

RuntimeSlicer: Towards Generalizable Unified Runtime State Representation for Failure Management

Lingzhe Zhang
Peking University
Beijing, China
zhang.lingzhe@stu.pku.edu.cn

Tong Jia*
Peking University
Beijing, China
jia.tong@pku.edu.cn

Weijie Hong
Peking University
Beijing, China
hongwj@stu.pku.edu.cn

Mingyu Wang
Peking University
Beijing, China
mingyuwang25@stu.pku.edu.cn

Chiming Duan
Peking University
Beijing, China
duanchiming@stu.pku.edu.cn

Minghua He
Peking University
Beijing, China
hemh2120@stu.pku.edu.cn

Rongqian Wang
Huawei Technologies Co., Ltd.
Beijing, China
wangrongqian2@huawei.com

Xi Peng
Huawei Technologies Co., Ltd.
Hong Kong SAR, China
pancy.pengxi@huawei.com

Meiling Wang
Huawei Technologies Co., Ltd.
Shenzhen, China
wangmeiling17@huawei.com

Gong Zhang
Huawei Technologies Co., Ltd.
Shenzhen, China
nicholas.zhang@huawei.com

Renhai Chen
Huawei Technologies Co., Ltd.
Beijing, China
chenrenhai@huawei.com

Ying Li*
Peking University
Beijing, China
li.ying@pku.edu.cn

ABSTRACT

Modern software systems operate at unprecedented scale and complexity, where effective failure management is critical yet increasingly challenging. Metrics, traces, and logs provide complementary views of system runtime behavior, but existing failure management approaches typically rely on task-oriented pipelines that tightly couple modality-specific preprocessing, representation learning, and downstream models, resulting in limited generalization across tasks and systems. To fill this gap, we propose RuntimeSlicer, a unified runtime state representation model towards generalizable failure management. RuntimeSlicer pre-trains a task-agnostic representation model that directly encodes metrics, traces, and logs into a single, aligned system-state embedding capturing the holistic runtime condition of the system. To train RuntimeSlicer, we introduce Unified Runtime Contrastive Learning, which integrates heterogeneous training data sources and optimizes complementary objectives for cross-modality alignment and temporal consistency. Building upon the learned system-state embeddings, we further propose State-Aware Task-Oriented Tuning, which performs unsupervised partitioning of runtime states and enables state-conditioned adaptation for downstream tasks. This design allows lightweight task-oriented models to be trained on top of the unified embedding without redesigning modality-specific encoders or preprocessing pipelines. Preliminary experiments on the AIOps 2022 dataset demonstrate the feasibility and effectiveness of RuntimeSlicer for system state modeling and failure management tasks.

CCS CONCEPTS

• Software and its engineering → Maintaining software.

KEYWORDS

Failure Management, State Representation, Metric, Trace, Log

1 INTRODUCTION

Modern software systems continue to evolve at an unprecedented scale and level of complexity. From e-commerce platforms and social media services to financial trading systems and cloud-native infrastructures, today’s software systems routinely support billions of users and execute business-critical workloads under highly dynamic conditions.

However, the widespread adoption of microservice architectures, containerized deployments, and elastic scaling mechanisms has significantly increased the dynamism, heterogeneity, and unpredictability of runtime environments. As a result, large-scale software systems experience failures more frequently, leading to substantial operational disruptions and financial losses [5, 30]. According to an ITIC report, unplanned IT downtime in production environments costs large enterprises over one million U.S. dollars per hour on average [12]. These challenges highlight the critical need for timely failure management, including anomaly detection, failure localization and failure classification.

System traces, metrics, and logs capture rich information about the runtime behavior of software systems, recording both the internal states of executing components and the critical events occurring during system operation. As such, they constitute data sources for failure management, providing insights into normal system behavior as well as deviations that may indicate emerging failures.

