6] for the maintenance of the systems.

Microservice management consists of several tasks, including anomaly detection, root cause localization, and failure classification, and *etc*. The selection and evaluation of algorithms for these tasks are crucial. This paper defines an *operation scenario* as the fault orchestration within the dataset. For example, if an evaluator wants to assess the performance of anomaly detection algorithms under network faults, the operation scenario for this evaluation is network faults. Alternatively, if an evaluator wants to evaluate the performance of anomaly detection algorithms in a real production environment, which may encounter various faults, the operation scenario is a combination of different faults.

The current standard process for evaluating an algorithm typically involves collecting data that suits the specific operation scenario, selecting multiple baseline algorithms, reproducing these algorithms, and finally comparing their performance on the dataset using representative evaluation metrics. Among these steps, researchers are compelled to invest significant efforts in complex and time-consuming tasks like data collection, data cleaning, and reproducing baseline algorithms in addition to their primary focus on algorithm evaluation, which significantly diminishes research efficiency. Furthermore, selecting the appropriate algorithm poses certain complexities, as different groups have varying algorithmic needs. For individuals not well-versed in algorithms, determining whether to select supervised or unsupervised methods, deep learning, reinforcement learning, or other approaches is particularly challenging.

Regarding the automation of operations such as data simulation, data collection, and model integration, the *MicroOps* platform proposed in [7] focuses on microservice data simulation and AIOps model development. Researchers can easily deploy microservices on the platform, flexibly construct datasets through load testing and fault simulation, and conduct offline training and cloud-based real-time AIOps model testing. However, the limitation of *MicroOps* lies in the fact that, although it is capable of model development, it lacks systematic evaluation mechanisms for different AIOps tasks after the model is developed. Furthermore, it does not provide unified evaluation metrics for comparing algorithms applied to the same AIOps task. Besides, some benchmark works define the unified evaluation metrics with an evaluation leaderboard, such as *TimeSeriesBench*, a time series anomaly detection benchmark. The issue of *TimeSeriesBench* is that it only uses static datasets and does not conduct evaluations tailored to various operation scenarios in microservices (such as fault

types), limiting its applicability in real-world microservice environments. Therefore, we aim to implement a platform that automates the three stages of data collection, model integration, and model evaluation, and propose *a novel mechanism for consistent scenario-oriented evaluation in operation scenarios.*

Implementing the above evaluation platform faces the following challenges: 1) Maintaining different algorithms on the platform may encounter dependency environment conflicts. 2) Numerous types of operation scenarios make it impractical to generate a separate leaderboard for each scenario. Therefore, a mechanism is required to flexibly generate leaderboards for any operation scenario.

Based on these challenges, we provide the industry with a scenario-oriented, evaluation-consistent evaluation platform *AIOpsArena*, offering targeted and practical evaluation solutions. *AIOpsArena* has two main features: 1) **Algorithm Hot-plugging**: We develop an algorithm hot-plugging function. By deploying algorithms in containers, we achieve one-click deployment and removal, perfectly resolving the environment conflicts. 2) **Multiple Leaderboards**: The user can define different operation scenarios and generate different leaderboards. We build a testbed to automatically inject faults into the microservice system, collect datasets, and provide them for evaluation applications, which eliminates the limitations of operation scenarios of offline datasets and saves manual time costs. Moreover, *AIOpsArena*utilizes a unified multi-modal dataset scheme that can reduce the complexities of data preprocessing, thereby enabling more effective training and evaluation of AIOps algorithms.

The paper's main contributions are as follows:

1) We develop a comprehensive framework *AIOpsArena* [1]. *AIOpsArena* facilitates fault injection, data collection, and the upload and execution of various types of algorithms. It also deploys an evaluation application that can use real-time datasets and select algorithms integrated into the platform for evaluation, then display the evaluation leaderboard.
2) We design a scenario-oriented, evaluation-consistent evaluation solution, which enables consistently evaluating specific operation scenarios thus being more practical.
3) We release datasets collected from *AIOpsArena*[2] that include typical operation scenarios and propose a unified multimodal dataset scheme for microservice systems. Our commitment to maintaining and updating these datasets supports ongoing research and development efforts in the field.

