Intelligent Monitoring Metrics-Based Reliability Management for Large-Scale Cloud Systems

GU, Wenwei

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Doctor of Philosophy
in
Computer Science and Engineering

The Chinese University of Hong Kong
July 2025

Thesis Assessment Committee

Professor KING Kuo Chin Irwin (Chair)

Professor LYU Rung Tsong Michael (Thesis Supervisor)

Professor XU Qiang (Committee Member)

Professor PEI Dan (External Examiner)

Abstract of thesis entitled:

Intelligent Monitoring Metrics-Based Reliability Management for Large-Scale Cloud Systems
Submitted by GU, Wenwei
for the degree of Doctor of Philosophy
at The Chinese University of Hong Kong in July 2025

In recent years, many traditional software systems have been migrated to cloud computing platforms. They serve hundreds of millions of users around the world on a 24/7 basis. Due to their increasing scale and complexity, performance issues become inevitable, which may lead to potential violations of Service Level Agreements (SLAs), causing user dissatisfaction and financial losses. Thus, ensuring the reliability of these cloud systems is paramount. A common practice of reliability engineering involves the gathering of system monitoring metrics, also known as Key Performance Indicators (KPIs), from software and hardware components. The collected data are then analyzed to identify any potential performance issues and root causes. Despite the significance of reliability management in ensuring the smooth operation of cloud systems, challenges such as the large volume of data, data noise, and high rates of false positives significantly hinder effective management. To address these challenges, this thesis explores intelligent reliability management for cloud systems through data-driven approaches. Our research is driven by the monitoring metrics collected from real-world large-scale cloud systems. Our contributions are as follows:

Firstly, we propose ISOLATE, an anomaly detection approach aimed at pinpointing performance issues in cloud systems. Unlike

traditional methods that monitor key performance indicators (KPIs) independently, ISOLATE considers both the relational and temporal dimensions of KPIs, utilizing a graph neural network with attention mechanisms to explore the intricate dependencies among cloud components. This method captures both long-term trends and multi-scale temporal patterns through a combination of GRU and convolution networks, effectively identifying correlation-violated metrics. To further enhance robustness against noisy data, ISOLATE employs a positive unlabeled learning strategy, utilizing pseudo labels generated from a subset of verified negative examples.

Secondly, we develop ADAMAS, an innovative AutoML-based anomaly detection framework tailored for practical application in production cloud systems. Unlike conventional methods that primarily rely on singular service metrics, ADAMAS dynamically adjusts to the complexities of cross-service anomalies through a novel unsupervised evaluation function. This function aids in the automated optimization of model structures and parameter settings, improving anomaly detection capabilities. Furthermore, ADAMAS integrates a lightweight human-in-the-loop mechanism, enabling the incorporation of expert knowledge to refine anomaly detection continually and effectively bridge the gap between detected anomalies and genuine business impacts. Additionally, ADAMAS monitors the frequency of false predictions, allowing for proactive model reconfiguration and fostering a perpetual cycle of system enhancement.

Thirdly, we introduce KPIRoot, an innovative method designed for root cause localization in cloud systems that efficiently integrates both similarity and causality analyses. KPIRoot addresses the limitations of existing methods that typically focus only on the similarity of anomalous trends among key performance indicators (KPIs) to diagnose service quality issues. In complex cloud environments characterized by interdependent services, this approach often proves inadequate. KPIRoot enhances the root cause analysis by not only

measuring the trend alignment of KPIs but also considering the sequential order of their variations to establish causality. Additionally, the method incorporates symbolic aggregate approximation to create a compact representation of each KPI, significantly improving the efficiency of the analysis.

In summary, this thesis targets improving the reliability management of large-scale cloud systems with data analysis on monitoring metrics. Extensive experiments on both public and industrial datasets collected from real-world cloud systems validate the effectiveness and efficiency of our proposed algorithms.

論文題目：大規模雲端系統基於監控指標的智慧可靠性管理

作者　　：辜文蔚
學校　　：香港中文大學
學系　　：計算機科學與工程學系

修讀學位：哲學博士

摘要　　：

近年來，許多傳統軟體系統已經遷移到雲計算平台，這些平台為全球數億用戶提供7天24小時的服務。由於規模和複雜度的增加，性能問題不可避免，可能導致違反服務水平協議（SLA），造成用戶不滿和經濟損失。因此，確保這些雲系統的可靠性至關重要。可靠性工程的一個常見做法是從軟硬體組件中收集系統監控指標（KPI），然後分析數據以識別潛在的性能問題和根本原因。然而，數據量大、數據噪音以及高誤報率等挑戰嚴重阻礙了有效管理。為了解決這些挑戰，本論文探討了通過數據驅動的方法進行的智能可靠性管理。研究基於從現實世界大規模雲系統中收集的監控指標，做出以下貢獻：

首先，我們提出ISOLATE，一種針對雲系統性能問題的異常檢測方法。不同於傳統獨立監控KPI的方法，ISOLATE考慮了KPI的關係和時間維度，利用圖神經網絡和注意力機制探索雲組件間的複雜依賴關係。此方法通過GRU和卷積網絡的結合捕捉長期趨勢和多尺度時間模式，有效識別違反相關性的指標。為增强對噪音數據的魯棒性，ISOLATE採用Positive Unlabeled Learning策略，利用經驗證的負例生成的偽標籤。

其次，我們開發了ADAMAS，一個創新的基於自動機器學習（AutoML）的異常檢測框架，以針對大規模的雲系統。不同於傳統方法，ADAMAS能夠動態調整以應對跨服務異常的複雜性，通過新穎的無監督評估函數自動優化模型結構和參數設置，提高異常檢測能力。此外，ADAMAS整合了一個輕量級人機互動機制，融入專家知識來持續改進異常檢測，並有效彌合檢測到的異常與實際業務影響之間的差距。ADAMAS還監控誤報頻率，允許主動的模型重配置，促進系統的持續改進。

第三，我們引入KPIRoot，一種創新方法，設計用於雲系統中根因定位，並有效整合相似性和因果分析。KPIRoot不僅測量KPI的趨勢對齊，還考慮其變化的順序以建立因果關係。此外，該方法採用符號聚合近似法，創建每個KPI的緊湊表示，顯著提高分析效率。

總之，本論文旨在通過監控指標的數據分析改善大規模雲系統的可靠性管理。基於現實世界雲系統收集的公開和工業數據集的大量實驗驗證了我們提出算法的有效性和效率。

Acknowledgement

Firstly and foremost, I am profoundly grateful to my advisor, Professor Michael R. Lyu, whose guidance and encouragement have been instrumental in my academic journey. One of the most impactful experiences was during my qualifying exam, where my performance was less than stellar. I was nervous, my responses were halting, and I had not published any papers then. Michael reassured me, suggesting that significant progress could be made in the next two years if I focused on several key areas. He advised me to practice English daily and engage more with group members to aim for publishing in top-tier conferences. Following his advice, I worked diligently, and even after my first submission to ASE 2023 was rejected, Michael remained optimistic. He told me that my paper had merit and that it was merely a matter of time and luck before it found the right venue. His positivity and approach to mentoring deeply inspired me and sparked my interest in pursuing a career in academia as a university professor. Michael's spirit will continue to influence my academic career.

Secondly, I wish to express my deep appreciation to my thesis committee members, Prof. Irwin King and Prof. Qiang Xu, for their insightful comments and invaluable suggestions throughout the development of this thesis and during my term presentations. I am also profoundly grateful to Prof. Dan Pei from Tsinghua University for graciously serving as the external examiner for my thesis.

Thirdly, I was fortunate to receive invaluable guidance from Dr. Jiazhen Gu. He constantly emphasized the importance of logical

thinking in writing. When working on papers, he supervised me in organizing my ideas and preparing a clear and logical presentation before starting to write. He advocated for a thoughtful approach, encouraging me to spend less time on repetitive tasks and more on reading papers and thinking deeply about real-world problems and solutions. His motto, "plan thoroughly before taking action," has profoundly influenced me, highlighting the importance of practical impact beyond just publication. Under his mentorship, I achieved significant progress. His high standards in logic and writing, along with his focus on the essence of problem-solving, have left a lasting impact on my work and mindset. As my mentor at Huawei, he greatly enriched my research experience.

Fourthly, I am also fortunate to have worked alongside many exceptional research fellows and group members. I am grateful for the collaboration and significant contributions of Zhuangbin Chen, Jianping Zhang, Yun Peng, Jinyang Liu, Yintong Huo, Renyi Zhong, Yichen Li, Zhihan Jiang, Yizhan Huang and Jinxi Kuang. Their cooperation on various research projects and their insightful suggestions have greatly enriched the content of this thesis.

Finally, I thank my parents for their unwavering support and encouragement throughout my studies.

To my family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

In recent years, cloud computing has become increasingly popular. Many software services are deployed on large-scale cloud systems, such as Microsoft Azure, Google Cloud Platform, and Amazon Web Services [239]. Cloud computing has benefited many enterprises, and its marketplace has been growing significantly. The global cloud computing market is expected to reach $832.1 billion by 2025, representing a compound annual growth rate of 17.5% [184]. These cloud computing systems serve millions of users worldwide on a 24/7 basis, offering a wide range of services [49, 51, 125]. However, due to its growing scale and complexity, performance issues, even failures, are inevitable [83] in cloud systems. Performance issues may degrade overall availability and increase service response times, causing SLA (Service Level Agreement) violations. While hardware and software failures can cause service interruptions [64]. Both the performance issues and failures can result in substantial economic losses and user dissatisfaction [5, 109]. For example, a failure that takes a top cloud provider offline in the US for 3 to 6 days would result in $15 billion of economic loss [204]. Therefore, intelligent reliability management that promptly identifies, diagnoses, and resolves these issues has become a key selling point for cloud service providers to deliver services to their customers.

Leading cloud providers integrate a wide array of monitoring tools to continuously track the health condition of their cloud systems. Different types of monitoring data are generated to reflect the system state from different perspectives, including monitoring metrics [25, 79, 127], logs [60, 61, 66, 241, 254], traces [23, 63, 75, 253], and alerts [100, 114, 244]. Especially, in this thesis, we mainly focus on monitoring metrics-based reliability engineering in cloud systems. By analyzing this data, cloud providers can quickly identify and diagnose potential issues, address them, and maintain high service availability levels through mitigating strategies.

Site reliability engineers (SREs) typically use basic tools to analyze observability data [226], identify performance anomalies, determine the root cause, and diagnose the issue when unexpected interruptions or service downgrades occur. For instance, imagine a global streaming service hosted on Google Cloud Platform encountering intermittent connectivity issues. Users report difficulty accessing content, leading engineers to investigate the root cause. By reviewing network logs and monitoring metrics in dashboards, they detect unusual latency spikes in data transmission between geographically dispersed data centers. Further analysis of the logs reveals that a recent update to the content delivery network (CDN) configuration inadvertently increased the load on certain network paths. Engineers quickly reconfigure the CDN settings to redistribute traffic more evenly, restoring optimal service performance and ensuring seamless streaming for users worldwide. Analyzing monitoring metrics, logs, traces, and alerts is critical for proactively identifying and diagnosing performance issues in cloud systems. However, such manual practice has two major limitations. Firstly, a cloud system is typically vast in scale and consists of tremendous software and hardware components (*e.g.*, microservices, virtual machines, and servers) [56, 158, 225]. Each component may have tens of metrics to be collected in the backend monitoring system, resulting in a large volume of monitoring metrics [154, 206]. Analyzing

large amounts of data manually is labor-intensive and prone to error [26]. Secondly, effective analysis necessitates comprehensive domain knowledge of cloud infrastructure and applications. This requirement poses a significant challenge due to the rapid evolution and increasing complexity of cloud technologies. As these systems grow and change, keeping up to date with the latest configurations, architectures, and potential failure points becomes increasingly difficult for SREs. This knowledge gap can lead to delays in diagnosing issues and implementing timely solutions. Thus, automatic approaches that help SREs identify and diagnose performance issues are preferred over human-dependent IT operations.

The large amounts of data generated by cloud systems support the data-driven solution, leveraging AI techniques (machine learning, deep learning, and even large language models) to help with the system maintenance and operations, also known as AIOps. Nevertheless, it is still challenging to develop these solutions in real-world, large-scale cloud systems. We summarize these challenges as follows:

- **Complicated Correlation Between Components**: Cloud service system typically consists of various components (*e.g.*, storage, computing, middleware). These components often have intricate dependencies, making it difficult to identify the issues and discover the root cause accurately. Developing models that can accurately capture and analyze this correlation is a non-trivial task.

- **Gap between the technical and business interpretation of anomalies**: The challenge of discerning true business anomalies that cause service interruption from false positives that manifest as outliers in monitoring data is significant. Not all anomaly patterns manifesting in monitoring metrics indicate genuine performance issues. Normal fluctuations due to expected behaviors, such as auto-scaling responses, should not be flagged as

performance anomalies. These false positives can lead to meaningless troubleshooting efforts if not properly handled. The gap between true anomalies and false anomalies also poses challenges in model training, as not all anomalies detected in monitoring metrics are true performance issues.

- **Evolving Data with Concept Drift**: Cloud technologies and infrastructures are continually advancing, with frequent updates, new features, and configuration changes that lead to concept drift within monitoring data. This rapid evolution poses a significant challenge for maintaining and updating performance anomaly detection models. Developing models that adapt to new patterns and behaviors within the system and effectively handle concept drift with minimal human intervention is a complex and demanding task.
- **Diversity in Data Anomaly Patterns**: Modern service technologies, such as microservices and serverless functions, decouple software into sophisticated and fine-grained units, leading to great diversity and dynamism not only in functionalities but also in anomaly patterns. According to the No Free Lunch Theorem, it is challenging for a single method to effectively identify the entire spectrum of anomalies across different services. For instance, while a common approach to anomaly detection involves learning the normal pattern of a metric time series and identifying deviations, this may not be effective for anomalies that deviate in ways that do not prominently violate time series periodicity.
- **Real-time Processing with Large Volume of Data**: Cloud environments often encompass thousands of servers, virtual machines, and numerous applications running simultaneously. Managing and processing the sheer volume of data generated in such large-scale systems requires significant computational resources. The vast volume of underlying monitoring data and

the tight pressure to resolve issues quickly necessitate the design of efficient solutions to process large amounts of data efficiently, often within seconds.

To address these challenges, we conduct research on intelligent reliability management for cloud systems with data-driven approaches. The overall pipeline of our work is illustrated in Figure 1.1. Our studies are driven by the monitoring metric data collected from the backend monitoring system. Firstly, we propose an approach ISOLATE aimed at pinpointing performance issues with multivariate monitoring metrics collected from cloud systems. It considers both the relational and temporal information of monitoring metrics, utilizes a graph neural network with attention mechanisms to explore the intricate dependencies among cloud components, and captures both long-term trends and multi-scale temporal patterns through a combination of GRU and convolution networks. It further enhances robustness against noisy data through a positive unlabeled learning strategy. Secondly, we develop ADAMAS, an AutoML-based anomaly detection framework that dynamically adjusts to the complexities of cross-service anomalies through a novel unsupervised evaluation function. Furthermore, ADAMAS integrates a lightweight human-in-the-loop mechanism, enabling the incorporation of expert knowledge to refine anomaly detection continually and effectively bridge the gap between detected anomalies and genuine business impacts. Finally, we introduce KPIRoot, an effective and efficient framework for root cause analysis in cloud systems with monitoring metrics. It integrates the strength of similarity analysis and causality analysis. Additionally, using the SAX representation of KPI significantly improves the method's efficiency.