A large body of prior work has explored failure management techniques based on these runtime data modalities. Metric-based approaches analyze key indicators of system performance and resource utilization—such as latency, throughput, and CPU or memory usage—to detect anomalies or infer potential failure

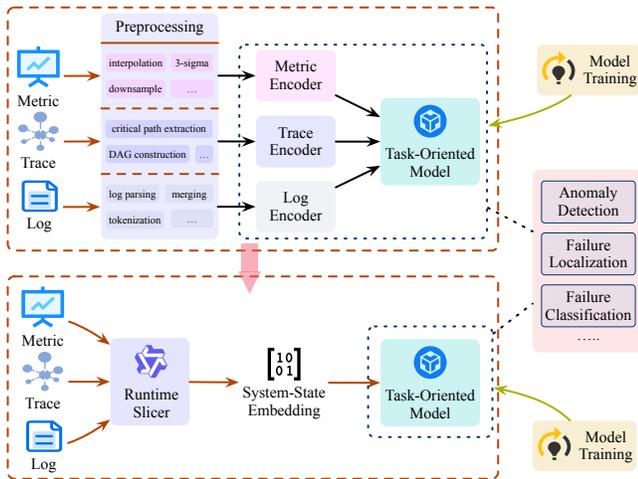


Figure 1: Comparison between conventional failure management training pipelines and the RuntimeSlicer-enabled unified training workflow.

causes [3, 16, 17, 19–21, 25, 38]. Log-based methods leverage system logs as sequences of structured or semi-structured events, combining temporal patterns and semantic information to uncover abnormal behaviors and failure signatures [4, 8–10, 14, 22, 29, 31–35]. Trace-based approaches focus on service invocation relationships and request propagation paths across distributed components, and are commonly applied to failure localization and failure classification tasks in microservice systems [2, 6, 15, 27, 28, 42]. More recently, multimodal approaches attempt to jointly leverage logs, metrics, and traces to improve failure detection and diagnosis by fusing complementary runtime signals from different data sources [11, 13, 18, 23, 24, 26, 36, 37, 39–41].

The effectiveness of existing failure management methods has been demonstrated. However, as illustrated in Figure 1, current approaches still suffer from fundamental limitations. Specifically, most existing methods adopt a task-oriented pipeline: metrics, traces, and logs are first processed through modality-specific preprocessing procedures, followed by separate encoders tailored to each data modality. The resulting representations are then fused and fed into task-specific models, such as anomaly detection, failure localization, or failure classification models. During training, the modality encoders and task-oriented models are optimized jointly in an end-to-end manner. This tightly coupled design causes failure-related patterns to be implicitly entangled with task objectives and encoder architectures, making the learned representations highly dependent on specific tasks, datasets, and system configurations. As a result, the extracted features are difficult to reuse across tasks or environments, leading to limited generalization when system behaviors, workloads, or failure patterns change.

To fill this gap, we propose **RuntimeSlicer**, a unified runtime state representation model towards generalizable failure management. RuntimeSlicer pre-trains a task-agnostic representation model that directly ingests metrics, traces, and logs, and encodes them into a single, aligned system-state embedding capturing the holistic runtime condition of the system.

Unlike conventional task-oriented pipelines that jointly train modality-specific encoders and downstream models, RuntimeSlicer decouples representation learning from failure management tasks. Once trained, RuntimeSlicer can be applied to different systems to produce system-state embeddings, on top of which lightweight task-oriented models—such as anomaly detection, failure localization, and failure classification—can be trained without redesigning preprocessing pipelines or re-training modality encoders.

To train RuntimeSlicer, we introduce **Unified Runtime Contrastive Learning**, a representation learning framework that integrates diverse training data sources, including labeled datasets, runtime collection from live systems, and controlled failure injection. The training objective combines multiple complementary loss functions, including modal consistency loss for cross-modality alignment, temporal consistency loss for preserving runtime continuity, and an optional weak anomaly loss to incorporate coarse-grained failure signals.

Building upon the learned system-state embeddings, we further observe that system states are naturally structured and recurrent, corresponding to distinct runtime conditions such as workload levels, traffic patterns. Motivated by this observation, we propose **State-Aware Task-Oriented Tuning**, which first performs unsupervised state partitioning over system-state embeddings to identify latent runtime states. For each identified state, RuntimeSlicer employs a state-aware adaptation mechanism to train state-conditioned task models, enabling downstream failure management tasks to adapt to heterogeneous runtime conditions.

We conduct preliminary experiments on the AIOps 2022 dataset [1], which is collected from a mature microservices-based e-commerce system, to evaluate the effectiveness of RuntimeSlicer in distinguishing system runtime states and supporting failure management tasks.