II. Typical AIOps Tasks

A. Anomaly Detection

Anomaly detection aims to identify anomalies, outliers, or events that deviate from the norm. In AIOps, anomaly detection is widely used to detect various abnormal system behaviors. To identify these anomalies, detectors need to utilize different telemetry data, such as metrics, logs, and traces. As a result, anomaly detection can be further categorized into handling one or more specific telemetry data sources, including metric anomaly detection, log anomaly detection, and trace anomaly detection. Additionally, multi-modal anomaly detection techniques can be employed if multiple telemetry data sources are involved in the detection process [8].

B. Root Cause Localization

Existing methodologies predominantly utilize Key Performance Indicators (KPIs), application logs, and distributed traces to analyze the root causes. Some approaches directly process these data sources to identify which services are experiencing anomalies. Others utilize this data to construct graphs or topological models that depict the dependencies among services within the system [9].

C. Failure Classification

This process involves studying the patterns or fingerprints [10] of past failure cases. When a new anomaly is encountered, the system compares it with the learned patterns, thereby categorizing it into a known failure type or identifying it as a new type.

III. System Design

In this section, we introduce the design and implementation of *AIOpsArena*.

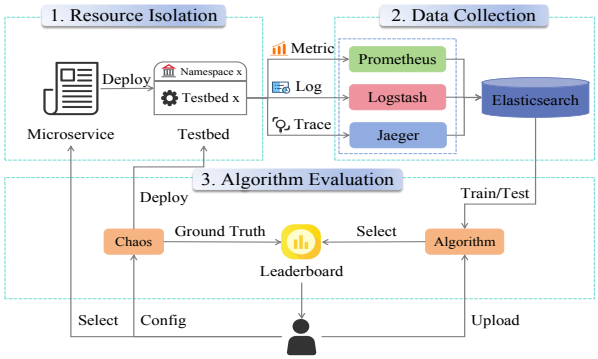

Fig. 1: Architecture of *AIOpsArena*.

A. Resources Isolation

To provide users with an isolated environment, *AIOpsArena* leverages Kubernetes [7] to orchestrate microservices systems and Docker to manage algorithms. To achieve user isolation, *AIOpsArena* identifies users based on their email addresses. All resources belonging to a user, including microservices, data, and algorithms, are associated with their respective email addresses. Users are granted permission solely to access public resources and manage their own resources. *AIOpsArena* provides users with several selectable microservices systems. The microservices deployed by the user, referred to as the testbed, are allocated to a unique namespace, thereby avoiding

[1]The source code is available at https://github.com/AIOpsArena/aiopsarena, and the system is deployed and can be accessed by https://microservo.aiops.cn.
[2]Dataset available at: https://github.com/AIOpsArena/dataset

interference between different users. Additionally, the algorithms used by users run in containers, which inherently provide environmental isolation.

B. Data Collection

Once a user creates a testbed, *AIOpsArena* automatically collects and aggregates multimodal data, including metrics, logs, and traces. We employ Logstash [7], Prometheus [7], and Jaeger [7] to collect system logs, metrics, and microservice traces, respectively. Subsequently, we initiate separate collection services for each testbed, efficiently writing into Elasticsearch, with all data stored in indices named after the corresponding namespaces. To facilitate model evaluation with *AIOpsArena*, we provide an extractor for each data type. These extractors export raw data from the database, preprocess it according to our designed schema, and save it as CSV files. For example, concerning metric data, we can customize the data sampling interval and save the data for all microservice instances according to the metric names.

C. Algorithm Evaluation

There are three modules used to evaluate algorithms: the Chaos Module is used for fault injection to simulate operation scenario, the Algorithm Module is used to integrate algorithms into *AIOpsArena*, while the Evaluation Module is used to select algorithms and evaluate their results.

Algorithm Module. we release a AIOpsArena-SDK[3] that implements a Django framework that user-developed algorithms need to implement RESTful interfaces for the two phases of model training and model testing. After the user submits the SDK, *AIOpsArena* runs the code in a docker container and sends the user request to the SDK. For training request, the SDK use the extractors to pull the specified data from Elasticsearch for the training method to use and stores the trained model; for testing request, the SDK also pulls the data for the testing method and stores the test results for each request. We store all the information of algorithm containers, such as IP address, port, and email address, in the database.