1.2 Thesis Contributions

The contributions of this thesis are summarized as follows:

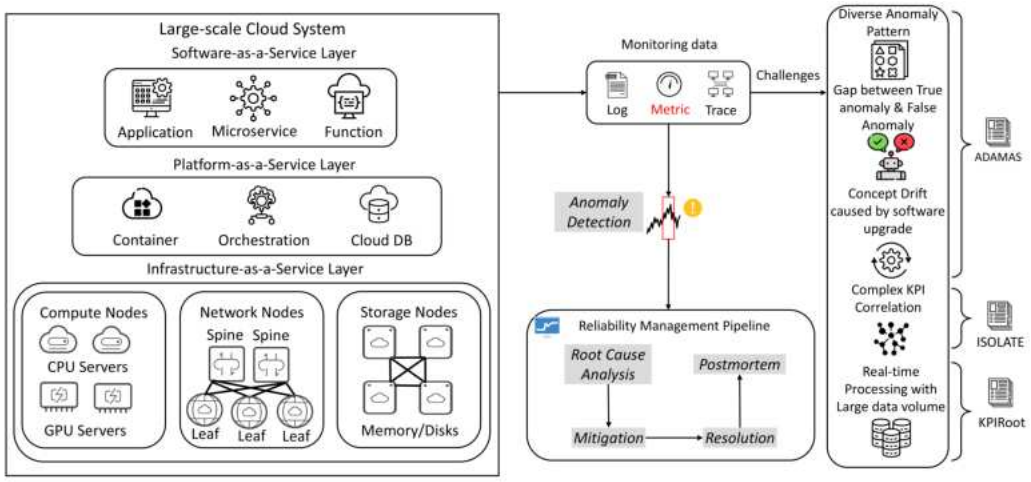


Figure 1.1: Challenges in Monitoring Metric-based Reliability Management of Cloud Systems

1. **Multivariate Performance Anomaly Detection with Relational-Temporal Features**

   In modern cloud systems, identifying performance issues is usually addressed as an anomaly detection problem based on multivariate metrics. Some existing approaches detect anomalies based on individual metrics, focusing solely on temporal abnormal patterns within each metric. However, modern cloud services comprise various components (*e.g.*, storage, computing, middleware), whose corresponding monitoring metrics exhibit complex inter-dependencies. For instance, under normal conditions, there is a correlation between disk I/O operations per second and network throughput. As disk I/O increases due to higher data processing and storage demands, network throughput also increases as more data is transferred to and from storage systems. A significant deviation from this correlation, such as a sudden drop in network throughput while disk I/O remains high, could indicate issues like a network bottleneck or misconfiguration. This deviation can thus signal un-

derlying performance problems in the cloud service.

To address the limitation of overlooking correlations, several graph neural network-based approaches have been proposed to capture the spatial-temporal dependencies among multiple metrics. While these methods consider metric correlations, they typically embed these correlations within latent feature representations implicitly. In contrast, we propose an approach that explicitly incorporates correlation violations. This correlation violation-aware method is crucial for effectively detecting performance problems by using correlation violations as indicators of anomalies.

Identifying performance issues through correlation violations presents four challenges. Firstly, modeling the complicated correlations among numerous metrics automatically and effectively is difficult. A cloud application may consist of tens of microservices, each with tens of metrics, resulting in a large number of metrics with complex correlations. Even experienced engineers struggle to comprehensively elucidate these relationships. Secondly, accurately identifying correlation violations from these intricate inter-metric correlations and determining performance issues is a non-trivial task. Thirdly, due to the vast volume of metrics, it is time-consuming for engineers to investigate underlying issues based on a binary (normal or abnormal) result. Automatically pinpointing the anomalous metrics is necessary. Finally, metric data from large-scale production systems is often noisy, with mild performance issues lacking obvious anomalies. This noise can blur the distinction between normal and anomalous instances, leading to more false positives and negatives. Manually annotating such noisy data is infeasible, while ignoring the noise can result in unsatisfactory outcomes.

To tackle these challenges, we propose Identifying Performance

Issues Based on Relational-Temporal Features (ISOLATE), an automated approach that identifies performance issues by capturing both relational and temporal anomalous features of metrics. ISOLATE utilizes graph neural networks to explicitly capture the complex relational features among metrics, employing a graph attention mechanism to characterize metric correlations. To identify correlation violations effectively, ISOLATE uses relational embedding, which is decoupled from the temporal information of raw metrics, and feeds this into downstream anomaly detection components. Highlighting correlation violation metrics provides insights for software reliability engineers regarding the root cause of performance issues. Additionally, to mitigate the impact of noisy data, ISOLATE adopts Positive Unlabeled learning (PU learning), which iteratively finds positive samples and updates the models. PU learning produces both negative and positive samples, whereas traditional Variational Auto Encoder-based (VAE-based) methods only handle negative examples. ISOLATE employs a novel LC-VAE (Label-Conditional VAE) to distinguish normal and abnormal patterns effectively.

2. **Adaptive Domain-aware Anomaly Detection**

   Cloud services exhibit significant diversity and dynamism, not only in functionalities but also in anomaly patterns. As a result. This situation presents two major challenges for practical anomaly detection in production systems. Firstly, it is difficult for a single method to effectively identify the entire spectrum of anomalies across different services, as suggested by the No Free Lunch Theorem. Secondly, an anomaly detected on a technical level might not translate to a performance issue impacting overall business performance or customer experience, leading to many false positives. Frequent service updates and changing user behaviors cause the anomaly patterns of services

to evolve, a phenomenon known as concept drift, further complicating the problem.

Although efforts have been made to address these challenges, they suffer from limitations that hinder practical applications. For instance, techniques like Automated Machine Learning (AutoML) aim to prevent the need for service-specific anomaly detection solutions. However, these approaches often require substantial labeled data or struggle to evaluate model performance effectively, making it difficult to find the optimal model architecture and parameters. In real-world systems, obtaining sufficient labeled data for each cloud service is challenging. On the other hand, the human-in-the-loop mechanism has proven effective in incorporating domain knowledge to bridge the gap between predicted anomalies and actual business exceptions. However, these approaches heavily rely on human expertise to provide high-quality feedback, affecting scalability and efficiency. Using limited feedback data to retrain models rarely guarantees performance.

To this end, we propose ADAMAS, an AutoML-based anomaly detection framework adaptable to different cloud services with diverse and evolving abnormal patterns. ADAMAS consists of two stages: a Label-free Configuration Search stage and a Feedback-based Adaptive Learning stage. In the first stage, ADAMAS employs Bayesian Optimization to search for the best model architecture and parameters for anomaly detection. During the search, ADAMAS uses the proposed Noise-Free Mean Squared Error with Kurtosis (NFMK) evaluation function to estimate model performance without labels. In the second stage, ADAMAS adopts a lightweight, adaptive learning approach to efficiently incorporate expert knowledge. During online serving, engineers can provide feedback on whether a detected anomaly is a false positive. Using this manually labeled data, ADAMAS employs Metric Stream Clustering (MSC)

to group similar anomalous patterns into clusters and leverage historical feedback to distinguish true performance issues from false positives, significantly reducing the required feedback. To address continuous service updates, ADAMAS triggers model retraining when the cumulative mispredicted samples exceed a threshold. In this process, domain knowledge introduced by human feedback guides the configuration search.

3. **Efficient Monitoring Metric Root Cause Localization**

   Cloud service systems typically consist of many instances, and a common way is to monitor specific monitoring metrics that can reflect the overall performance of the service, *e.g.*, latency, error count, and traffic, which we refer to as alarm KPIs. When a performance issue is detected (*i.e.*, the alarm KPI is abnormal), it is crucial to identify the root cause (*e.g.*, which underlying instances cause the abnormal performance of the service). A practical root cause localization approach for Key Performance Indicators (KPIs) in cloud systems must meet the requirements of efficiency and interpretability. Due to the vast number of KPIs and the urgency to resolve issues quickly, the approach needs to process large amounts of data (*e.g.*, thousands of KPIs) efficiently (*e.g.*, within seconds). Additionally, it should produce interpretable results to help engineers take effective remedial actions, which is crucial for maintaining cloud systems.

   Current root cause localization methods typically rely on statistical or deep learning models. Statistic-based methods compute linear relationships between KPIs and identify the root cause. However, these methods are computationally expensive due to the need to calculate correlations for every KPI pair and often have low accuracy when dealing with complex KPIs in cloud systems. Recent studies have adopted deep learning models, such as graph neural networks, to model KPI relationships for

root cause localization. Despite their potential, these methods also suffer from high computational costs and lack interpretability.

To address these limitations, we propose KPIRoot, an effective and efficient root cause localization approach designed to identify the root cause underlying KPIs when an anomaly is detected in a monitored alarm KPI within cloud systems. To meet the efficiency requirement, KPIRoot first employs Symbolic Aggregate Approximation (SAX) to downsample the time-series data of KPIs, facilitating the extraction of anomaly segments. By filtering out normal KPI data, KPIRoot focuses on anomaly patterns rather than the entire time series, significantly optimizing efficiency. Next, KPIRoot performs both similarity and causality analyses to localize the root cause KPIs. Specifically, underlying KPIs with high similarity in anomaly patterns to the alarm KPI are more likely to be the root causes. Additionally, causality analysis validates the cause-and-effect relationships in the temporal dimension, ensuring that the anomaly pattern of root cause KPIs precedes that of the alarm KPI. Finally, KPIRoot combines the results of similarity and causality analyses to produce a correlation score for each underlying KPI. A higher score indicates a higher likelihood of the KPI being the root cause. With a time complexity of $\mathcal{O}(\sqrt{n})$ (n is the length of the KPIs), KPIRoot can process thousands of KPIs within seconds, enabling real-time resolution of performance issues.

1.3 Thesis Organization

The remainder of this thesis is organized as follows:

1. **Chapter 2: Reliability Management in Cloud Systems**

   Chapter 2 introduces the background and challenges of reliability management in cloud systems. Section 2.1 describes

the typical architecture of cloud systems. Section 2.2 then discusses commonly used monitoring data for reliability management in cloud systems. Finally, Section 2.3 presents monitoring metric-based intelligent solutions that can be integrated into the reliability management of cloud systems.

2. **Chapter 3: Literature Survey on Monitoring Metric-based Intelligent Cloud Reliability Management**

   Chapter 3 reviews existing studies on metric-based cloud reliability management in the literature. We cover a range of approaches, including metric-based anomaly detection, root cause analysis, and clustering. Additionally, we discuss the limitations of these studies in this section, which we try to address in the following sections.

3. **Chapter 4: Multivariate KPI Anomaly Detection**

   Identifying performance issues is often formulated as an anomaly detection problem, which is tackled by analyzing each metric independently. However, this approach overlooks the complex dependencies existing among cloud components. Chapter 4 proposes a method called ISOLATE, a learning-based approach that leverages both the relational and temporal features of metrics to identify performance issues. Specifically, Section 4.1 first introduces the problem and summarizes the contributions made in improving multivariate metric anomaly detection. Then, Section 4.2 presents some typical performance issues due to the violation of the correlation of monitoring metrics, and how the correlation of metrics plays a crucial role in detecting and diagnosing performance issues. The following Section 4.3 describes the details of the proposed method, ISOLATE. Section 4.4 provides a comprehensive evaluation of the proposed method. Section 4.5 shares the industrial experience of deploying ISOLATE in Huawei Cloud and discusses

the threats to the validity of this study. Finally, Section 4.6 summarizes this chapter.

4. **Chapter 5: Adaptive AutoML-based Anomaly Detection**

   It is a common practice in the reliability engineering of cloud services that involves the collection of monitoring metrics, followed by a comprehensive analysis to identify performance issues. However, existing methods often fall short of detecting diverse and evolving anomalies across different services. Moreover, there exists a significant gap between the technical and business interpretation of anomalies, *i.e.*, a detected anomaly may not have an actual impact on system performance or user experience. In Chapter 5, we propose an adaptive AutoML-based anomaly detection ADAMAS to address these limitations. Firstly, Section 5.1 first introduces the problem and summarizes the contributions made in achieving adaptive and domain-aware metric anomaly detection. Next, Section 5.2 provides examples from an industrial scenario that demonstrate the gap between the true anomalies resulting in business interruption and technical anomalies manifested in data. We also give a brief introduction to the general workflow of AutoML. Then, Section 5.3 introduces the design details of ADAMAS, aiming to achieve high adaptiveness and domain-knowledge awareness. Section 5.4 extensively evaluates our method. Section 5.5 discusses the reduction of human efforts with ADAMAS, its overhead, and some potential threats to the validity. Finally, we summarize this chapter in Section 5.6.

5. **Chapter 6: Efficient KPI Root Cause Localization**

   Root cause localization pinpoints the specific metrics when performance issues happen, which are responsible for the degradation of overall service quality, facilitating prompt problem diagnosis and resolution. To this end, existing methods generally identify root-cause KPIs by locating those that exhibit

a similar anomalous trend to the overall service performance. While straightforward, solely relying on the similarity calculation may be ineffective when dealing with cloud systems with complicated interdependent services. In Chapter 6, we propose an effective and efficient method for root cause localization that integrates the advantages of similarity analysis and causality analysis. Specifically, Section 6.1 first introduces the problem and summarizes the contributions made in achieving efficient and effective root cause localization. Section 6.2 provides an overview of the problem of metric-based root cause localization and demonstrates an industrial example that motivates our design. Following this, Section 6.3 then introduces the detailed design of KPIRoot. In Section 6.4, we provide a comprehensive evaluation using real-world data collected from Huawei. Additionally, Section 6.5 discusses the industrial experience of deploying KPIRoot within Huawei Cloud. Section 6.6 discusses the difference between our approach and existing root cause analysis approaches for microservice systems. Then, some potential threats to the study's validity are discussed. Finally, Section 6.7 summarizes this chapter.

6. **Chapter 7: Conclusion and Future Work** In this chapter, we summarize this thesis and discuss our future work. We plan to focus on methods that enhance the reliability of LLM training systems, especially in their storage systems, like large-scale high-bandwidth memory clusters.

□ **End of chapter.**

Chapter 2

Reliability Engineering in Cloud Systems

Software Reliability Engineering (SRE) is a rigorous discipline that focuses on ensuring that software systems operate reliably, performing their intended functions under specified conditions without failure. This discipline integrates concepts from engineering, testing, and management to predict, measure, and enhance the reliability of software. Key components of software reliability engineering include reliability modeling, which involves developing mathematical models to predict software reliability based on historical data and anticipated usage patterns. Failure analysis is also crucial, as it identifies potential failure modes and assesses their impacts. Reliability testing, such as stress and load testing, is used to identify reliability issues, while metrics like mean time between failures (MTBF) and mean time to repair (MTTR) provide quantifiable measures of reliability. Continuous monitoring further ensures that systems are proactively tracked for performance and reliability issues, allowing for timely identification and resolution. Ultimately, reliability improvement strategies, such as code reviews and automated testing, help enhance software reliability by addressing design weaknesses and fostering robust error-handling mechanisms.

Reliability engineering in cloud systems is specifically tailored to ensure cloud services and infrastructure remain dependable, avail-

able, and resilient. This involves defining service level agreements (SLAs) and service level objectives (SLOs) to set expectations for service availability and performance, which act as benchmarks for reliability. Error budgeting plays a critical role, balancing development velocity with reliability by quantifying acceptable levels of unreliability and informing decisions about new feature releases. Automated recovery and self-healing mechanisms are essential, enabling systems to automatically detect and recover from failures with minimal downtime. Comprehensive monitoring and observability tools provide deep insights into system behavior and health, facilitating quick detection of anomalies. Resilience engineering focuses on building fault-tolerant architectures that can withstand and recover from disruptions like network outages or hardware failures. Effective incident management processes ensure rapid response to reliability issues, encompassing detection, communication, documentation, and post-mortem analysis. While these practices offer significant benefits, such as improved reliability and scalability, they also pose challenges, including the complexity of diverse technologies, the need for specialized skills, and the critical task of balancing development speed with system reliability.

As cloud systems provided by platforms like Azure and Google Cloud continue to expand, SRE encounters substantial challenges in maintaining system reliability. The vast scale and complexity of these modern cloud infrastructures necessitate advanced automation and sophisticated monitoring to sustain system stability. The task of balancing rapid development with stringent reliability objectives becomes increasingly intricate, compelling SRE teams to incessantly innovate and adapt their practices to meet the escalating demands imposed by expansive cloud environments. This dynamic landscape requires a comprehensive approach that integrates cutting-edge technologies and methodologies to effectively address the multifaceted challenges of reliability management.