2 METHODOLOGY

RuntimeSlicer-enabled failure management is organized into two main stages: system-state embedding generation and state-aware task-oriented tuning. As shown in Figure 2, the left part of the pipeline illustrates the embedding generation process, where metrics, traces, and logs collected within the same time window are jointly fed into a pre-trained RuntimeSlicer model. RuntimeSlicer, trained via Unified Runtime Contrastive Learning, encodes these runtime signals into a compact system-state embedding.

Once the system-state embedding is obtained, it can be used as input to the State-Aware Task-Oriented Tuning stage, which supports downstream failure management models. It is worth noting that the primary goal of this paper is to pre-train a generalizable runtime state representation through RuntimeSlicer. The proposed state-aware task-oriented tuning serves as one possible instantiation demonstrating how the learned embedding can be exploited, while the representation itself is compatible with a wide range of downstream tasks and model designs.

2.1 Unified Runtime Contrastive Learning

Unified Runtime Contrastive Learning is designed to learn a unified and task-agnostic representation of system runtime states from heterogeneous observability data. Unlike task-oriented training

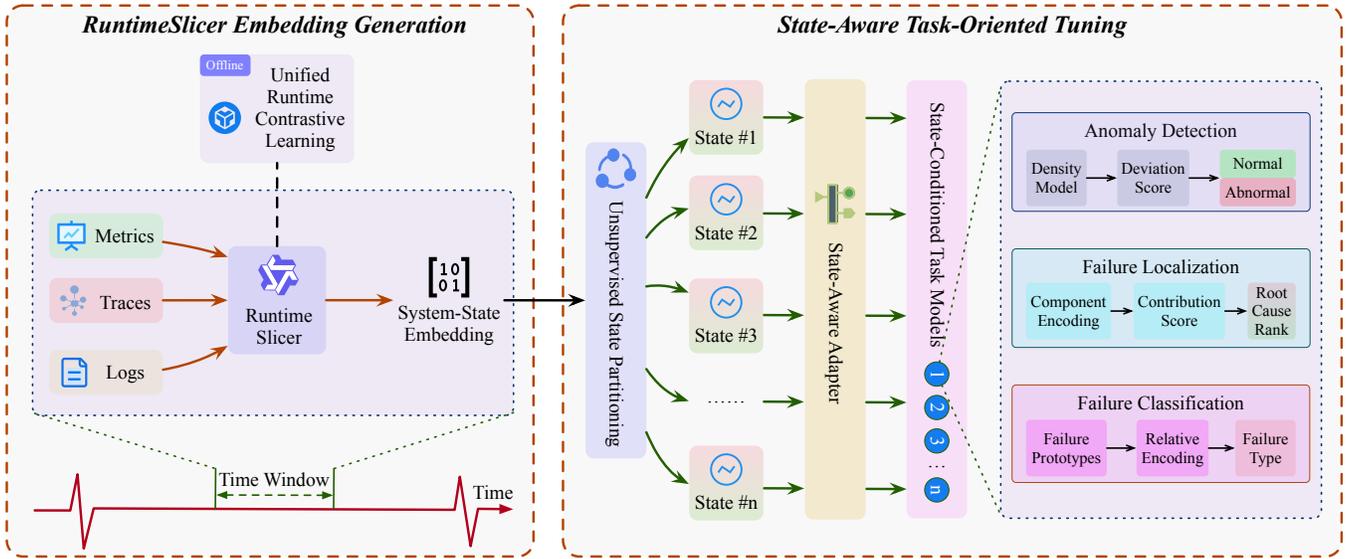


Figure 2: Pipeline of RuntimeSlicer for failure management. The left side illustrates how RuntimeSlicer, trained via Unified Runtime Contrastive Learning, ingests metrics, traces, and logs to produce a system-state embedding. The right side shows how this embedding is leveraged for State-Aware Task-Oriented Tuning of downstream failure management models.