Chaos Module. Current research commonly uses Chaos Mesh [7], ChaosBlade, or ChaosMeta for fault simulation. Considering that Chaos Mesh supports scheduled tasks and can inject various Kubernetes faults, we choose to develop the chaos module based on ChaosMesh. In Chaos Mesh, faults in Kubernetes are defined by custom YAML files. Therefore, to facilitate user interaction, *AIOpsArena* provides a series of YAML templates for common faults. These templates allow users to generate fault instances by filling in basic, easily understandable fields. *AIOpsArena* stores all fault information in MySQL, ready for subsequent algorithm evaluation.

Evaluation Module. We design a consistent and dynamic scenario-oriented evaluation mechanism that assists users in evaluating the best-performing algorithms in specific operation scenarios. Consistency is reflected in the fact that the testbed used for evaluation is not one-time; users can continuously simulate operation scenarios on the testbed and evaluate the performance of different algorithms. Dynamism is demonstrated through real-time data collection and algorithm selection.

Before evaluation, the user needs to create a fault injection plan and inject it into the testbed. After fault injection, the period during which the fault occurred is exported as a dataset, and the algorithms to be evaluated are selected. *AIOpsArena* will run all the selected algorithms using this dataset and aggregate the results into a leaderboard for the user to review.

We provide three common evaluation metrics for anomaly detection algorithms: Point-based, Range-based, Event-based [11], three evaluation metrics for fault classification: *Accuracy@k*, *Avg@k*(Average Accuracy), *MAR*(Mean Average Rank) and three evaluation metrics for root cause localization: *Micro-F1*, *Macro-F1*, *Weighted-F1*.

IV. Dataset Scheme

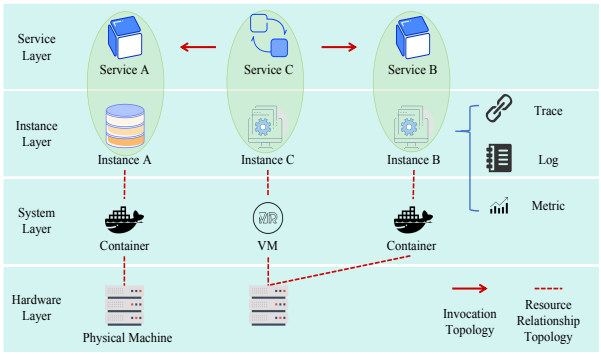

Fig. 2: Observability Dataset Scheme.

The core concept of designing an observable dataset scheme lies in using operational objects and topologies as the framework, integrating multi-modal data such as metrics, logs, and traces to support AIOps tasks. When an IT object's metric triggers an alert, we need to analyze the impact of the fault by tracing upwards and downwards to identify the root cause and retrieving the object's log information for further fault diagnosis. Similarly, when a microservice invocation (trace) experiences latency or failure, we can analyze the key health metrics and contextual log information of the associated object through metrics and logs.

Based on the above concept, we develop an abstracted, observable data model as shown in Fig.2. We construct a vertical layered object model relationship from top to bottom (microservices, instances, systems, virtualization, hardware) according to the deployment context of microservices systems. Horizontally, we build the invocation relationships between services and instances based on APM invocation relationships. Each instance's status information can be monitored through various collection tools, gathering relevant metrics, logs, and trace data. Then, multi-modal data can be organically aggregated based on the instance ID to facilitate AIOps tasks.

[3]We design a template called AIOpsArena-SDK for algorithm integration, available at https://github.com/AIOpsArena/hot-plugging

V. Case Study

In this section, we illustrate the practical application of *AIOpsArena* through a detailed case study. First, we present the minimum overhead required to deploy the entire system. Then, we provide a presentation to a typical user workflow. Subsequently, we conduct an in-depth analysis based on the results obtained from this case demonstration.

A. System Overhead

To deploy this system running on a Linux operating system, a server must be configured with 64GB of RAM, a 32-core CPU, and 150GB of disk space. A stable, high-speed network connection is essential to ensure seamless deployment. Under optimal network conditions, the deployment is expected to take approximately 30 minutes.