To accommodate the large scale and complexity of cloud sys-

tems, tools for real-time monitoring and automatic analysis have been developed and extensively studied. These tools provide deep insights into system performance and facilitate proactive detection and resolution of potential issues before they impact users. Platforms like Azure and Google Cloud have established robust reliability data management protocols, which standardize practices across their infrastructures to ensure consistency and dependability. These protocols include the use of SLOs, error budgets, and automated recovery mechanisms that enable systems to self-heal and maintain optimal functionality. Additionally, the integration of machine learning and artificial intelligence into monitoring systems enhances predictive capabilities, allowing SRE teams to anticipate and mitigate failures more effectively. As these cloud systems mature, continuous refinement of reliability strategies is essential to sustain high levels of performance and customer satisfaction in an ever-evolving technological landscape.

In the following sections, we first introduce the architecture of cloud service systems in Section 2.1. Then we introduce the common reliability monitoring data of modern cloud systems in Section 2.2. Finally, we introduce intelligent monitoring metric-based practice in enhancing the reliability of cloud systems 2.3.

2.1 Architecture of Cloud Service Systems

Cloud computing services are broadly classified into three primary models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). These categories represent distinct levels of control, flexibility, and management, each tailored to address specific business requirements. IaaS provides foundational computing infrastructure, PaaS delivers a development platform for creating and deploying applications, and SaaS offers complete, ready-to-use software solutions accessible via the internet. These service models collectively enable organizations to opti-

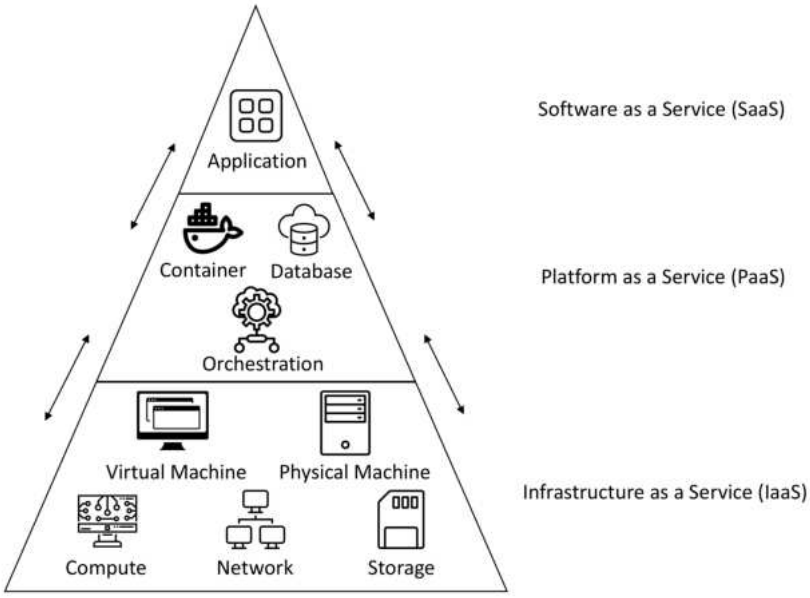


Figure 2.1: Architecture of Typical Cloud Service Systems

mize their IT operations, reduce costs, and enhance scalability while focusing on their strategic goals. The overall architecture of cloud service system is illustrated in Figure 2.1.

Infrastructure as a Service (IaaS) Layer. Infrastructure as a Service (IaaS) is the most fundamental cloud service model, offering essential virtualized computing resources such as virtual machines, storage, and networking over the internet. This model allows users to rent infrastructure components on a pay-as-you-go basis, providing a cost-effective alternative to investing in physical hardware. IaaS empowers organizations with high levels of flexibility and scalability, enabling them to adapt quickly to changing workloads and business demands. It is particularly advantageous for businesses that require dynamic resource allocation or need to handle unpredictable spikes in traffic. However, while IaaS eliminates the need to manage physical infrastructure, it places the responsibility for operating systems, applications, and middleware on the user, necessitating a certain level of technical expertise. Leading providers of IaaS include Amazon Web Services (AWS), Microsoft Azure, and Google Cloud

Platform (GCP), all of which offer robust solutions that support diverse use cases ranging from hosting websites to running large-scale enterprise applications.

Platform as a Service (PaaS) Layer. Platform as a Service (PaaS) builds upon IaaS by providing a higher level of abstraction, focusing on the needs of developers who require an efficient environment for building, testing, and deploying applications. PaaS simplifies the complexities of infrastructure management by offering development tools, middleware, database systems, and runtime environments as part of the service. This enables developers to concentrate on writing code and implementing application logic rather than dealing with hardware and software provisioning. PaaS is particularly well-suited for projects involving rapid development cycles, collaborative workflows, and continuous integration/continuous deployment (CI/CD) pipelines. Moreover, it streamlines application lifecycle management, making it easier to scale applications as needed. Prominent examples of PaaS platforms include Google App Engine, Microsoft Azure App Service, and Heroku, all of which provide robust ecosystems for creating modern, cloud-native applications.

Software as a Service (SaaS) Layer. Software as a Service (SaaS) represents the most advanced and user-centric cloud service model, delivering fully functional software applications directly to end users over the internet. Unlike IaaS or PaaS, SaaS requires no installation, maintenance, or management of the underlying infrastructure or software, as these responsibilities are handled entirely by the service provider. SaaS applications are hosted in the cloud and accessed through web browsers, offering unparalleled convenience, scalability, and accessibility from virtually any device with an internet connection. This model is ideal for businesses aiming to minimize IT complexity and focus on their core operations, as it eliminates the need for extensive technical expertise or on-premises hardware. SaaS solutions cover a wide range of applications, including customer relationship management (CRM), enterprise resource

planning (ERP), collaboration tools, and industry-specific software tailored to specialized needs. Examples of popular SaaS providers include Salesforce, Microsoft Office 365, and Google Workspace, which have become essential tools for businesses seeking streamlined workflows, enhanced productivity, and reduced operational overhead.

Cross-layer Interactions. Cross-layer interactions in cloud computing systems involve the dynamic interplay between Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) layers, which collectively enhance the functionality and efficiency of cloud solutions. IaaS serves as the foundational layer, providing the essential infrastructure components such as virtual machines, storage, and networking, which underpin both PaaS and SaaS offerings. PaaS, operating atop the IaaS layer, leverages these infrastructure resources to deliver a robust platform that simplifies application development, deployment, and scaling. This interaction allows PaaS to dynamically allocate resources from IaaS, ensuring optimal performance and cost-efficiency based on varying workloads and developer needs. Moreover, PaaS abstracts the complexities of infrastructure management, enabling developers to focus on application logic and innovation without delving into the intricacies of hardware provisioning. At the SaaS level, applications utilize the extensive capabilities of PaaS to manage complex operations and integrations, while simultaneously relying on the scalable infrastructure provided by IaaS for hosting and execution. This hierarchical relationship ensures that SaaS applications can deliver seamless user experiences, characterized by responsiveness and reliability, without direct involvement in infrastructure management. Additionally, cross-layer interactions facilitate integrated security measures, enabling consistent governance and compliance across all layers. For instance, security protocols initiated at the IaaS level can propagate through PaaS and SaaS, ensuring data protection and privacy throughout the service stack. This interconnected framework

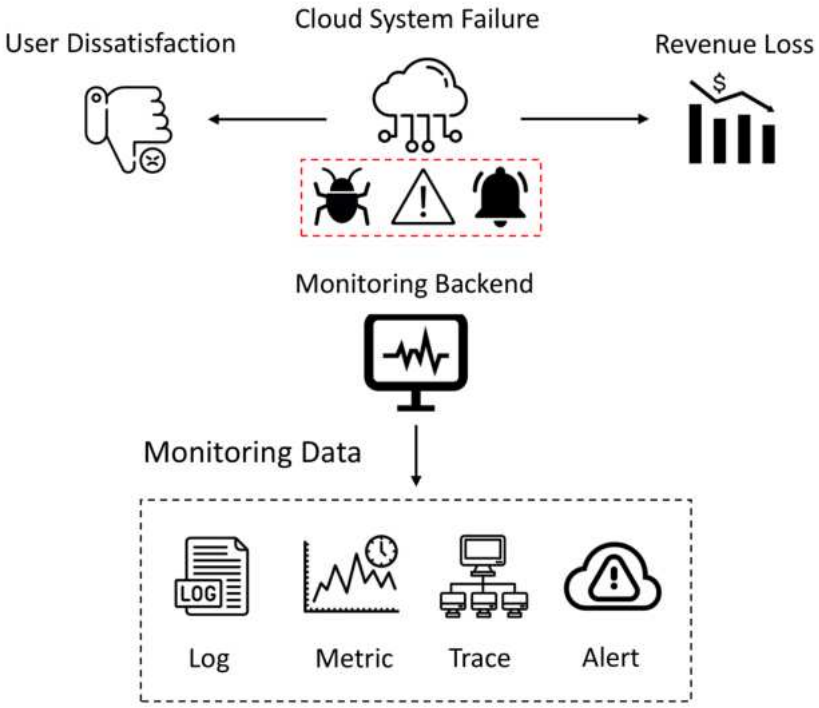


Figure 2.2: Cloud System Monitoring Data

enables organizations to optimize resource usage, streamline development processes, and enhance the value delivered through cloud services, ultimately creating agile and scalable solutions tailored to specific business objectives.

2.2 Cloud System Monitoring Data

Cloud system monitoring is essential for ensuring the health, performance, and security of cloud-based applications and infrastructure. It involves the continuous observation and analysis of various components within a cloud environment to quickly identify and address any issues that arise. This practice typically entails the collection of three key data types: metrics, logs, and traces. Monitoring metrics offer quantitative insights into resource performance, measuring factors such as CPU usage, memory utilization, disk I/O, and

network traffic. These metrics are vital for assessing system health and identifying potential performance bottlenecks. Logs provide detailed records of events within the cloud environment, encompassing application operations, system events, and security incidents. They offer a chronological view of activities, enabling effective troubleshooting and compliance monitoring. Traces track the flow of requests across distributed systems, capturing the lifecycle and performance of these requests to help pinpoint bottlenecks and failures. Alerts are notifications triggered by predefined conditions in metrics, logs, and traces. They provide timely awareness of potential issues, allowing for rapid response to maintain system stability and performance. By leveraging monitoring tools like Amazon CloudWatch, Azure Monitor, and Google Cloud Operations Suite, organizations can proactively detect and resolve issues, optimize system performance, ensure security and compliance, and manage costs effectively. When performance issues and failures occur, they can lead to revenue losses and user dissatisfaction. To mitigate these risks, engineers conduct detailed analyses on monitoring data to identify and resolve problems promptly. By leveraging monitoring tools like Amazon CloudWatch, Azure Monitor, and Google Cloud Operations Suite, organizations can proactively detect and resolve issues, optimize system performance, ensure security and compliance, and manage costs effectively. The overall pipeline of monitoring data collection is illustrated in Figure 2.2

Metrics: Metrics are quantitative measures that provide insights into the performance and health of cloud systems. Examples include CPU utilization, memory usage, disk I/O, and network traffic. By continuously collecting and analyzing metrics, organizations can monitor resource usage, identify performance bottlenecks, and ensure that systems are running efficiently. Metrics are crucial for real-time monitoring and can trigger alerts when predefined thresholds are exceeded, indicating potential issues that need immediate attention.

Logs: Logs are detailed records of events that occur within a cloud environment. They capture information about application operations, system events, and security incidents. Logs provide a chronological account of activities, making them invaluable for diagnosing issues, tracking user activities, and ensuring security compliance. By analyzing logs, administrators can detect anomalies, investigate errors, and understand the context of specific events. Log-based alerting involves monitoring these logs for specific patterns or keywords that indicate problems or security threats.

Traces: Traces track the flow of requests through distributed systems, capturing the lifecycle and performance of each request. They provide a detailed view of how different services interact, helping to identify bottlenecks and diagnose performance issues in complex architectures. Traces typically include spans, which are individual units of work within a trace, detailing start and end times, duration, and any errors encountered. Trace-based alerting involves monitoring these traces for performance anomalies or error patterns, triggering alerts when predefined conditions are met.

Alerts: Alerts, also known as events or alarms, are crucial for maintaining the health and performance of cloud systems. They provide timely awareness of issues so they can be resolved quickly. Alerts are generated by monitors—always-on programs—based on predefined policies. These policies describe the circumstances under which customers want to be alerted, such as thresholding metrics, anomaly detection in logs, or specific patterns in traces. An alert typically contains key information like an ID, timestamp, severity level, textual message, and source, which helps the recipient quickly understand the issue and take appropriate action.

By leveraging metrics, logs, traces, and alerts, cloud systems can ensure optimal performance, quickly detect and resolve issues, and maintain a secure and compliant environment. This integrated approach to monitoring and management is essential for the smooth operation of cloud-based applications and infrastructure.

2.3 Metric-based Intelligent Reliability Engineering

In modern cloud environments, managing reliability data manually presents significant challenges due to the scale and complexity of these systems. With numerous components generating vast amounts of monitoring metrics, it becomes increasingly difficult for human operators to effectively monitor, analyze, and respond to issues in real-time. The sheer volume of data can overwhelm teams, leading to missed alerts, delayed responses, and potential system failures. Additionally, the intricacies of distributed systems, with their complex dependencies and interactions, complicate the manual diagnosis of problems and identification of root causes. This can result in inefficient troubleshooting, prolonged downtimes, and increased operational costs.

To address these challenges and enhance operational efficiency, it is essential to leverage intelligent and automatic analysis of reliability data. By incorporating advanced analytics, machine learning, and automation tools, organizations can proactively detect anomalies, predict potential issues, and streamline incident response processes. Intelligent systems can continuously analyze large datasets, identify patterns, and provide actionable insights faster and more accurately than manual methods. Embracing these technologies not only improves the reliability and performance of cloud systems but also allows teams to focus on strategic initiatives and innovation, ultimately driving better business outcomes.

In modern cloud systems, the sheer volume and diversity of metrics generated can be overwhelming, making it essential to implement automatic anomaly detection and clustering techniques. Automatic anomaly detection helps identify unusual patterns or behaviors in real time, ensuring prompt responses to potential issues. Simultaneously, metric-based clustering groups similar data points, enabling the identification of patterns, trends, and anomalies at scale. Together, these techniques aim to provide deeper insights into system

performance and usage patterns, facilitating more informed decision-making and resource optimization.

2.3.1 Metric-based Anomaly Detection

Metric-based anomaly detection is a technique used to identify unusual patterns or behaviors in system metrics, indicating potential issues such as hardware failures, security breaches, or performance bottlenecks. It involves monitoring various quantitative measures like CPU usage, memory utilization, and network latency to detect deviations from the norm. This is effective for identifying functional and performance anomalies at both the application and service levels, assuming consistent conditions between training and runtime. However, its accuracy can decrease if runtime conditions change, such as during peaks in user requests or when new services replace outdated ones. Alternatively, other techniques like Service Level Objective (SLO) checks and heartbeat protocols are used to detect anomalies, though they may limit the type or granularity of anomalies that can be detected.

In terms of methods, machine learning-based anomaly detection includes techniques like One-Class Support Vector Machine (OCSVM), Isolation Forest (iForest), and Local Outlier Factor (LOF), which leverage clustering and density estimation to identify anomalies. Deep learning-based methods include models like Deep Autoencoder with Gaussian Mixture Model (DAGMM), Long Short-Term Memory (LSTM) networks, and Graph Neural Networks (GNNs) that capture complex temporal and spatial dependencies for more nuanced anomaly detection. AutoML-based methods, such as HyperBand and ProxyBO, automate model selection and hyperparameter tuning to improve anomaly detection efficiency, though they often operate offline and may not adapt well to online scenarios with dynamic conditions. Each of these methods provides unique capabilities for managing the complexities of anomaly detection in cloud environments.

2.3.2 Metric-based Root Cause Analysis

Root cause analysis in the context of monitoring metrics is a crucial process aimed at identifying the underlying factors responsible for anomalies or performance issues within cloud-based applications and services. This analysis involves examining key performance indicators (KPIs) that are monitored by agents deployed alongside application services. By meticulously analyzing these metrics, organizations strive to determine the precise causes of observed anomalies, which could be related to hardware failures, software bugs, network disruptions, or other operational challenges. Effective root cause analysis not only aids in resolving existing issues but also contributes to preventing future occurrences, thereby enhancing the reliability and performance of cloud systems.