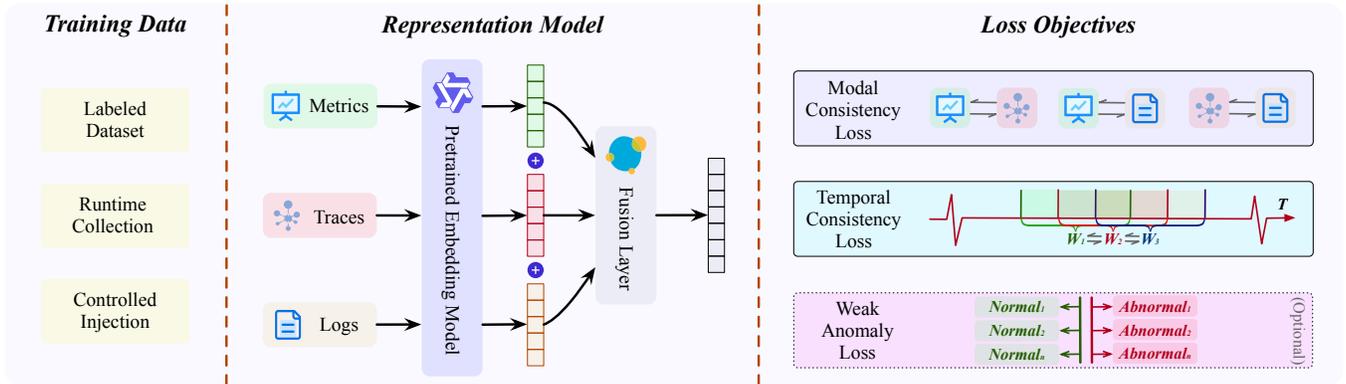


Figure 3: Training Pipeline of Unified Runtime Contrastive Learning

paradigms that entangle representation learning with specific failure management objectives, our goal is to pre-train a representation model that captures the intrinsic structure of runtime behaviors and can be reused across different systems and downstream tasks. As shown in Figure 3, Unified Runtime Contrastive Learning consists of three key components: training data construction, representation model, and loss objectives.

Training Data Construction. To support robust and scalable representation learning, we leverage three complementary sources of training data:

(1) **Labeled Datasets.** We utilize publicly available datasets, such as AIOps 2022 [1] and ART [24], which provide synchronized metrics, traces, and logs along with failure annotations. These datasets offer explicit supervision and serve as a reference for learning failure-aware runtime representations.

(2) **Runtime Collection.** We deploy a set of open-source distributed software systems, including Train-Ticket [43] and Online Boutique [7], under diverse workload configurations. During execution, we continuously collect metrics, traces, and logs generated by the running systems without requiring manual labeling. This data source enables RuntimeSlicer to capture the intrinsic structure and natural variability of system runtime behaviors.

(3) **Controlled Injection.** Building upon the runtime collection setup, we further introduce a limited amount of fault data through controlled failure injection using Chaos Mesh. This data source is intentionally kept small, as learning a unified runtime state representation does not fundamentally rely on explicit anomaly labels. Instead, injected failures are used to provide weak supervisory signals that enhance the model’s ability to distinguish coarse-grained abnormal and normal states.

Representation Model. RuntimeSlicer is instantiated by integrating a shared pre-trained embedding backbone with minimal architectural adaptation for unified multimodal representation. Given a time window t , we denote a multimodal runtime observation as Equation 1.

$$x_t = (x_t^M, x_t^T, x_t^L) \quad (1)$$

In the equation, x_t^M , x_t^T , and x_t^L correspond to metrics, traces, and logs respectively. RuntimeSlicer adopts a unified pre-trained embedding backbone f_θ (Qwen3-Embedding-0.6B in our implementation), shared across modalities, to encode each input into a latent semantic embedding, as illustrated in Equation 2.

$$e_t^M = f_\theta(x_t^M), \quad e_t^T = f_\theta(x_t^T), \quad e_t^L = f_\theta(x_t^L). \quad (2)$$

Unlike conventional architectures that design modality-specific encoders independently, RuntimeSlicer leverages a shared linguistic-statistical embedding prior to ensure that heterogeneous signals are mapped into a unified embedding space, enabling natural comparability and modality-aligned reasoning.

The resulting modality embeddings are aggregated via a lightweight fusion layer g_ϕ , implemented as a shallow multi-layer perceptron (MLP) with optional gating as Equation 3.

$$z_t = g_\phi \left(\text{Concat}(e_t^M, e_t^T, e_t^L) \right) \quad (3)$$

The fusion layer finally produces a compact system-state embedding $z_t \in \mathbb{R}^d$ that summarizes the holistic runtime condition of the system, reflecting workload fluctuations, temporal dependencies, service interactions, and emergent failures.

Loss Objectives. Unified Runtime Contrastive Learning jointly enforces three complementary constraints to shape the system-state embedding space.