B. User Workflow

1) Microservice Deployment: We choose to deploy the Online Boutique integrated by *AIOpsArena* on the *Homepage* page. Once deployed, the user load is injected into the testbed. *AIOpsArena* automatically collects data from the testbed and displays it on the *DataDisplay* page.
2) Algorithm Upload: We implement training and testing interfaces for the Diagfusion algorithm within the AIOpsArena-SDK. Subsequently, we upload the SDK to *AIOpsArena* via the *AlgorithmMarket* page and start the algorithm container.
3) Algorithm Training: On the *AlgorithmMarket* page, we select the Diagfusion algorithm we previously uploaded and train it using eight hours of data collected from the testbed. Since Diagfusion is a multimodal fault diagnosis algorithm, we select three data types: logs, metrics, and trace data, focusing on periods in which we have simulated faults.
4) Fault Simulation: To evaluate the performance of the root cause localization algorithm in operation scenarios characterized by high network latency, we inject multiple network delay faults through the *FaultInjection* page. These faults last 10 minutes each, with varying delays between 2 to 3 seconds.
5) Dataset Collection: After the injected faults conclude, we extract the corresponding time-period data named *NetworkDelay* via the *Dataset* page.
6) Algorithm Evaluation: On the *Leaderboard* page, we initiate an evaluation of root cause localization algorithms. We select the uploaded Diagfusion algorithm and several other algorithms already deployed on *AIOpsArena*, using *NetworkDelay* as the evaluation dataset. We can view the performance of the algorithms on various evaluation metrics after the experiment finishes.

C. Result Analysis

Fig.3 presents the evaluation leaderboard of anomaly detection algorithms including Deeplog [4], USAD [3], TimesNet [2], DLinear [5], and Autoformer [6]. Fig.3a shows the experimental results on the *NetworkDelay* dataset, while Fig.3b displays the outcomes on the *DiskFull* dataset. *NetworkDelay* impacts system operations by increasing network latency between microservices, whereas *DiskFull* affects the system by filling up the disk of a microservice instance. On both datasets, TimesNet shows the best performance due to its capability to effectively convert one-dimensional time series data into two-dimensional data, thereby more efficiently learning features within and across data cycles. Conversely, Deeplog performs the worst on the *NetworkDelay* dataset, as network-related failures are not very apparent in log data, making fault detection ineffective. However, this does not imply that Deeplog's overall performance is poor. In Fig.3b, Deeplog demonstrates a higher F1 score than USAD. This advantage is due to the more apparent manifestations of DiskFull-type failures in logs, which allow Deeplog to utilize its strong log parsing capabilities. Moreover, Deeplog's model, with its powerful pattern recognition ability, can learn key anomaly patterns from vast amounts of log data and adapt to changing log patterns, which is particularly crucial for large systems that handle massive datasets. These experimental results validate the effectiveness of our scenario-oriented evaluation mechanism, where each algorithm has its suitable scenario. We aim to provide more granular evaluations to help researchers and practitioners find the most appropriate algorithms.

VI. Related Work

Previous related work can be categorized as microservice systems, evaluation leaderboard, and AIOps platforms.

A. Evaluation Leaderboard

TimeSeriesBench [11] is a benchmark platform for time series anomaly detection, evaluating the performance of univariate anomaly detection algorithms. TimeSeriesLibrary [12] provides a leaderboard for five time-series-related tasks: long-term and short-term forecasting, imputation, anomaly detection, and anomaly classification, showcasing the top three performing models. Although the datasets used in the above evaluations are representative, they are still limited, and the results only offer a rough estimate of overall performance. We are more focused on algorithms' performance in specific operational scenarios. Therefore, the evaluations provided by *AIOpsArena* are dynamically adjusted based on changing operation scenarios.

B. AIOps Platforms

[13] proposes a multi-dimensional, metric-driven microservice fault simulation system that can create on-demand fault scenarios within customizable service topologies and automatically collect complex performance metrics. [14] introduces a model update and management pipeline framework that continuously trains, packages, and deploys models based on the current system state. [7] proposes a more comprehensive AIOps platform that covers the entire lifecycle of microservice AIOps research, including microservice deployment, data simulation, data collection, and model integration, providing users with consistent and seamless automation support. In

Algotithm Name	Evaluation Result		
	point_based_f1		
	precision	recall	f1_score
deeplog re-evaluate	0.796	0.563	0.660
usad re-evaluate	0.671	0.793	0.720
timesnet re-evaluate	0.770	0.951	0.851
dlinear re-evaluate	0.697	0.817	0.753
autoformer re-evaluate	0.761	0.909	0.829

(a) Run experiments on the *NetworkDelay* dataset.