There are several broad categories of methods used in root cause analysis. Statistical analysis techniques leverage statistical methods to evaluate KPIs, identifying anomalous patterns that might indicate potential root causes; examples include ϵ-diagnosis. Topology graph-based analysis methods utilize topology graphs to model service dependencies and interactions within applications, as seen in techniques like MicroRCA. Lastly, causality graph-based analysis involves the construction and examination of causality graphs to explore the relationships between services and their KPIs, identifying causal dependencies that might lead to anomalies, with examples including CauseInfer, Microscope, CloudRanger, and MicroCause. Each method provides unique insights into the complex interactions within cloud environments, helping to pinpoint possible root causes of service issues.

2.3.3 Metric-based Clustering

Metric-based clustering is a technique that groups similar data points based on specific metrics or features, aiding in the identification of patterns, trends, and anomalies within cloud systems and IT infras-

tructure. By employing clustering algorithms such as K-Means, hierarchical clustering, DBSCAN, and Gaussian Mixture Models, organizations can optimize resource allocation, detect anomalies, analyze performance, and plan capacity. The process involves collecting and preprocessing data, selecting relevant features, choosing an appropriate algorithm, training the model, and interpreting the results. Metric-based clustering enhances system understanding, enables proactive management, and improves efficiency by identifying and grouping similar workloads and usage patterns. By implementing these intelligent reliability engineering techniques, organizations can overcome the challenges posed by the complexity and volume of data in cloud environments, ensuring robust and efficient operations.

□ **End of chapter.**

Chapter 3

Literature Survey on Monitoring Metric-based Intelligent Cloud Reliability Management

In this chapter, we review existing studies on monitoring metric-based intelligent cloud reliability engineering in the literature. Extensive studies have been conducted in this area. Our goal is to provide a comprehensive overview of these studies, highlighting their contributions and discussing their limitations.

3.1 Metric-based Analysis

3.1.1 Metric-based Anomaly Detection

Detecting performance issues through monitoring metrics in online service systems has become increasingly important. These metrics, often represented as multivariate time series, are critical for assessing the real-time status of entire systems. The primary challenges in detecting anomalies within multivariate metrics are twofold: effectively modeling complex relational and temporal dependencies [212] and addressing noise introduced during manual labeling processes, which can be difficult to eliminate. Related studies are generally divided into approaches based on machine learning or signal pro-

cessing and those based on deep learning.

One-Class Support Vector Machine (OCSVM) [168] is a clustering-based method that defines boundaries around normal data, identifying anomalies as points outside these borders. Isolation Forest (iForest) [121] employs multiple isolation trees, assuming anomalies are rare and isolated, making them easy to detect due to their short path lengths. Local Outlier Factor (LOF) uses density estimation for anomaly detection, identifying samples with significantly lower density compared to their neighbors as anomalies. JumpStarter [136] utilizes compressed sensing for data reconstruction, employing shape-based clustering to reduce data volume and outlier-resistant sampling to minimize anomalous values. ADSketch [25] is an online performance anomaly detection approach that uses pattern sketching to compare anomaly patterns with historically identified ones, updating cluster synopses with evolving data.

The advent of deep learning has spurred a multitude of studies focused on anomaly detection in multivariate metrics data [233, 234]. The Deep Autoencoder with Gaussian Mixture Model (DAGMM) [255] identifies anomalous data points without considering temporal patterns. MSCRED [231] employs a multi-scale convolutional recurrent encoder-decoder to detect anomalies based on reconstruction error. Long Short-Term Memory (LSTM) models [76, 242] are utilized to detect performance degradation anomalies in software systems by leveraging prediction errors. The LSTM-VAE [157] combines LSTM networks with a Variational Autoencoder (VAE) to reconstruct sliding window distributions from multivariate metrics, effectively handling temporal dependencies [25]. Similarly, LSTM-NDT [76] utilizes LSTM networks with non-parametric dynamic thresholds to ensure system reliability. OmniAnomaly [182] extends LSTM-VAE with normalizing flow, using reconstruction error for detection, though its effectiveness diminishes with severe training noise. THOC [174] fuses multi-scale temporal features using hierarchical clustering, detecting anomalies through multi-layer

distance loss. MTSAD [201] integrates active learning with VAE-based anomaly detection, distinguishing between normal and abnormal samples in reconstruction error and latent space. TranAD [189], a transformer-based model, employs attention-based sequence encoders for efficient anomaly detection, enhanced by self-conditioning and adversarial training for stability. ACVAE [240] uses adversarial mechanisms and contrast learning in a VAE framework, improving the decoder's discriminatory power and the encoder's sample acquisition. SLA-VAE [73] is a semi-supervised model using convolutions for capturing inter-metric dependence, with active learning to update the VAE model using uncertain samples.

Recently, Graph Neural Networks (GNNs) have emerged as promising tools in deep learning-based anomaly detection, effectively capturing temporal and spatial dependencies among multivariate metric pairs [89]. MTAD-GAT [243] pioneers the use of graph-attention networks to model sensor relationships, combining reconstruction and forecasting for anomaly detection. Graph Deviation Network (GDN) [35] learns pairwise relationships via cosine similarity, modeling time series as graphs using adjacency matrices, predicting future values and using forecast error as an anomaly score. GTA [22] is a Transformer-based framework that automatically learns sensor dependencies, employing Influence Propagation (IP) graph convolution and multi-branch attention for enhanced training efficiency. FuSAGNet [59] leverages a sparse autoencoder to extract latent feature representations, predicting future time series using sparse representations and graph structures learned by GNNs. TopoMAD [69] is a topology-aware model integrating GNNs, LSTM, and VAE, extracting topological information through pod-based division in Kubernetes microservices. However, due to the dynamic topologies in Huawei Cloud from frequent deployments and updates, TopoMAD is not applicable in our scenario.

In large-scale cloud systems, manual model selection and hyperparameter tuning are time-consuming and error-prone, prompt-

ing the adoption of AutoML techniques. HyperBand [107] combines randomized search through Successive Halving with an early-stopping mechanism, though its random nature and disregard for historical configuration data can lead to suboptimal results. ProxyBO [175] speeds up neural architecture search using three proxy functions, but these proxies are not specifically designed for anomaly detection. AutoAD [160] is the first AutoML framework employing unsupervised measurement for model evaluation, though it uses MSE as the evaluation function, which may not accurately reflect performance. AutoKAD [230] applies Bayesian Optimization [10], iteratively searching for optimal configurations using the MSE-NF function. Both methods explore the relationship between performance and hyperparameter settings, searching for the best configuration. However, they fail to distinguish between true and predicted anomalies, lacking domain knowledge from cloud system experts, which can result in high false positives and burden cloud operators. Moreover, existing AutoML-based anomaly detection frameworks are typically offline, making them unsuitable for online scenarios with evolving software and anomaly patterns.

3.1.2 Metric-based Root Cause Analysis

Determining the root cause of performance anomalies in online service systems has become a critical area of research. The objective of root cause localization using monitoring metrics in cloud systems is to identify a subset of monitored Key Performance Indicators (KPIs), which can then be investigated to alleviate performance issues. LOUD [141] operates on the premise that services exhibiting anomalous KPIs are likely to cause anomalous behavior in correlated services. It utilizes graph centrality to determine the degree of correlation between KPIs and observed performance anomalies. AID [221] measures the intensity of dependencies between monitoring KPIs of cloud services by calculating the similarity between

status KPIs of the caller and the callee. AID aggregates these similarities into a unified value representing the dependency intensity, making it useful for root cause localization by identifying the KPI triggering alerts. Similarly, CloudScout [222] employs the Pearson Correlation Coefficient to assess the similarity between services based on KPIs at the physical machine level, such as CPU usage.

Several approaches focus on identifying fault-indicating attribute combinations within KPI data. CMMD [218] uses a graph attention network for cross-metric root cause localization, modeling relationships between fundamental and derived metrics. HALO [238] introduces a hierarchical search approach, utilizing conditional entropy to capture relationships among attributes and locate fault-indicating combinations. iDice [118] treats the root cause as a combination of attribute values, identifying anomalies through specific attribute dimension co-occurrences, and ranks these combinations using a Fisher distance-based score function. However, iDice is not suitable for large-scale issue reports with high-dimensional metrics typical of cloud systems. MID [50] employs a meta-heuristic search to dynamically detect emerging issues from large-scale issue reports with higher efficiency.

3.1.3 Monitoring Metric-based Clustering

Effectively handling the large volume of metrics generated by cloud systems is crucial, and clustering plays a key role in this process. For example, Principal Component Analysis (PCA) is used to transform multivariate metric data into univariate time series before clustering [90]. ROCKA [113] is a robust and rapid monitoring metric clustering algorithm based on shape-based distance, designed to handle challenges like noise and phase shifts in large-scale anomaly detection. ROCKA clusters metrics based on their underlying shapes, ensuring high accuracy and reducing model training time with minimal performance loss. OmniCluster [237] is another system instance

clustering method that combines a one-dimensional convolutional autoencoder with a novel three-step feature selection strategy. This approach efficiently clusters system instances, significantly reducing the training overhead of anomaly detection models. However, these methods often require pairwise distance computations before performing clustering, which can be computationally intensive and challenging to scale in cloud environments with tens of millions of monitoring metrics.

3.2 Log-based Analysis

3.2.1 Log Parsing

The foundational step in automated log analysis is log parsing, a field that has seen significant research activity in recent years [60,86, 94, 254]. The objective is to distill raw log messages into structured events by separating constant templates from variable parameters. Historically, research in this domain has bifurcated into two primary paradigms: syntax-based and semantic-based approaches.

These methods, which operate on the logs' textual structure and patterns, represent the classical approach. Early and foundational methods within this category are **frequency-based**, which leverage frequent token positions or n-gram patterns to identify templates [32, 150, 191, 192]. A second category operates on the principle of **similarity**, computing distances between log messages to cluster them and subsequently extract their common, static components [58,177,188]. A third, more diverse group relies on **heuristics**, employing specialized algorithms or data structures, such as fixed-depth trees in *Drain* [62], to efficiently differentiate templates based on designed characteristics [37, 87, 140, 145, 148, 203].

In contrast to syntax-driven methods, semantic-based parsers aim for a deeper understanding of logs, which can yield superior parsing accuracy [77, 110]. Advancements in language models have revolu-

tionized this paradigm. Early semantic approaches formulated log parsing as a token classification problem, employing neural architectures like Bi-LSTM networks [129].

However, the recent advent of Large Language Models (LLMs) has catalyzed a significant shift, creating a new frontier for log parsing research. This new wave of research explores the trade-offs between large-scale commercial LLMs and smaller, open-source alternatives. On one hand, the immense power of commercial models has been harnessed for high accuracy. For instance, models like ChatGPT can dynamically generate log templates without prior knowledge by analyzing message context [161]. Similarly, DivLog [216] enhances parsing by using GPT-3 for in-context learning, selecting demonstrations from a candidate set of labeled logs. However, these commercial APIs' high cost and latency prompted efficiency-focused solutions like LILAC [85], which incorporates an adaptive cache to store parsing results and reduce redundant queries.

To address cost and data privacy concerns more directly, a parallel line of research has focused on fine-tuning and prompting open-source LLMs. LogPPT [102] was an initial attempt, utilizing a masked language model (RoBERTa) with few-shot learning to classify log tokens. Subsequent work, such as LLMParser [137], systematically demonstrated that even small open-source LLMs can achieve high accuracy with minimal fine-tuning. Building on these ideas, Hooglle [20] adopted an LLM pre-trained specifically on labeled log data. A recent unsupervised approach, LibreLog [138], was introduced to leverage open-source LLMs like Llama3-8B. LibreLog first groups logs using a tree structure and then refines templates through a novel combination of retrieval-augmented generation and iterative self-reflection, eliminating the need for manual labels. This privacy-preserving method significantly improved both parsing accuracy and processing speed compared to previous state-of-the-art LLM-based parsers.

3.2.2 Advancements in Log-based Anomaly Detection

Once logs are structured via parsing, anomaly detection is a primary downstream application. The evolution of these methods can be traced through two major eras: those built on classical machine learning and more recent approaches employing deep learning.

Pioneering work in this area applied established ML techniques to log event sequences. For instance, Principal Component Analysis (PCA) was utilized by Xu et al. [217] to identify system problems from console logs. Other approaches focused on mining invariants between log messages [130] or clustering similar log sequences to recommend representative patterns for problem identification [119]. To improve context, some methods like Log3C [65] incorporated system monitoring metrics alongside log data. A comprehensive evaluation by He et al. [66] in their work on Loglizer systematically explored the application of various ML methods for this task.

The advent of deep learning brought about a paradigm shift, enabling models to automatically learn complex temporal and semantic patterns from log sequences without manual feature engineering. Du et al. [38] introduced *DeepLog*, which employed a Long Short-Term Memory (LSTM) network to model normal execution flows and detect deviations. *LogAnomaly* [144] extended this by incorporating both log count and semantic vectors to create more robust sequence representations. A key distinction within these DL-based methods lies in the training paradigm. While early models like *DeepLog* and *LogAnomaly* operated in an unsupervised manner, subsequent research demonstrated the superior performance of supervised models, such as CNN-based [132] and GRU-based LogRobust [241] architectures. Recognizing the scarcity of labeled data, semi-supervised methods like PLElog [220] were developed to bridge this gap using probabilistic label estimation.

A more recent direction focuses on the complex interactions between system entities, often interleaved within log files. To address

this challenge, researchers have proposed graph-based methods. An example is Lograph [30], which introduces a log semantic association mining approach to convert log sequences into a Log-Entity Graph. This method can implicitly group interleaved logs by explicitly modeling the dependencies between logs and the entities that generate them. It then utilizes a Heterogeneous Graph Attention Network to capture anomalous patterns.

However, despite their high accuracy, DL-based methods have drawbacks. The computational overhead, in terms of both time and space complexity, of architectures like LSTMs and Transformers can make them difficult to deploy on local instances for real-time monitoring. Furthermore, their performance can degrade when faced with evolving log patterns not seen during training, a common challenge in dynamic operational environments.

□ **End of chapter.**

Chapter 4

Multivariate KPI Anomaly Detection

Cloud systems, typically comprised of various components (*e.g.*, microservices), are susceptible to performance issues, which may cause service-level agreement violations and financial losses. Identifying performance issues is thus of paramount importance for cloud vendors. In current practice, crucial metrics, *i.e.*, key performance indicators (KPIs), are monitored periodically to provide insight into the operational status of components. Identifying performance issues is often formulated as an anomaly detection problem, which is tackled by analyzing each metric independently. However, this approach overlooks the complex dependencies existing among cloud components. Some graph neural network-based methods take both temporal and relational information into account, however, the correlation violations in the metrics that serve as indicators of underlying performance issues are difficult for them to identify. Furthermore, a large volume of components in a cloud system results in a vast array of noisy metrics. This complexity renders it impractical for engineers to fully comprehend the correlations, making it challenging to identify performance issues accurately. To address these limitations, we propose Identifying Performance Issues based on Relational-Temporal Features (ISOLATE), a learning-based approach that leverages both the relational and temporal fea-

tures of metrics to identify performance issues. In particular, it adopts a graph neural network with attention to characterizing the relations among metrics and extracts long-term and multi-scale temporal patterns using a GRU and a convolution network, respectively. The learned graph attention weights can be further used to localize the correlation-violated metrics. Moreover, to relieve the impact of noisy data, ISOLATE utilizes a positive unlabeled learning strategy that tags pseudo labels based on a small portion of confirmed negative examples. Extensive evaluation on both public and industrial datasets shows that ISOLATE outperforms all baseline models with 0.945 F1-score and 0.920 Hit rate@3. The ablation study also proves the effectiveness of the relational-temporal features and the PU-learning strategy. Furthermore, we share the success stories of leveraging ISOLATE to identify performance issues in Huawei Cloud, which demonstrates its superiority in practice.