(1) Modal Consistency Loss. To ensure that metrics, traces, and logs reflect a coherent system condition within the same time window, we align modality-specific embeddings using an InfoNCE-style contrastive term. For a batch of time windows $\{t_i\}_{i=1}^B$, let z_i^M, z_i^T, z_i^L denote metric-, trace-, and log-level embeddings respectively. The loss aligns each modality pair by treating same-window representations as positives and others as negatives as Equation 4, where $\text{sim}(\cdot)$ denotes cosine similarity and τ is a temperature parameter. Losses for (T, L) and (M, L) pairs are computed analogously.

$$\mathcal{L}_{\text{modal}} = -\frac{1}{B} \sum_{i=1}^B \log \frac{\exp(\text{sim}(z_i^T, z_i^M)/\tau)}{\sum_{j=1}^B \exp(\text{sim}(z_i^T, z_j^M)/\tau)} \quad (4)$$

(2) Temporal Consistency Loss. Runtime states exhibit smooth temporal evolution, where adjacent time windows sharing similar runtime spans are expected to remain semantically close in the embedding space. Let s_i be the state embedding of window t_i , and $\omega_{ij} \in [0, 1]$ denote their temporal-overlap ratio, which softly weights the expected similarity as Equation 5, where δ is a slack margin and Z is a normalization constant. This formulation naturally induces smoothness for temporally coherent runtime periods while avoiding excessively forcing embeddings of unrelated windows to collapse, thereby preserving meaningful separation when temporal evidence suggests divergence.

$$\mathcal{L}_{\text{temp}} = \frac{1}{Z} \sum_{i \neq j} \omega_{ij} \max(\delta - \text{sim}(s_i, s_j), 0) \quad (5)$$

(3) Weak Anomaly Separation Loss (Optional). When coarse anomaly labels exist, we introduce a light constraint to prevent abnormal states from becoming overly similar to normal ones. Let \mathcal{N} and \mathcal{A} denote normal and abnormal sets of embeddings as Equation 6, where γ controls the maximum permitted similarity. Notably, this term does not cluster anomalies, which only avoids accidental entanglement between normal and abnormal states.

$$\mathcal{L}_{\text{anom}} = \mathbb{E}_{s_n \in \mathcal{N}, s_a \in \mathcal{A}} \left[\max(\text{sim}(s_n, s_a) - \gamma, 0) \right] \quad (6)$$

2.2 State-Aware Task-Oriented Tuning

Since RuntimeSlicer fundamentally captures the underlying system state, we further enable downstream failure management to become state-adaptive. Let $\mathcal{S} = \{s_1, \dots, s_N\}$ denote the learned system-state embeddings. We first perform unsupervised state partitioning as shown in Equation 7, identifying latent runtime regimes such as workload tiers, traffic conditions, and resource-pressure states:

$$C = \{C_1, \dots, C_K\}, \quad C_k = \{s_i \in \mathcal{S} \mid \text{assign}(s_i) = k\}. \quad (7)$$

To operationalize such structure, we introduce State-Aware Adapter, a lightweight module that conditions downstream task models on cluster-specific contextual information. For a downstream failure-management task \mathcal{T} , we derive a set of State-Conditioned Task Models $\{\theta_1, \dots, \theta_K\}$, each tuned on samples belonging to one state cluster as Equation 8 which yields state-specialized models while maintaining a shared global backbone.

$$\theta_k = \arg \min_{\theta} \mathbb{E}_{s_i \in C_k} [\mathcal{L}_{\mathcal{T}}(x_i; \theta, s_i)] \quad (8)$$

For different downstream tasks, RuntimeSlicer instantiates the above mechanism as follows: Anomaly Detection—state-conditioned density models estimate deviations from typical behavior to produce anomaly scores; Failure Localization—state-aware component encodings yield contribution scores for root-cause ranking; and Failure Classification—state-specific prototypes enable relative encoding-based classification within each state regime.

3 PRELIMINARY EVALUATION

To evaluate RuntimeSlicer, we conduct a preliminary study on the AIOps 2022 dataset to assess its feasibility.

We first examine the quality of RuntimeSlicer’s learned representations by evaluating its ability to distinguish runtime system states. As illustrated in Figure 4, we compare 2D t-SNE visualizations of runtime-state embeddings generated by Qwen3-Embedding-0.6B versus RuntimeSlicer. The baseline embeddings exhibit no clear structure, with points densely mixed across the space, whereas RuntimeSlicer yields well-separated clusters, indicating that it captures latent system-state patterns more effectively. This suggests that RuntimeSlicer provides state-aware representations that are more suitable for downstream operational tasks.