Algotithm Name	Evaluation Result		
	point_based_f1		
	precision	recall	f1_score
deeplog re-evaluate	0.612	0.858	0.714
usad re-evaluate	0.453	0.855	0.592
timesnet re-evaluate	0.724	0.832	0.774
dlinear re-evaluate	0.776	0.598	0.676
autoformer re-evaluate	0.544	0.899	0.678

(b) Run experiments on the *DiskFull* dataset.

Fig. 3: Evaluation Leaderboard of Anomaly Detection.

comparison, *AIOpsArena* offers comprehensive automation support similar to [7]. However, while [7] focuses on model development, *AIOpsArena* concentrates on the evaluation after model development, effectively reducing the workload of researchers during evaluation and enabling them to focus more on model improvement.

VII. Conclusion and Future Prospects

In this paper, we propose *AIOpsArena*, an algorithm-dynamic and scenario-oriented algorithm evaluation framework. To provide evaluations tailored to specific operation scenarios, *AIOpsArena* automates processes from fault injection, data collection and cleansing, to algorithm evaluation. Additionally, *AIOpsArena* supports deploying various algorithms, enabling one-click deployment and removal. To demonstrate the functionality of scenario-oriented evaluations, we conducted evaluations in typical operation scenarios, presenting the corresponding evaluation leaderboards. In future work, we commit to deploying more state-of-the-art open-source algorithms on the platform. Additionally, we plan to add the capability to upload offline datasets, further enriching the evaluation functionality.

VIII. Acknowledgments

We would like to thank Haiming Zhang, Changhua Pei, and their teams at the Computer Network Information Center of the Chinese Academy of Sciences (CNIC, CAS) for providing the Chinese Science and Technology Cloud platform (CSTCloud). Their support has been vital to the success of our work. This work is supported by the Advanced Research Project of China (No. 31511010501), and the National Natural Science Foundation of China (62272249, 62302244).

References

[1] Nuha Alshuqayran, Nour Ali, and Roger Evans. A systematic mapping study in microservice architecture. In *2016 IEEE 9th international conference on service-oriented computing and applications (SOCA)*, pages 44–51. IEEE, 2016.

[2] Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. Timesnet: Temporal 2d-variation modeling for general time series analysis. In *The eleventh international conference on learning representations*, 2022.

[3] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A Zuluaga. Usad: Unsupervised anomaly detection on multivariate time series. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3395–3404, 2020.

[4] 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*, pages 1285–1298, 2017.

[5] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 11121–11128, 2023.

[6] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in neural information processing systems*, 34:22419–22430, 2021.

[7] Yuewei Li, Zhigang Wang, Qi Qi, Yuhan Jing, Jinming Wu, Zhikang Wu, Yan Lu, Chengsen Wang, Xingyu Wang, and Jingyu Wang. Microops: Rapid microservice data simulation and aiops model development platform. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 52–56. IEEE, 2024.

[8] Qian Cheng, Doyen Sahoo, Amrita Saha, Wenzhuo Yang, Chenghao Liu, Gerald Woo, Manpreet Singh, Silvio Saverese, and Steven CH Hoi. Ai for it operations (aiops) on cloud platforms: Reviews, opportunities and challenges. *arXiv preprint arXiv:2304.04661*, 2023.

[9] Jacopo Soldani and Antonio Brogi. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. *ACM Computing Surveys (CSUR)*, 55(3):1–39, 2022.

[10] 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.

[11] Haotian Si, Changhua Pei, Hang Cui, Jingwen Yang, Yongqian Sun, Shenglin Zhang, Jingjing Li, Haiming Zhang, Jing Han, Dan Pei, et al. Timeseriesbench: An industrial-grade benchmark for time series anomaly detection models. *arXiv preprint arXiv:2402.10802*, 2024.

[12] THUML. Time-series-library. https://github.com/thuml/Time-Series-Library.

[13] Tingzhu Bi, Yicheng Pan, Xinrui Jiang, Meng Ma, and Ping Wang. Vecrosim: A versatile metric-oriented microservice fault simulation system (tools and artifact track). In *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, pages 297–308. IEEE, 2022.

[14] Ruibo Chen, Yanjun Pu, Bowen Shi, and Wenjun Wu. An automatic model management system and its implementation for aiops on microservice platforms. *The Journal of Supercomputing*, 79(10):11410–11426, 2023.