4.1 Introduction

Cloud computing has surged in popularity in recent years. Large-scale cloud vendors, *e.g.*, Microsoft Azure, Amazon Web Services, and Google Cloud Platform, provide customers with various services over the Internet [162]. Due to the scale and complexity, performance issues are inevitable [83] in cloud systems. Such performance issues may degrade overall availability and increase service response time, thus causing SLA (Service Level Agreement) violations and substantial economic losses [5, 109]. Therefore, identifying performance issues accurately is a critical task during the maintenance of cloud systems [78].

In current practice, cloud vendors typically collect crucial metrics (*i.e.*, Key Performance Indicators, KPI), such as CPU utilization and network latency, and then analyze the collected metrics to identify performance issues. Simple as the process might seem, it is a non-trivial task. In particular, a cloud system is typically vast in scale and

consists of tremendous software and hardware components (*e.g.*, microservices, and virtual machines and servers) [56, 158, 225]. Each component may have tens of metrics to be collected in the backend monitoring system, resulting in a large volume of monitoring metrics [154, 206]. Since analyzing tremendous data manually is labor-intensive and error-prone [26], automatic approaches that help on-call engineers identify performance issues are preferred.

In the literature, the performance issue identification problem is generally formulated as an anomaly detection problem based on multivariate metrics [15, 166, 215]. Some existing approaches [174, 182, 242] detect anomalies based on individual metrics, *i.e.*, they only consider the temporal abnormal patterns on individual metrics. However, modern cloud service typically consists of various components (*e.g.*, storage, computing, middleware), whose corresponding monitoring metrics have complicated inter-dependencies [29, 133, 166]. Take disk I/O operations per second and network throughput as an example: Under normal operating conditions, there is typically a correlation between these two metrics; as disk I/O operations increase due to higher data processing and storage demands, network throughput also increases as more data is being transferred to and from the storage systems. However, if a performance issue arises, this correlation can be violated. For instance, if disk I/O remains high while network throughput suddenly drops significantly, it could indicate a network bottleneck, a network gateway failure, or a network misconfiguration. This deviation from the expected correlation between disk I/O and network throughput can thus be a strong indicator of underlying performance issues in the cloud service.

To alleviate the drawback of overlooking correlations, several graph neural network-based approaches that take the spatial-temporal dependency between multiple metrics [22, 35, 59, 127, 243] are proposed. Though these approaches consider the correlation between metrics, these correlations are typically embedded within the latent feature representation in an implicit manner. In other words, these

approaches essentially detect temporal anomalies in these latent representations. In contrast, we propose a correlation violation-aware approach that incorporates the correlation violation explicitly. To effectively detect correlation-violated performance problems, it is essential to account for the correlation violation among metrics as performance anomaly criteria.

However, there are four challenges in identifying performance issues through correlation violations. Firstly, it is hard to automatically and effectively model the complicated correlations among a variety of metrics. A cloud application could comprise tens of microservices. Each microservice may still have tens of metrics, leading to a large number of metrics. Moreover, the correlation between metrics is complicated. Even experienced engineers cannot comprehensively clarify the relations among different metrics. Secondly, even with these correlations extracted, the process of accurately identifying the correlation violations from these intricate inter-metric correlations and then determining the happening of performance issues is a non-trivial task. Thirdly, due to the huge volume of metrics, it is still time-consuming for engineers to investigate the underlying issues simply given a binary (*i.e.*, normal and abnormal) result. Automatically pinpointing the anomalous metrics is required. Finally, metric data from large-scale production systems are noisy [25], *i.e.*, mild performance issues without obvious anomalies in the metric data. This can blur the distinction between normal and anomalous instances, leading to more false positives and false negatives. Unfortunately, it is infeasible to annotate a huge volume of noisy data manually, while simply neglecting such noises may lead to unsatisfying results.

To address these challenges, we propose Identifying Performance Issues Based on Relational-Temporal Features (ISOLATE), an automated approach to identify performance issues through both capturing the violation of relational features and detecting temporal anomalous features of metrics. To solve the challenge of intricate

correlations, ISOLATE utilizes graph neural networks to capture the complicated relational features among metrics explicitly, namely, it employs the graph attention mechanism to characterize the correlations between metrics. To effectively identify the correlation violation between metrics, relational embedding, which is uncoupled from the temporal information of raw metrics, is utilized and fed to downstream anomaly detection components. The correlation-violation metrics are highlighted to provide insights to software reliability engineers like the root cause of the performance issue. Furthermore, to relieve the impact of noisy data, Positive Unlabeled learning (PU learning) is adopted, which finds positive samples and updates the models iteratively. Since PU learning produces both negative and positive samples, while traditional Variational Auto Encoder-based (VAE-based) methods can only take negative examples, ISOLATE employs a novel LC-VAE (Label-Conditional VAE) to distinguish normal and abnormal patterns from the labeled inputs effectively.

To evaluate the performance of ISOLATE, we conduct extensive experiments on a widely-used public dataset and two industrial datasets collected from Huawei Cloud. The experimental results demonstrate that compared with state-of-the-art baselines, ISOLATE achieves the best performance issue identification accuracy with an average F1 score of 0.945. ISOLATE also provides anomalous metrics localization with a Hit rate@3 of 0.920. Moreover, we also conduct ablation studies to validate the effectiveness of our design. A case study with two real cases in Huawei Cloud further shows the practical usefulness of our proposed ISOLATE.

We summarize the main contributions of this work as follows:

- We propose an end-to-end model that captures the relational violation among monitoring metrics explicitly, offering a more precise way to identify correlation-violated performance anomalies. The violated correlations are utilized to localize the root cause of the performance anomaly. Besides, to alleviate the negative effect

of concealed noise in training data, we adopt positive unlabeled learning (PU Learning) on metrics performance anomaly detection and propose the Label-Conditional Variational Autoencoder (LC-VAE) that works seamlessly with PU Learning.

- We conduct extensive evaluations of ISOLATE on two industrial datasets collected from large-scale online service systems of Huawei Cloud and a publicly available dataset. The results demonstrate that ISOLATE outperforms eighteen state-of-the-art methods.
- We have successfully deployed ISOLATE into the troubleshooting system of Huawei Cloud. The success stories of our deployment confirm the applicability and effectiveness of our method.

4.2 Background

In this section, we first introduce the background knowledge about performance issue detection in modern cloud systems. Then, we present an example of an industrial scenario that motivates this work. Finally, we summarize some typical performance issues due to the violation of the correlation of monitoring metrics, which aims to provide a comprehensive understanding of the performance issues and how the correlation of metrics plays a crucial role in detecting and diagnosing performance issues.

4.2.1 Monitoring Metrics in Cloud Service Systems

In recent years, cloud service systems have gained significant attention due to their ability to provide scalable, on-demand resources and services. Typically, coupled multivariate metrics are collected at run time to monitor the overall status of the cloud service systems. The collected metrics provide insights into the performance of logical and physical resources within the system, allowing operators

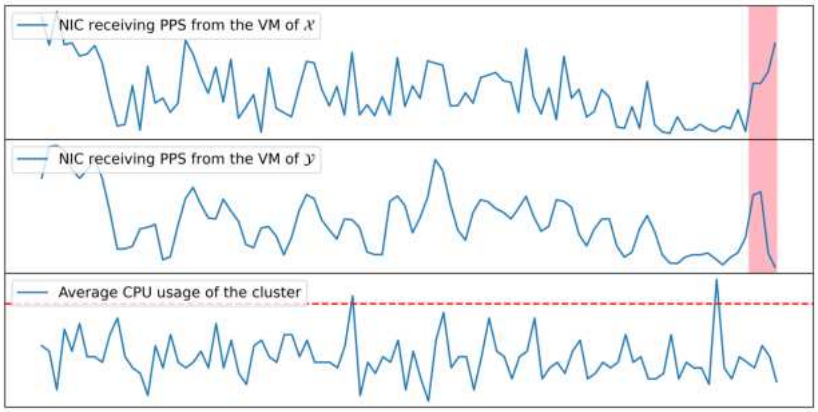


Figure 4.1: A real-world example of performance anomaly

to identify and address performance issues before they lead to service disruptions. However, due to the complex inter-dependencies between system components [48], these metrics are often strongly correlated with each other, reflecting the interconnected nature of the system that makes it challenging to isolate and identify specific performance issues. For example, the performance of a virtual machine may be influenced by the workload placed on the underlying physical server, and the performance of a microservice may depend on the performance of other microservices it interacts with. As a result, the metrics collected from different components can exhibit strong correlations with each other, reflecting the complex interdependencies within the system.

4.2.2 Performance Issues due to Correlation Violation

Performance issues have emerged as a primary concern, potentially undermining the effectiveness of cloud service systems. Some typical factors contributing to these performance issues are resource overload and network latency. Resource overload occurs when the demand for resources exceeds the available capacity, resulting in performance degradation and potential service disruptions [146]. Network latency, exacerbated by the distributed nature of cloud sys-

tems, can result in reduced application responsiveness, impacting the overall user experience [11]. While these performance issues can be detected by considering single metrics, there are other performance issues that are caused by the violation of correlations between different system metrics, which do not arise from a single metric exceeding a threshold but from a violation in the expected correlation between metrics. Considering the intrinsic correlations between metrics in cloud service systems, accurately capturing and localizing these correlation violation metrics is of great significance to performance issue identification. By localizing anomalous metrics, we gain valuable insights into the specific metrics that deviate from normal behavior (*i.e.*,violation of correlation to other metrics). Through accurate metric localization, system engineers can swiftly identify the entities that undergo performance anomalies, facilitating prompt troubleshooting and proactive issue resolution.

An industrial case in the online service system of Huawei Cloud is shown in Figure 4.1. Three metrics of the Elastic Load Balancing (ELB) service are shown in the figure for convenience of presentation. Specifically, the first metric represents the network interface card (NIC) receiving packets per second (PPS) from a virtual machine running microservice $\mathcal{X}$, and the second metric represents the NIC receiving PPS from a virtual machine running microservice $\mathcal{Y}$, which is the downstream microservice that processes the outputs of $\mathcal{X}$. The third metric is the average CPU usage of all virtual machines of the ELB service. We can observe that the variation tendencies between the first and second metrics are somewhat consistent during the anomaly-free period. The third metric is used to monitor the resource usage of the service and can raise alerts when there are resource overload issues. However, with the existence of load balance [47], a sudden spike in CPU usage will not necessarily lead to a performance issue. Thus, if we trigger alerts based on pre-defined thresholds (as the red dashed line shows), many false alarms will be reported and cause an alert storm that aggravates the burden of

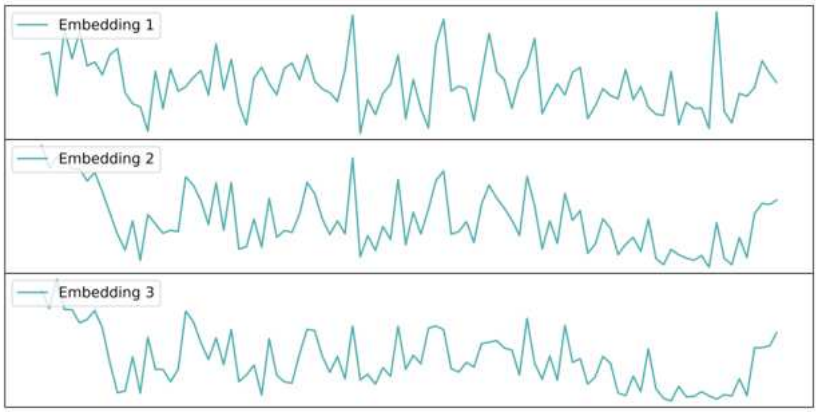


Figure 4.2: The latent embedding of the graph attention layer

engineers [244].

The red areas in Figure 4.1 denote a confirmed network congestion issue by engineers. This issue is caused by a network device failure affecting communication between virtual machines running microservices $\mathcal{X}$ and $\mathcal{Y}$. As a result, the NIC receiving PPS from the virtual machine running $\mathcal{Y}$ suddenly drops even if the NIC receiving PPS from the virtual machine running $\mathcal{X}$ increases. In this case, the correlation between the first and second metrics is critical to rapidly identifying this issue.

Some of the typical performance issues due to correlation violation are listed in Table 4.1. Specifically, we take the memory leak as another example to illustrate how the correlation violation reflected on system monitoring metrics can imply performance issues. In cloud service systems, efficient memory management is of great importance and has a direct impact on both the scalability and the operational costs of the cloud environment [139]. A crucial part of the memory management strategy is using slab memory allocation [88], which are pre-allocated extents in persistent memory and containers of free blocks [33] that aid in efficient memory usage. While the garbage collection manages application memory automatically in virtual machines [34], works to obtain the lifetime of the data objects and then allocates and releases memory space ac-

Table 4.1: A List of Typical Performance Issues Caused by Correlation Violation

Performance Issues	Metrics	Description
Disk Read/Write Delay	CPU I/O wait time Disk I/O	High CPU I/O wait time with low Disk I/O indicates potential disk failure or disk controller problems
Memory Leak	Memory slab size Garbage collection time	High memory slab size with high garbage collection time indicates potential memory leak
API Server Issue	API requests CPU usage	High API requests processed with low CPU usage indicates a potential problem with the API server or an overly efficient request handling mechanism
Network Congestion	NIC A receiving PPS NIC B receiving PPS	High NIC A receiving PPS with low receiving PPS of NIC B or vice versa can indicate potential network congestion
Hardware Interrupt Issue	Interrupt requests CPU usage	High CPU interrupt requests with low CPU usage indicate a potential hardware issue or inefficient interrupt handling
Software Interrupt Issue	Soft interrupt requests CPU usage	High CPU soft interrupt request with low CPU usage indicates potential software issue or inefficient interrupt handling
Network Buffering Issue	NAT gateway PPS Memory usage	High NAT gateway PPS with low memory usage indicates a potential efficient network buffering mechanism or a network issue
Database Indexing Issue	Database query Memory usage	Slow database query latency with low memory usage indicates potential issues with database indexing or query optimization

cordingly [176]. Memory leaks occur when a program fails to track and release allocated memory back to the system after it's no longer needed [207]. Usually, it is observed that garbage collection time or slab size increases under heavy workloads. An increase in slab size, with a relatively stable garbage collection time, could indicate that the system is effectively managing memory by creating new slabs to handle the increased workload. Similarly, a rise in garbage collection time, with a relatively stable slab size, might suggest that the system reuses existing memory efficiently, resulting in more objects being created and, hence, more garbage to collect. Generally speaking, software systems should efficiently manage memory un-

der heavy workloads, either by creating new memory blocks (indicated by increased slab size) or working harder to clear up unused objects (indicated by increased garbage collection time). Thus, these two cases are not typically considered performance anomalies. However, when both the slab size and garbage collection time consistently increase, even when the workload has reduced, it could suggest a memory leak because memory is being allocated faster than it's being released. It should be noted that a heavy workload can cause these two metrics to increase, hinting at a memory leak when such a leak exists. However, a heavy workload itself will not cause a memory leak, as the root cause of memory leaks lies in programming errors.

It is worth noting that a straightforward way to identify this issue is to build a new metric that combines the first metric and the second metric. However, since a cloud service system typically has a variety of metrics, it is infeasible for engineers to design such combined metrics comprehensively. To alleviate this problem, some GNN-based methods like Graph Attention (GAT) attempt to embed these correlations into latent representations. Nevertheless, we have identified that these methods incorporate correlation information into the latent feature representation implicitly through the combination of all metrics with attention weights. Then the temporal anomalies in these latent representations are detected. The GAT embedding of the example in Figure 4.1 is shown in Figure 4.2, where we can observe that the correlation violation between the first and second metrics is even impaired, as the uptrend of the first metric and the downtrend of the second metric are counteracted. Thus, it is crucial to isolate the correlation from the temporal information to detect violation-related performance anomalies.