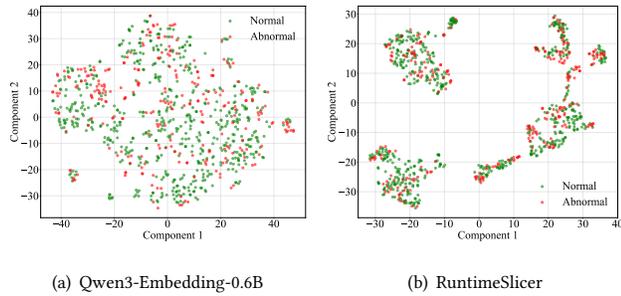


Figure 4: Comparison of runtime-state embeddings generated by Qwen3-Embedding-0.6B and RuntimeSlicer

Next, we evaluate its performance on downstream failure management tasks. We report Precision, Recall, and F1 for all tasks; additionally, for Failure Localization and Failure Diagnosis, we also report Mean Reciprocal Rank (MRR). As shown in Table 1, RuntimeSlicer combined with State-Aware Task-Oriented Tuning achieves promising performance across all three failure-management tasks. However, we observe performance degradation in cases where certain runtime states are underrepresented in the training data. We plan to focus on optimizing performance for such scenarios in future work.

Table 1: Failure Management Results

Task	Precision	Recall	F1-Score	MRR
Anomaly Detection	97.27%	81.18%	88.50%	-
Failure Localization	69.57%	67.33%	68.43%	70.15%
Failure Diagnosis	87.57%	75.12%	80.88%	83.35%

4 CONCLUSION

This paper presents a unified runtime state representation for generalizable failure management, eliminating the need for task-specific encoder training. To this end, we introduce Unified Runtime Contrastive Learning, which jointly leverages heterogeneous labeled and unlabeled data to learn system-state representations, yielding RuntimeSlicer. Based on these embeddings, we further propose a demonstration framework—State-Aware Task-Oriented Tuning—to showcase how RuntimeSlicer supports downstream failure-management tasks. Preliminary experiments verify its feasibility, and future work will focus on improving generalization through large-scale fine-tuning.

ACKNOWLEDGMENT

This work was supported by the Huawei–Peking University Joint Laboratory of Mathematics.