4.3 METHODOLOGY

In this section, we present ISOLATE, an approach for identifying performance issues based on multivariate metrics in cloud systems. First, we will give a formal definition of the problem. Then, we introduce the overview of ISOLATE and illustrate the design details of our proposed relational-temporal embedding, Label-Conditioned Variational Auto-encoder (LC-VAE), and a novel positive unlabeled strategy. Eventually, we will introduce how to localize the problematic metrics with the correlations obtained in the previous stage.

4.3.1 Problem Formulation

Our objective is to identify performance issues from a variety of monitoring metrics by identifying when the performance anomalies happen and pinpointing the metrics that exhibit temporal or relational anomalies. Specifically, a group of metrics can be seen as a multivariate time series $X \in R^{N \times M}$, where N denotes the number of observations collected at an equal interval, *i.e.*, the length of a time series [182] and M is the number of metrics. $x_t = [x_t^1, x_t^2, ..., x_t^M]$ is an M-dimensional vector [76] that reflects the running status of the system at timestamp t. While the N-dimensional vector $x^k = [x_1^k, x_2^k, ..., x_N^k]$ is the k^{th} metric during the whole monitoring period. In addition, a sliding window of historical values $x_{t-c:t}^k = [x_{t-c+1}^k, x_{t-c+2}^k, ..., x_t^k]$ is used for modeling the pattern of a current observation, where c is the length of the sliding window.

To determine whether there is an occurrence of performance issues at observation x_t, the anomaly score $s_t \in [0, 1]$ that represents the degree of being anomalous for each $x_{t-c:t}$ is calculated. Then, the anomaly result can be obtained by comparing the anomaly score against a pre-defined threshold θ. If $s_t > \theta$, the approach will predict the observation x_t as an anomaly. However, it is still unclear to engineers how the anomaly happens. Thus, a kind of anomaly interpretation can be achieved through anomalous metrics localiza-

tion, *i.e.*, pinpointing a set of metrics $\{x_{k_1}, x_{k_2}, ...x_{k_r}\}$, that is helpful for engineers to find the monitored components that are related to anomaly by the degree of deviation of temporal pattern or break of correlation with other metrics, where r is the number of metrics that are recommended as the anomalous metrics.

Normalization is performed on each individual metric to unify the range of all metrics and improve the robustness of our model. We normalize the metrics with the maximum and minimum values first:

$$x^k_{pre} = \frac{x^k - min(x^k)}{max(x^k) - min(x^k)} \tag{4.1}$$

Where $max(x^k)$ and $min(x^k)$ represent the maximum and the minimum value of the training set, are computed within the training data and will be used for testing data. For simplicity, we omit the "pre" subscript in the following elaboration.

4.3.2 Overview

We propose ISOLATE, an automated method that learns correlations among metrics, detects performance anomalies, and locates anomalous metrics. The overview of ISOLATE is shown in Figure 4.3, which contains two main parts: the relational-temporal embedding part and the performance issue identification with the LC-VAE part. Specifically, since both abnormal temporal patterns and correlation violations between metrics can indicate performance issues, in the relational-temporal embedding part, ISOLATE captures the relational and temporal patterns from the original metrics (Section 4.3.3). In particular, due to the lack of information about the correlation among metrics, a complete graph is constructed, then ISOLATE employs graph attention to extract the correlation among metrics. ISOLATE also captures the temporal pattern of each metric through GRU and temporal convolution. In the performance is-

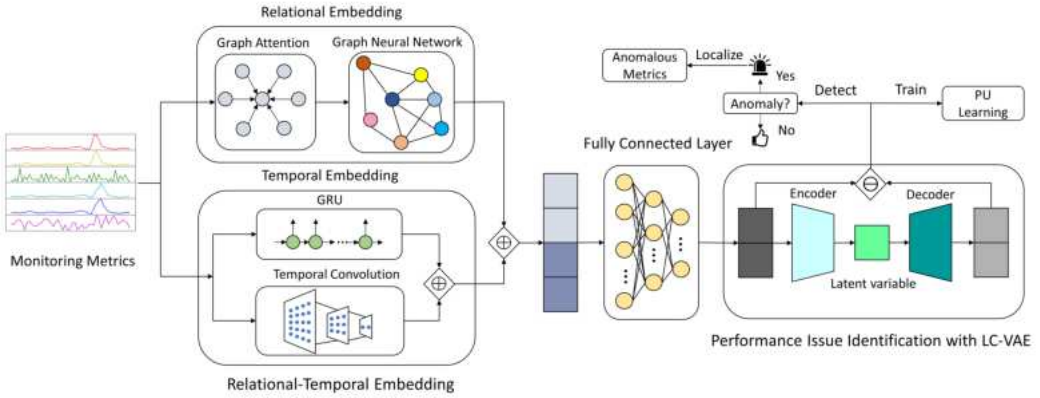


Figure 4.3: The Overview of the Proposed Method ISOLATE

sue identification part, given the inevitable presence of background noise within the data, we propose to use the positive unlabeled learning (PU Learning) strategy during the training phase of ISOLATE (Section 4.3.4), which identifies positive samples in unlabeled training data, avoiding the impact of noisy data. As PU learning leads to pseudo-labeled data, a novel label-conditional-VAE (LC-VAE) is adopted to distinguish anomalies from normal patterns (Section 4.3.4). Unlike existing VAE-based methods that solely learn from normal samples, the LC-VAE can learn features from normal and abnormal samples, achieving better performance. Upon the detection of an anomaly, the correlation learned from relational-temporal embedding can aid in localizing the correlation-violating metrics (Section 4.3.5).

4.3.3 Relational-Temporal Embedding

The relational-temporal embedding part takes a group of metrics as inputs. Relational embedding is designed to extract the intrinsic dependencies between metrics and embed the dependencies as a feature vector. Temporal embedding is used to obtain the temporal patterns of metrics as another feature vector because metrics are time

series.

Relational Embedding

Specifically, for multivariate metrics with size $M \times N$, we can treat each metric $x_i, (i = 1, 2, ...M)$ as a feature vector. The correlations between nodes can be depicted by an adjacency matrix $A \in R^{M \times M}$. Since we don't have prior knowledge about the correlation between different metrics, we should construct a fully connected graph. We then adopt graph attention networks (GAT) [17] to learn the correlation between metrics. The attention score is calculated as follows:

$$a_{ij} = \frac{\exp(p^T \text{LeakyReLU}(w \cdot (x_i \oplus x_j)))}{\sum_{k=1}^{M} \exp(p^T \text{LeakyReLU}(w \cdot (x_i \oplus x_k)))} \tag{4.2}$$

The symbol $\oplus$ represents the concatenation operator between two metrics x_i and x_j, $w \in R^{2N \times d}$ is a matrix of learnable parameters to aggregate the two features. After a nonlinear activation function LeakyReLU [214], another learnable vector $p \in R^d$ is applied. To make the training process more robust and reduce the impact of noise, a threshold α is set to make the adjacency matrix a sparse binary matrix. In other words, the edge between two nodes will be removed if the attention score is below the threshold α. Then, a widely used graph convolution layer [98] is adopted. The formula of graph convolution is shown as follows:

$$\hat{h}^{(l+1)} = \sigma(\widetilde{D}_l^{-\frac{1}{2}} \widetilde{A}_l \widetilde{D}_l^{-\frac{1}{2}} h^{(l)} \Theta_l) \tag{4.3}$$

where σ is the ReLU activation function and $\widetilde{A}_l = A_l + I$ is the adjacency matrix at the layer l, $\widetilde{D} \in R^{M \times M}$ is the degree matrix of $\widetilde{A}_l$, $h^{(l)}$ is the output representation of the hidden layer l and $\Theta_l \in R^{M \times F}$ is a learnable parameter. Due to the limitation of space, we only show one layer in Figure 4.3.

We further apply graph pooling layers between the graph convolution layers to reduce the number of parameters, which can also avoid overfitting. Specifically, as proposed by [104], self-attention [193] is utilized to focus more on important features and less on unimportant features. Thus, self-attention scores can be obtained by using another graph convolution. After obtaining the attention scores Z, a portion of the nodes and features will be reserved according to the scores. A hyperparameter k refers to the pooling ratio, and the corresponding nodes with the top $\lfloor kM \rfloor$ value of Z will be retained.

Readout layer [18] is useful to aggregate all node features and get a summarized representation, which is used to output the relational embedding. The output of the readout layer is as follows:

$$r_l = \frac{1}{N_l} \sum_{n=1}^{N_l} h_n^{(l)} \oplus \max_{n=1}^{N_l} h_n^{(l)} \tag{4.4}$$

where N_l denotes the node number of layer l, $h_n^{(l)}$ is the n^{th} node feature of $h^{(l)}$ and $\oplus$ is concatenation operator. The outputs of each layer will go through readout layers and will be added up as the output of the relational embedding module, because the features of different graphs with different sparsities can be combined.

Temporal Embedding

Existing metrics anomaly detection works [76, 242] utilize LSTM to acquire sequential information as Long-term temporal dependency inherently exists in monitoring metrics [74]. However, LSTM suffers from the gradient vanishing problem incurred by long-time lags [42]. To overcome the drawbacks of LSTM, we apply a GRU to capture the sequential information t_g, especially the long-term pattern in the metrics. Then, global average pooling layers are applied on the time series dimension of the output of GRU to get the temporal embedding.

Temporal convolution is useful for capturing the multi-scale temporal information of metrics. Unlike the existing methods that embed metrics with only recurrent neural networks, we also deploy causal convolution implemented by shifting the output of the 1D convolution. To further increase the receptive field of the convolutions, we use dilated convolutions. Dilated convolution is equivalent to filling the convolution kernel with zero padding so as to get a larger convolutional filter. Thus, we adopt dilated causal convolution (DC convolution) [153] to extract the temporal embedding of the metrics as it has advantages over the original convolutional operation with a larger receptive field, which is beneficial to our temporal embedding module as it can capture the behaviors of monitoring metrics at multi-scale. A block that consists of DC convolution, batch normalization [81], and activation function (*i.e.* ReLU) is utilized to form the temporal convolution. Eventually, the temporal embedding will be concatenated to the relational embedding and go through a fully connected layer to get the relational-temporal embedding.

4.3.4 Performance Issues Identification with LC-VAE

With the extracted relational-temporal embedding, we then use a novel Label Conditional VAE to detect performance issues. Unlike traditional autoencoder-based methods [15, 182, 215, 219, 255] that only take normal samples as input to capture the distribution of metrics, our proposed Label Conditional VAE takes the label obtained from positive unlabeled learning as a part of the input to further help the model differentiate anomalies from normal data because there exist some mild performance issues that are ignored by engineers in training data.

Positive Unlabeled Learning

Unsupervised methods assume that the training data is anomaly-free. However, there are inevitably some unlabeled anomalies that are ignored by engineers. This assumption can lead to a decline in the overall accuracy as the presence of these hidden anomalies can skew the model's understanding of normal behavior. In our scenario, we only have a small portion of negative data (representing normal, anomaly-free metrics) with high confidence, but we don't have positive data (representing anomalous samples) because finding positive from a large number of unlabeled samples is like looking for a needle in the ocean.

To tackle this, ISOLATE tries to find out the anomalous samples using the idea of positive unlabeled learning (PU learning) [99]. Specifically, as shown in Figure 4.4, a small amount of negative samples (around 5%) labeled by engineers is utilized for training the model. These labels are obtained by initially randomly selecting 5% of the entire training dataset. Engineers then meticulously filter out the noise in these samples and label them as true negatives. This manual labeling process is feasible due to the high confidence in the selected data and can be completed within a few minutes. Consequently, we incorporate human expertise into our method.

With the model trained on these labeled negatives, the remaining unlabeled training data can be predicted. Intuitively, the anomalous samples concealed in the training data are hard to reconstruct with this model and, thus, have a high anomaly score. After obtaining the anomaly score, the samples with an anomaly score that exceeds a pre-defined threshold β will be labeled as positive. All the data with pseudo labels are used to update the model. Finally, this updated model will be used to detect performance issues from metrics.

Label-Conditional VAE

Though the posterior of the distribution $p_\theta(z|y,e)$ is critical for training and prediction of the model, it is hard to obtain. Thus, the variational inference is used to fit a neural network as the approximation posterior $q_\phi(z|y,e)$. Suppose the prior of the latent variable Z is Gaussian distribution $\mathcal{N}(\mathbf{0},\mathbf{1})$ and y is the true label of the input sliding windows. Then both posteriors of e and z are chosen to be Gaussian distribution: $p_\theta(e|y,z) = \mathcal{N}(\mu_\theta(z), \sigma^2_\theta(z))$ and $q_\phi(z|y,e) = \mathcal{N}(\mu_\phi(e), \sigma^2_\phi(e))$, where $\mu_\theta(z)$, $\sigma_\theta(z)$ and $\mu_\phi(e)$, $\sigma_\phi(e)$, are the means and standard deviations of input embedding and latent variable. The input embedding e will be concatenated with the one-hot label vector y, and then the latent variable z will be sampled from a posterior $q_\phi(z|y,e)$ at the encoder, which is usually derived by linear layers [97]. Eventually, the latent variable will also be concatenated with y, and the input embedding will be reconstructed from $p_\theta(e|y,z)$ at the decoder. The reconstructed embedding can be denoted as $\hat{e}$.

In general, the parameters of the LC-VAE can be estimated efficiently with the stochastic gradient variational Bayes (SGVB) algorithm [165]. The evidence-lower bound (ELBO) is a surrogate objective function that can help the estimation. Besides, to better capture the latent pattern of normal sliding windows, we add an additional reconstruction error term. The loss function of the proposed LC-VAE is shown as follows:

$$\begin{aligned}\mathcal{L}_{LCVAE} =& sgn(0.5-y) * (-KL(q_\phi(z|y,e)\|p_\theta(e|y,z)) \\ &+ \frac{1}{S}\sum_{s=1}^{S}(\log p_\theta(e_s|y) + \lambda \cdot \|e_s - \hat{e_s}\|_2))\end{aligned} \tag{4.5}$$

Where S is the number of sliding window samples, the first two terms are from evidence-lower bound (ELBO), and the third term is the reconstruction error of the embedding. In this way, we combine the strength of reconstruction and probabilistic estimation together.

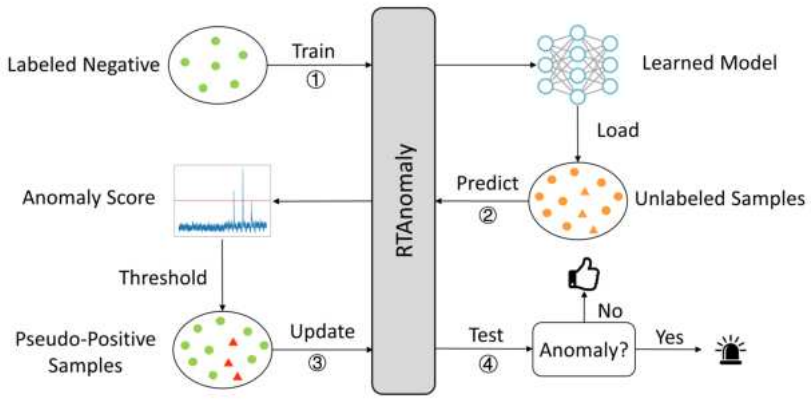


Figure 4.4: Positive Unlabeled Learning

The coefficient $\lambda > 0$ is to trade off the loss terms. As mentioned above, metrics anomaly detection works by learning the normal patterns of sliding windows of metrics; thus, we should minimize the loss function for a coming normal sliding window. However, when there is an anomalous sample input, we should avoid the model to learn the pattern of anomaly by maximizing the loss function. Thus, we denote the loss function with the Signum function to control the sign. In this way, during the detection phase, the anomalies are easy to differentiate by ISOLATE. The reconstruction error $\|e_s - \hat{e_s}\|_2$ will be used as the anomaly score during the detection phase.

4.3.5 Correlation Violation Metrics Localization

Once an anomaly has been detected in a service system, further analyses will be enacted to determine the possible causes for such a performance anomaly [115, 171, 251]. This allows application operators to determine which part of the service this performance issue reported is related to. For monitoring metrics, to further understand the mechanism of performance issues, pinpointing a few metrics that are highly correlated with the root cause is crucial [224, 226]. For example, when we observed that the throughput metrics of two devices were highlighted, it seemed to be a performance issue related

to the communication between two devices. While our model highlights the CPU utilization of a service, it is likely to occur due to a lack of computing resources in the run-time environment [152].

However, existing methods regarding monitoring metrics have not integrated anomaly detection and metric localization, namely root cause localization together in a unified pipeline [181]. In this case, the knowledge during the anomaly detection phase cannot be shared with the localization. We integrate these two closely related tasks into a unified framework to provide more hints to system operators.

Since there exist correlations between metrics of service in modern online service systems, the learned correlation graph between metrics during the anomaly detection phase is useful for localizing the metrics that reveal the cause of the anomaly [135, 198, 208]. Regarding localizing anomalous metrics, correlation information can be more indicative than temporal information. It is straightforward to understand that no matter whether the anomaly is a temporal anomaly that happens on some specific metrics or an anomaly due to the contravention of correlation compared with the anomaly-free stage, the correlation between anomalous metrics and others will undergo drastic changes. However, when some metrics collectively spike, they do not violate the correlations and thus are not anomalous metrics. So, we only utilize the correlations in metrics localization. Particularly based on the above assumption, we can calculate the correlation change as follows:

$$\Delta A_i = \sum_{j \neq i} \| A^a_{ij} - A^n_{ij} \|_1 \tag{4.6}$$

Where ΔA_i is the variation of correlation between normal and abnormal periods for metric i, the A^n_{ij} is computed by averaging a_{ij} on the training period, while the A^a_{ij} is the mean of a_{ij} during the anomaly segment. Eventually, ISOLATE would highlight a few metrics with high ΔA_i and recommend them to engineers to help them

Table 4.2: Statistics of Industrial Dataset

Industrial	Dataset A	Dataset B
Services	21	31
Metrics	4∼23	3∼25
Train Length	366,513	541,043
Test Length	244,356	360,716
Anomaly Ratio	6.71%	5.88%

get fine-grained information on the performance issue and double-check the devices related to these metrics.

4.4 EVALUATION

To comprehensively evaluate the effectiveness of our proposed approaches ISOLATE, we use both a public dataset and two real-world monitoring metric datasets from the online services of Huawei Cloud Company. Particularly, we aim to answer the following research questions (RQs):

- RQ1: How effective is ISOLATE compared with performance anomaly detection baselines?
- RQ2: How effective is each component of ISOLATE in performance anomaly identification?
- RQ3: How effective is ISOLATE in localizing the anomalous metrics?
- RQ4: How sensitive is ISOLATE to the parameters?
- RQ5: How efficient is ISOLATE regarding the number of metrics?

4.4.1 Datasets

We conduct experiments on a publicly available dataset. To confirm the practical significance of ISOLATE, we collect two datasets from

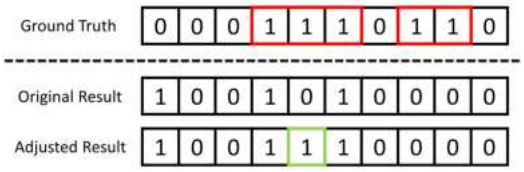


Figure 4.5: An Illustration of the Point Adjustment Process in Our Evaluation

large-scale online services of Huawei Cloud. The statistics of our industrial datasets are shown in Table 4.2

Public Dataset The public dataset for our experiments is SMD (Server Machine Dataset), which is collected from a large Internet company containing a 5-week-long monitoring metrics of 28 machines [182]. The authors divided the SMD into two subsets of equal size: the first half for the training set and the second half for the testing set. Domain experts labeled anomalies in the SMD testing set based on incident reports.

Industrial Dataset To evaluate the effectiveness of ISOLATE in production scenarios, we collect metrics Application CPU Usage, Memory Usage, Interface Throughput, and so on from the online service of the company. We collect metrics with a sampling interval of one minute for more than one week from two regions of the company. The anomalies representing the performance issues of the service are labeled by experienced software engineers with incidents associated with the metrics. The performance issues consist of correlation-violated issues like memory leaks or network congestion listed in Table 4.1 and non-correlation-violated issues like resource overload. Based on the incident reports, engineers also label the metrics that are correlated to the performance issues. Using these labels, we can also evaluate the accuracy of metrics localization of ISOLATE.

4.4.2 Experiment Setting

Baselines

The following methods are compared to evaluate the effectiveness of ISOLATE. All the baselines are implemented from the open-sourced codes released by the authors. For both the public dataset SMD and our industrial datasets in Huawei Cloud, we employed a grid search strategy to explore the most suitable parameter configurations. A summary of the baseline methods is shown in Table 4.3.

Table 4.3: A Summary of the Baseline Methods

Method Category	Supervision Type	Reference Baseline Methods
Signal Processing-based	Unsupervised	JumpStarter [136]
Machine Learning-based	Unsupervised	LOF [16], IForest [121], OCSVM [168]
LSTM-based	Unsupervised	LSTM [76, 242], LSTM-VAE [157], THOC [174]
Autoencoder-based	Unsupervised	DAGMM [255], OmniAnomaly [182], ACVAE [240] TranAD [189], MTSAD [201], MSCRED [231]
	Semi-supervised	SLA-VAE [73]
Graph Neural Network-based	Unsupervised	MTAD-GAT [243], GDN [35], GTA [22], FuSAGNet [59]

- *OCSVM* [168]. OCSVM is a clustering-based anomaly detection method that learns the boundary for the normal data points and identifies the data outside the border as anomalies. The input in this baseline is the observation of time series at each timestamp and the output is the anomalous timestamps.
- *IForest* [121]. Isolation Forest ensembles a number of isolation trees and recursively partitions the feature space to detect anomalies. The samples with awfully shorter heights are likely to be anomalies. In this baseline, the input is the observation at each timestamp and the output is the anomalous timestamps.
- *LOF* [16]. Local Outlier Factor (LOF) is based on density estimation that calculates the local density deviation of a given sample with respect to its neighbors. The anomalies have a substantially

lower density than their neighbors. The input is the observation at each timestamp and the output is the anomalous timestamps.

- *DAGMM* [255]. DAGMM is a model that utilizes an autoencoder to generate a low-dimensional representation and a Gaussian Mixture Model to go through a probabilistic estimation to obtain the anomaly score.
- *LSTM* [76, 242]. LSTM neural network captures the normal behaviors of metrics by forecasting the next values of metrics based on historical observations. Anomalies will be reported if the differences between predicted values and real values exceed a predefined threshold
- *LSTM-VAE* [157]. LSTM-VAE detects anomalies by integrating LSTM and VAE. It projects observations at each timestamp into a latent space and then estimates the distribution of it using VAE.
- *OmniAnomaly* [182]. OmniAnomaly is a model that captures the normal patterns by learning robust representations of metrics with stochastic Recurrent Neural Network (RNN) and planar normalizing flow based on the reconstruction error.
- *THOC* [174]. THOC is a model that captures the multi-scale temporal features from dilated recurrent layers by a hierarchical clustering mechanism and detects the anomalies by the multi-layer distances.
- *MTSAD* [201]. MTSAD is a deep unsupervised anomaly detection model that incorporates the strength of active learning, including three feedback strategies, namely denominator penalty, negative penalty, and metric learning.
- *MSCRED* [231]. MSCRED utilizes convolutional LSTM layers to capture the temporal information and embed the inter-metric information through a signature matrix. It detects anomalies with reconstruction errors of the input metric.

- *MTAD-GAT* [243]. MTAD-GAT is a graph attention-based model that captures both feature and temporal correlations. It passes these correlations to a Gated-Recurrent-Unit (GRU) network for reconstruction and forecast.
- *GDN* [35]. GDN learns the graph of relationships between metrics through graph structure learning. It then uses attention-based forecasting and deviation scoring to output anomaly scores.
- *GTA* [22]. GTA is a transformer-based model that employs graph structure learning to learn the relationship among multiple IoT time series, and the Transformer for temporal modeling and the reconstruction error for anomaly detection.
- *TranAD* [189]. TranAD is a transformer-based model that detects anomalies with reconstruction error by incorporating attention mechanisms and adopting adversarial training.
- *SLA-VAE* [73]. SLA-VAE is a semi-supervised VAE-based anomaly detection model for online service systems, which employs active learning to update the model.
- *ACVAE* [240]. ACVAE is a self-adversarial variational autoencoder combined with a contrast learning mechanism that allows the encoder to obtain more training samples.
- *FuSAGNet* [59]. FuSAGNet jointly optimizes reconstruction using a sparse autoencoder and forecasting using a graph neural network. It captures the interdependencies between time series in sensors.
- *JumpStarter* [136]. JumpStarter is a compressed sensing-based method combined with shape-based clustering and an outlier-resistant sampling algorithm. This combination ensures a shorter initialization time.

Evaluation Metrics

The anomaly detection problem is modeled as a binary classification problem, so the widely-used binary classification measurements can be applied to evaluate the performance of models. We employ Precision: $PC = \frac{TP}{TP+FP}$, Recall: $RC = \frac{TP}{TP+FN}$, F1 score: $F1 = 2 \cdot \frac{PC \cdot RC}{PC+RC}$. Specifically, TP is the number of abnormal samples that the model correctly discovered; FP is the number of normal samples that are incorrectly classified as anomalies; FN is the number of anomalous samples that failed to be detected by the model. F1 score is the harmonic mean of the precision and recall, which symmetrically represents both precision and recall in one metric. Following [7], we get the anomaly threshold by grid search for all baselines and ISOLATE to evaluate the performance.

In real-world applications, anomalies will last for a while, leading to consecutive anomalies in the monitoring metrics. Therefore, it is acceptable for the model to trigger an alert for any point in a contiguous anomaly segment if the delay is within the acceptable range. Thus, we adopt the evaluation strategy following [164, 182, 189] that marks the whole segment of continuous anomalies as an anomaly. In other words, we consider the model to correctly predict an anomalous segment if at least one timestamp is successfully predicted as an anomaly. An example of the point adjustment strategy is shown in Figure 4.5. The first anomaly segment is treated as correctly predicted, so the second point of the segment is adjusted to correctly predicted as anomalous.

Implementations

We run all the experiments on a Linux server with Intel Xeon Gold 6140 CPU @ 2.30GHZ and Tesla V100 PCIe GPU. The proposed model is implemented under the PyTorch framework and runs on the GPU. The hidden sizes of the GAT layer, GRU layer, and temporal convolution layers are 256, 128, and 128. The coefficient of loss

Table 4.4: Experimental Results of Different Anomaly Detection Methods.

Methods	SMD			Dataset A			Dataset B		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
OCSVM	0.4434	0.7672	0.5619	0.8639	0.5491	0.6446	0.8979	0.5955	0.6547
IForest	0.4231	0.7329	0.5364	0.9375	0.5782	0.6726	0.9443	0.6713	0.7431
LOF	0.5634	0.3986	0.4668	0.9218	0.5744	0.6693	0.9583	0.4302	0.6160
DAGMM	0.5951	0.8782	0.7094	0.7514	0.8108	0.7792	0.8112	0.9073	0.8421
LSTM	0.7855	0.8528	0.8178	0.9177	0.8107	0.8366	0.9166	0.6994	0.7665
LSTM-VAE	0.8698	0.7879	0.8083	0.8753	0.7443	0.7936	0.7969	0.7714	0.7839
OmniAnomaly	0.8368	0.8682	0.8522	0.8769	0.9084	0.8892	0.9437	0.7985	0.8607
THOC	0.7976	0.9095	0.8499	0.9502	0.8022	0.8347	0.9613	0.8400	0.8866
MTSAD	0.8745	0.9395	0.9042	0.8719	0.9171	0.8950	0.9377	0.8765	0.9026
MSCRED	0.7876	0.9374	0.8434	0.8928	0.8451	0.8570	0.9554	0.8522	0.8838
MTAD-GAT	0.8210	0.9215	0.8683	0.8519	0.9019	0.8682	0.9329	0.8769	0.9012
GDN	0.7670	0.9362	0.8312	0.8831	0.9109	0.9018	0.9614	0.8662	0.8994
GTA	0.8768	0.8987	0.8822	0.8671	0.9098	0.8784	0.9431	0.8791	0.8554
TranAD	0.8882	0.9023	0.8953	0.9275	0.8703	0.8867	0.8816	0.8929	0.8874
SLA-VAE	0.8672	0.9371	0.9019	0.9523	0.8605	0.8975	0.9361	0.8859	0.8937
ACVAE	0.8779	0.8384	0.8612	0.8988	0.8405	0.8645	0.9151	0.8432	0.8563
FuSAGNet	0.8319	0.9473	0.8736	0.9011	0.8736	0.8790	0.9038	0.8941	0.8978
JumpStarter	0.8913	0.9174	0.9063	0.8427	0.7959	0.8295	0.8697	0.7869	0.8435
ISOLATE	**0.8998**	**0.9745**	**0.9346**	**0.9871**	**0.9435**	**0.9497**	**0.9759**	**0.9367**	**0.9514**

function λ is 0.5. The dimension of the latent variable in LC-VAE is 10. The threshold of positive learning is 0.9. We train ISOLATE with the Adam optimizer [96] with a learning rate of 0.001, a batch size of 128, and an epoch number of 50. We have released the artifacts and data for future research purposes on https://github.com /WenweiGu/ISOLATE.

4.4.3 Experimental Results

RQ1 The effectiveness of ISOLATE

To answer this research question, we compare the performance of ISOLATE with other state-of-the-art baselines on a public dataset and two industrial datasets. First, We train ISOLATE on a small portion of negative samples. Then, ISOLATE will assign pseudo labels to the remaining training data according to the anomaly score and threshold β. During the test phase, the anomaly score for each timestamp will be computed. The anomaly threshold will be searched

Table 4.5: Experimental Results of the Ablation Study

Methods	SMD			Dataset A			Dataset B		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
ISOLATE *w/o* $\mathcal{RT}$	0.8512	0.9079	0.8723	0.9791	0.8285	0.8738	0.9465	0.8879	0.9035
ISOLATE *w/o* $\mathcal{PU}$	0.8730	0.9407	0.9062	**0.9912**	0.8405	0.8869	0.9396	0.9189	0.9231
ISOLATE	**0.8998**	**0.9745**	**0.9346**	0.9871	**0.9235**	**0.9497**	**0.9759**	**0.9367**	**0.9514**

following [7] to produce the prediction result.

The results are shown in Table 4.4, where the best F1 scores are marked with boldface. We can see the average F1 score of ISOLATE outperforms all baseline methods in three datasets. The experimental results are shown in Table 2. In Dataset B, the improvement achieved by ISOLATE is more significant as the metrics correlations between metrics in Dataset B are more complicated and the ratio of performance anomalies caused by correlation violation is higher. Generally speaking, ISOLATE 's good performance can be attributed to two reasons: Firstly, the utilization of relational-temporal embedding, as the anomalies can be caused by the violation of correlation, which can hardly be detected by finding the spikes on a single metric, it can also help facilitate localization of the anomalous metrics. Thus, there is a significant improvement in the recall of ISOLATE compared to other baselines. Secondly, ISOLATE effectively learns potential anomalous samples from the training data, thereby mitigating the risk of overfitting anomalous patterns during the training process. Consequently, ISOLATE achieves remarkable precision, ranking among the best when compared to other baseline methods.