REFERENCES

- [1] 2022. AIOPS 2022 Championship. <https://competition.aiops.cn/>.
- [2] Yang Cai, Biao Han, Jinshu Su, and Xiaoyan Wang. 2021. Tracemodel: An automatic anomaly detection and root cause localization framework for microservice systems. In *2021 17th International Conference on Mobility, Sensing and Networking (MSN)*. IEEE, 512–519.
- [3] Jiayang Dong, Haixu Wu, Haoran Zhang, Li Zhang, Jianmin Wang, and Mingsheng Long. 2024. Simmtm: A simple pre-training framework for masked time-series modeling. *Advances in Neural Information Processing Systems* 36 (2024).
- [4] Chiming Duan, Minghua He, Pei Xiao, Tong Jia, Xin Zhang, Zhewei Zhong, Xiang Luo, Yan Niu, Lingzhe Zhang, Yifan Wu, et al. 2025. LogAction: Consistent Cross-system Anomaly Detection through Logs via Active Domain. *arXiv preprint arXiv:2510.03288* (2025).
- [5] Stephen Elliot. 2014. DevOps and the cost of downtime: Fortune 1000 best practice metrics quantified. *International Data Corporation (IDC)* (2014).
- [6] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. 2021. Sage: practical and scalable ML-driven performance debugging in microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 135–151.
- [7] Google Cloud Platform. 2025. Online Boutique: A Cloud-First Microservices Demo Application. <https://github.com/GoogleCloudPlatform/microservices-demo>. Accessed: October 15, 2025.
- [8] Minghua He, Chiming Duan, Pei Xiao, Tong Jia, Siyu Yu, Lingzhe Zhang, Weijie Hong, Jin Han, Yifan Wu, Ying Li, et al. 2025. United we stand: Towards end-to-end log-based fault diagnosis via interactive multi-task learning. *arXiv preprint arXiv:2509.24364* (2025).
- [9] Minghua He, Tong Jia, Chiming Duan, Pei Xiao, Lingzhe Zhang, Kangjin Wang, Yifan Wu, Ying Li, and Gang Huang. 2025. Walk the talk: Is your log-based software reliability maintenance system really reliable? *arXiv preprint arXiv:2509.24352* (2025).
- [10] Weijie Hong, Yifan Wu, Lingzhe Zhang, Chiming Duan, Pei Xiao, Minghua He, Xixuan Yang, and Ying Li. 2025. CSLParser: A Collaborative Framework Using Small and Large Language Models for Log Parsing. In *2025 IEEE 36th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 61–72.
- [11] Xiaosong Huang, Hongyi Liu, Yifan Wu, Lingzhe Zhang, Tong Jia, Ying Li, and Zhonghai Wu. 2025. UDA-RCL: Unsupervised Domain Adaptation for Microservice Root Cause Localization Utilizing Multimodal Data. *IEEE Transactions on Services Computing* (2025).
- [12] Information Technology Intelligence Consulting (ITIC). 2024. *ITIC 2024 Global Server Hardware, Server OS Reliability Report*. Annual Report. ITIC.
- [13] 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.
- [14] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. 2022. Swiss-Log: Robust anomaly detection and localization for interleaved unstructured logs. *IEEE Transactions on Dependable and Secure Computing* 20, 4 (2022), 2762–2780.
- [15] Yufeng Li, Guangba Yu, Pengfei Chen, Chuanfu Zhang, and Zibin Zheng. 2022. MicroSketch: Lightweight and adaptive sketch based performance issue detection and localization in microservice systems. In *International Conference on Service-Oriented Computing*. Springer, 219–236.
- [16] Wenlong Liao, Shouxiang Wang, Dechang Yang, Zhe Yang, Jiannong Fang, Christian Rehtanz, and Fernando Porté-Agel. 2025. TimeGPT in load forecasting: A large time series model perspective. *Applied Energy* 379 (2025), 124973.
- [17] 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.
- [18] Hongyi Liu, Xiaosong Huang, Mengxi Jia, Tong Jia, Jing Han, Zhonghai Wu, and Ying Li. 2024. Uac-ad: Unsupervised adversarial contrastive learning for anomaly detection on multi-modal data in microservice systems. *IEEE Transactions on Services Computing* 17, 6 (2024), 3887–3900.
- [19] Hongyi Liu, Xiaosong Huang, Mengxi Jia, Lingzhe Zhang, Tong Jia, Zhonghai Wu, and Ying Li. 2025. AAAD: Asynchronous Inter-Variable Relationship-Aware Anomaly Detection for Multivariate Time Series. In *2025 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 1–6.
- [20] Hongyi Liu, Yinpeng Ma, Xiaosong Huang, Lingzhe Zhang, Tong Jia, and Ying Li. 2025. ORA: Job Runtime Prediction for High-Performance Computing Platforms Using the Online Retrieval-Augmented Language Model. In *Proceedings of the 39th ACM International Conference on Supercomputing*. 884–894.
- [21] Kashif Rasul, Arjun Ashok, Andrew Robert Williams, Arian Khorasani, George Adamopoulos, Rishika Bhagwatkar, Marin Biloš, Hena Ghonia, Nadhir Hassen, Anderson Schneider, et al. 2023. Lag-llama: Towards foundation models for time series forecasting. In *R0-FoMo: Robustness of Few-shot and Zero-shot Learning in Large Foundation Models*.
- [22] Yicheng Sui, Yuzhe Zhang, Jianjun Sun, Ting Xu, Shenglin Zhang, Zhengdan Li, Yongqian Sun, Fangrui Guo, Junyu Shen, Yuzhi Zhang, et al. 2023. Logkg: Log failure diagnosis through knowledge graph. *IEEE Transactions on Services Computing* 16, 5 (2023), 3493–3507.
- [23] 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.
- [24] Yongqian Sun, Binpeng Shi, Mingyu Mao, Minghua Ma, Sibao 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.

- [25] 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.
- [26] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. 2023. Nezza: 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.
- [27] Guangba Yu, Zicheng Huang, and Pengfei Chen. 2023. TraceRank: Abnormal service localization with dis-aggregated end-to-end tracing data in cloud native systems. *Journal of Software: Evolution and Process* 35, 10 (2023), e2413.
- [28] 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.
- [29] Lingzhe Zhang, Tong Jia, Mengxi Jia, Ying Li, Yong Yang, and Zhonghai Wu. 2024. Multivariate log-based anomaly detection for distributed database. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4256–4267.
- [30] Lingzhe Zhang, Tong Jia, Mengxi Jia, Yifan Wu, Aiwei Liu, Yong Yang, Zhonghai Wu, Xuming Hu, Philip Yu, and Ying Li. 2025. A Survey of AIOps in the Era of Large Language Models. *Comput. Surveys* (2025).
- [31] Lingzhe Zhang, Tong Jia, Mengxi Jia, Yifan Wu, Hongyi Liu, and Ying Li. 2025. ScalaLog: Scalable Log-Based Failure Diagnosis Using LLM. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 1–5.
- [32] Lingzhe Zhang, Tong Jia, Mengxi Jia, Yifan Wu, Hongyi Liu, and Ying Li. 2025. XRAGLog: A Resource-Efficient and Context-Aware Log-Based Anomaly Detection Method Using Retrieval-Augmented Generation. In *AAAI 2025 Workshop on Preventing and Detecting LLM Misinformation (PDLM)*.
- [33] Lingzhe Zhang, Tong Jia, Xinyu Tan, Xiangdong Huang, Mengxi Jia, Hongyi Liu, Zhonghai Wu, and Ying Li. 2025. E-log: Fine-grained elastic log-based anomaly detection and diagnosis for databases. *IEEE Transactions on Services Computing* (2025).
- [34] Lingzhe Zhang, Tong Jia, Kangjin Wang, Weijie Hong, Chiming Duan, Minghua He, and Ying Li. 2025. Adaptive root cause localization for microservice systems with multi-agent recursion-of-thought. *arXiv preprint arXiv:2508.20370* (2025).
- [35] Lingzhe Zhang, Tong Jia, Kangjin Wang, Mengxi Jia, Yong Yang, and Ying Li. 2024. Reducing events to augment log-based anomaly detection models: An empirical study. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 538–548.
- [36] Lingzhe Zhang, Tong Jia, Yunpeng Zhai, Leyi Pan, Chiming Duan, Minghua He, Mengxi Jia, and Ying Li. 2026. Agentic Memory Enhanced Recursive Reasoning for Root Cause Localization in Microservices. *arXiv preprint arXiv:2601.02732* (2026).
- [37] Lingzhe Zhang, Tong Jia, Yunpeng Zhai, Leyi Pan, Chiming Duan, Minghua He, Pei Xiao, and Ying Li. 2026. Hypothesize-Then-Verify: Speculative Root Cause Analysis for Microservices with Pathwise Parallelism. *arXiv preprint arXiv:2601.02736* (2026).
- [38] Lingzhe Zhang, Yunpeng Zhai, Tong Jia, Chiming Duan, Minghua He, Leyi Pan, Zhaoyang Liu, Bolin Ding, and Ying Li. 2025. MicroRemed: Benchmarking LLMs in Microservices Remediation. *arXiv preprint arXiv:2511.01166* (2025).
- [39] Lingzhe Zhang, Yunpeng Zhai, Tong Jia, Chiming Duan, Siyu Yu, Jinyang Gao, Bolin Ding, Zhonghai Wu, and Ying Li. 2025. ThinkFL: Self-Refining Failure Localization for Microservice Systems via Reinforcement Fine-Tuning. *arXiv preprint arXiv:2504.18776* (2025).
- [40] Lingzhe Zhang, Yunpeng Zhai, Tong Jia, Xiaosong Huang, Chiming Duan, and Ying Li. 2025. Agentfm: Role-aware failure management for distributed databases with llm-driven multi-agents. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. 525–529.
- [41] Shenglin Zhang, Pengxiang Jin, Zihan Lin, Yongqian Sun, Bicheng Zhang, Sibao 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.
- [42] Zhizhou Zhang, Murali Krishna Ramanathan, Prithvi Raj, Abhishek Parwal, Timothy Sherwood, and Milind Chabbi. 2022. CRISP: Critical path analysis of Large-Scale microservice architectures. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 655–672.
- [43] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. 2018. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Transactions on Software Engineering* 47, 2 (2018), 243–260.