Typically, the baseline methods have higher precision than recall because there are some anomalies that are not very apparent. We can observe that machine learning-based methods, namely OCSVM, IForest, and LOF have relatively low performance compared with other baseline models since these methods learn the metrics pattern at each timestamp independently without considering the temporal dependency. While LSTM-based methods and autoencoder-based methods, including LSTM, DAGMM, and LSTM-VAE, per-

form notably better than OCSVM, IForest, and LOF and achieve 0.7094∼0.8421 F1 score because these models take the historical observation window of the data, that helps to retain valuable historical temporal pattern. Among the baselines, we can find OmniAnomaly, THOC, MTSAD, MSCRED, TranAD, ACVAE, and SLA-VAE can achieve relatively better performance (especially MTSAD can achieve an F1 score like 0.9042) because these methods introduce some mechanisms to extract temporal information of metrics and ensure robust anomaly detection. It should be noted that SLA-VAE is a semi-supervised VAE-based method and it employs active learning to update the model. However, our proposed method outperforms SLA-VAE, suggesting that the improved performance of our model is not solely due to the semi-supervised mechanism, *i.e.*, PU learning, but also the design of combining correlational violation detection with temporal information. Specifically, OmniAnomaly models the metrics through stochastic variables and uses reconstruction probabilities to determine anomalies. As for MSCRED, the temporal information is also captured through convolutional LSTM. Meanwhile, in MTSAD, active learning has been incorporated into a variational autoencoder to ensure capability against noise. SLA-VAE is also a variational autoencoder-based architecture that employs both active learning and semi-supervised learning. Adversarial training is incorporated in the VAE-based model ACVAE and the transformer-based model TranAD, which assures their robustness. In THOC, the complex nonlinear temporal dynamics of the system's normal behavior are captured. Though extracting temporal information well, the limitation of these approaches lies in not taking the correlation features into consideration, which is essential to successfully detecting anomalies from multivariate metrics. Graph Neural Network (GNN)-based approaches offer some mitigation to this issue through jointly extracting the relational and temporal information of raw metrics, *e.g.*, MTAD-GAT, GDN, GTA, and FuSAGNet. However, without explicitly capturing the corre-

lation violation, these approaches still achieve suboptimal performance compared with ISOLATE.

RQ2 The effectiveness of components in ISOLATE

To answer this research question, we conducted an extensive ablation study on ISOLATE. Particularly, we derive two baseline models based on removing the relational-temporal embedding and positive unlabeled learning parts of ISOLATE to investigate the contribution of these two components.

- *ISOLATE w/o $\mathcal{RT}$* This baseline is a variant of ISOLATE that removes relational-temporal embedding that captures both information of metrics. Instead, only an LSTM layer is utilized in this baseline to embed the temporal information of the raw metrics, which will be further fed into the CVAE.
- *ISOLATE w/o $\mathcal{PU}$* This baseline removes the positive unlabeled learning that finds anomalous samples from a large number of normal samples. All the training data are considered normal in this baseline.

Table 4.5 shows the experimental results of ISOLATE and its variants. Overall, relational-temporal embedding and positive unlabeled learning help to improve the effectiveness of ISOLATE as it performs the best, while the degree of contribution of relational temporal embedding is larger. We attribute this to the good capability of relational temporal embedding in extracting both the temporal information of each metric and correlational information between metrics. When an anomaly happens due to a breach of relationship during the anomaly-free period, it can be easily identified by ISOLATE, while other methods have difficulty identifying it as they are more effective on temporal outliers. We observe that even without PU learning, our model is not worse than all baselines, which further

Table 4.6: Performance on Metrics Localization

Methods	Dataset A		Dataset B	
	Hit@1	Hit@3	Hit@1	Hit@3
DAGMM	0.5714	0.6667	0.5806	0.7419
LSTM-VAE	0.6190	0.7142	0.6451	0.7741
ISOLATE-Cor	0.7419	0.8387	0.7142	0.8571
ISOLATE	**0.8095**	**0.9048**	**0.8387**	**0.9354**

demonstrates the effectiveness of explicitly extracting correlation violation.

The variant without relational-temporal embedding is similar to LSTM-VAE, which employs LSTM to extract the sequential information and VAE to differentiate the anomaly from normal. However, due to the design of positive unlabeled learning and conditional VAE, the variant can identify the noise of training data and label them as positive. Thus, the performances on three datasets of ISOLATE without relational-temporal embedding in terms of F1 score are improved compared to LSTM-VAE, respectively. We believe that in some scenarios with a higher ratio of noise, positive unlabeled learning would play a greater role in improving performance.

RQ3 The effectiveness of ISOLATE in localizing the anomalous metrics

To further demonstrate the capability of ISOLATE, experiments on localizing the metrics are conducted. Specifically, we localize the metrics by following four methods:

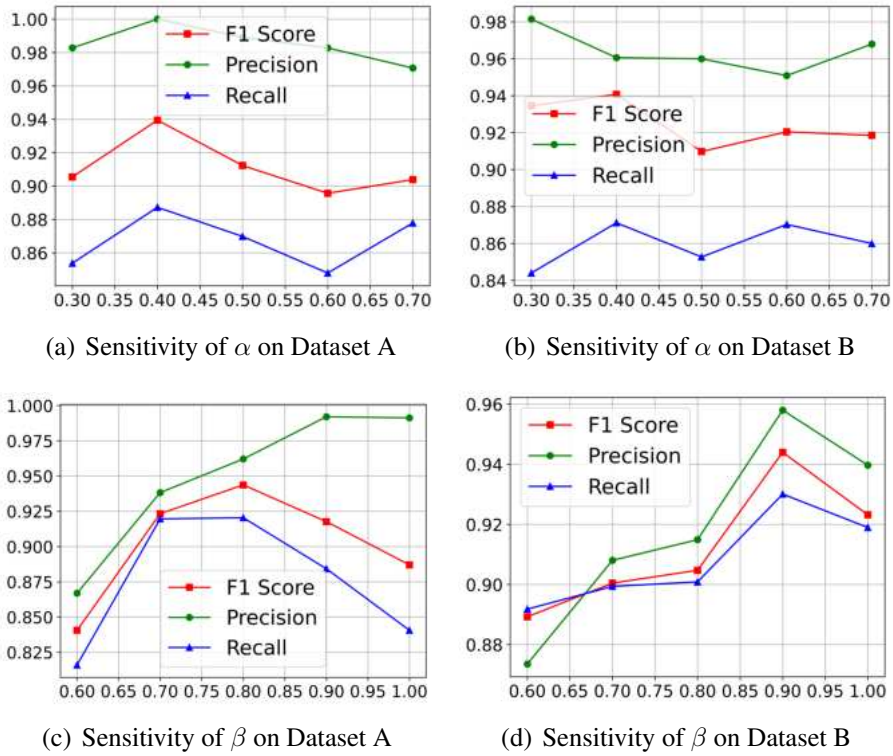
- *DAGMM.* This baseline uses the anomaly score of each metric output by DAGMM as the degree of the metric being anomalous. The metrics with the highest scores will be highlighted as anomalous metrics.
- *LSTM-VAE.* This baseline uses the anomaly score output by LSTM-VAE as the degree of being anomalous for the metrics.
- *ISOLATE-Cor.* This baseline sums up the correlation of a specific metric between other metrics as the degree of being the root

cause. Metrics with a high correlation with other metrics would be reported as anomalous metrics.

- *ISOLATE.* Different from ISOLATE-Cor, it uses the discrepancy between the anomaly-free period and anomaly period since the correlation between metrics may undergo drastic changes compared to the normal period when an anomaly happens.

As mentioned in Section 4.2, a metric that shows very abnormal behaviors compared to the normal period is not necessarily the most anomalous metric because it can have a small influence on other metrics and will self-heal. Using the correlation score itself can also cause inaccurate results because some metrics show consistently high correlations with others, no matter whether it is during an anomaly period or not. Indeed, the correlation difference between normal and abnormal time is a stronger indicator of anomalous metrics because when an anomaly happens, the correlation would be obeyed and cause a huge correlation change.

Table 4.6 presents the comparison of four methods, and we can observe that metrics localization using ISOLATE outperforms the other three baselines with a Hit@1 of 80.95%, 83.87%, and Hit@3 of 90.48%, 93.54% on two industrial datasets. Thus, our method can provide accurate hints for engineers on which part of the service system they can remedy to ensure the system's reliability. Compared to using the anomaly score of DAGMM and LSTM, using the correlation score to localize metrics (ISOLATE-Cor) is more effective as the anomaly scores of these two methods are computed on a single metric and are not aware of other metrics, while correlation considers the global information of metrics. Usually, the metrics that have a higher correlation with other metrics seem to play a greater influence on the service system and are more likely to be anomalous metrics when an anomaly happens.



(a) Sensitivity of α on Dataset A

(b) Sensitivity of α on Dataset B

(c) Sensitivity of β on Dataset A

(d) Sensitivity of β on Dataset B

Figure 4.6: The Sensitivity Analysis of Threshold α and β

RQ4 The sensitivity of ISOLATE to the parameters

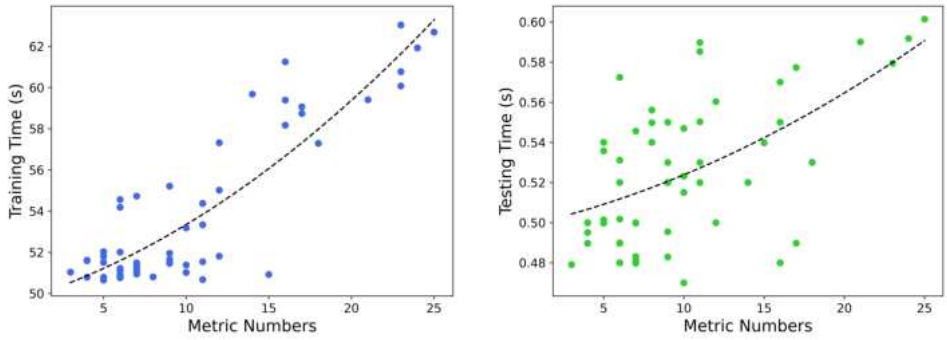
The threshold α is the parameter that determines the sparsity of the relational learning, while the threshold β, *i.e.*, the parameter that determines the label of training data during positive unlabeled learning, may affect the performance by affecting the label distribution of training samples. We hereon evaluate the sensitivity of ISOLATE to these two hyper-parameters on two industrial datasets. We change the value of α and β while keeping all other parameters unchanged in our experiments to guarantee fairness. Specifically, we choose the value of α in an appropriate range from 0.3 to 0.7 at a step of 0.1. The value of β is selected, ranging from 0.6 to 1 at a step of 0.1. When the value of β is 1, it is equivalent to the variant without positive unlabeled learning since all samples will be seen as normal. When the value of β is lower than 0.5, a large portion of samples

will be labeled as anomalous and introduce severe noise to training data, which is not the case in the real scenario.

Figure 4.6 presents the experimental results of RQ4. For the threshold α, the performance is relatively stable under different settings. It can be attributed to the distribution of learned graph attention scores, which are either close to 0 or 1. The polarization of graph attention scores means that the threshold chosen within the aforementioned range tends not to impact the performance in a significant manner, thus stable across different values of α. This makes ISOLATE easy to deploy in practice. For parameter β, a good threshold indeed helps improve the accuracy of our model. In dataset A, the best threshold is between 0.7 and 0.8, while in dataset B, the best threshold is between 0.8 and 0.9. This is because the anomaly ratio of Dataset B is slightly lower than Dataset A, so a lower amount of unlabeled anomalous samples exist in the training part. Compared to without positive unlabeled learning, a properly selected threshold would improve recall because some abnormal behaviors have been labeled as positive, and these abnormal patterns would be reported in the testing set, thus reducing false negatives.

RQ5 The efficiency of ISOLATE

In this section, we evaluate the efficiency of ISOLATE regarding the number of metrics. We perform our method on the industrial dataset A and B and record the training and testing costs. The training time is defined as the total time of feeding the preprocessed data to the model, and the testing time is the total time used to predict whether there are performance anomalies on all the sliding windows in the test set. All the data samples are trained with the same batch size for 20 epochs. The correlation between training/testing time and metrics number is shown in Figure 4.7, where we can observe that both of them are near quadratic correlations. In our scenario, the metrics collected for node-level and service-level performance anomaly detection are typically smaller than 100. According to the



(a) Training Time versus Metric Numbers (b) Testing Time versus Metric Numbers

Figure 4.7: The Efficiency Analysis of ISOLATE

polynomial approximation, the training and testing time for data that contains 100 monitoring metrics is 192.9 seconds and 3.89 seconds, respectively. Typically, model retraining will not be triggered more frequently than once per day due to the computation cost. Thus, the training time of 192.9 seconds is affordable for industrial deployment. Furthermore, since the monitoring metrics are typically collected at an interval of 1 minute, the testing time of 3.89 seconds in the online detection phase is enough for real-time performance issues identification. It should be noted that all the experiments are run on a Linux server with only one Tesla V100 PCIe GPU, however, the efficiency is even further enhanced when utilizing the significant computational power that can be harnessed from thousands of GPUs available in cloud service systems.

4.4.4 Case Study

To further demonstrate the effectiveness of ISOLATE, we conduct a case study concerning two industrial cases on identifying performance anomalies and localizing the anomaly on metrics that violate their intrinsic correlation. The first case is shown in Figure 4.8, which is the same example in Section 4.2.2. The first three rows show the monitoring metrics of the NIC receiving PPS from a vir-

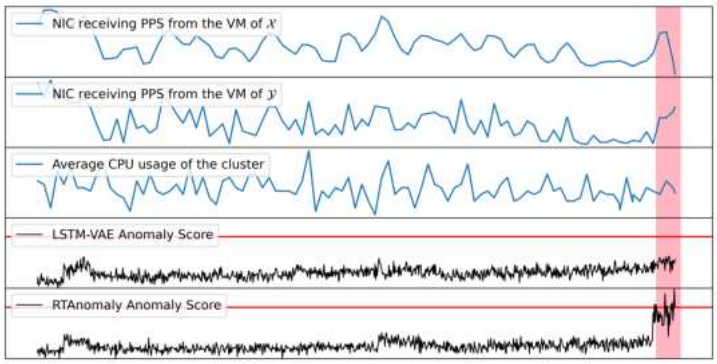


Figure 4.8: A case that ISOLATE finds false negative

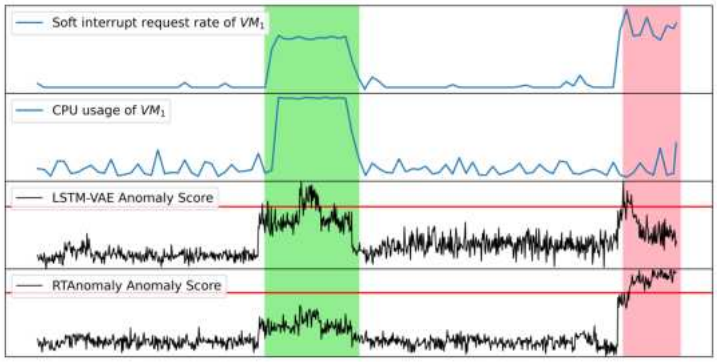


Figure 4.9: A Case that ISOLATE avoids false positive

tual machine running microservice $\mathcal{X}$, the NIC receiving PPS from a virtual machine running microservice $\mathcal{Y}$, and the average CPU usage of all virtual machines of the ELB service. The last two rows show the anomaly score of the baseline method LSTM-VAE and our proposed ISOLATE for comparison. This anomaly can hardly be discovered when observing each metric individually, as the correlation violation is critical to identifying this performance issue. However, our ISOLATE can find the false negative that is neglected by baseline models. The first and second metrics can also be highlighted to provide engineers with further insight into the performance issue.

Another example is shown in Figure 4.9. The first metric is the soft interrupt request rate of a virtual machine VM_1 running on an Elastic Cloud Server (ECS). The second metric is the CPU usage

of VM_1. We can observe that the green segment seems like a performance issue, as the soft interrupt request rate of both VM_1 and the CPU usage all encounter spikes. However, this is usually caused by increased user requests and should not be regarded as a performance issue. In contrast, since the correlations between these metrics are not violated, an anomaly alert should not be triggered. The red segment represents a soft interrupt issue because the CPU usage of VM_1 is not synchronized with the increase of the soft interrupt request rate of VM_1. Compared with the baseline method LSTM-VAE, ISOLATE can produce a significantly higher anomaly score in the true anomaly labeled with the red area, thus being more effective. In this case, though both ISOLATE and baseline methods like LSTM-VAE can detect the true anomaly, our ISOLATE can avoid sending a false alarm to engineers. This is helpful for the engineers to mend the service system because it prevents unnecessary interventions.

4.5 DISCUSSION

In this section, we share the success story of our deployment of ISOLATE in the industrial environment of the service system of Huawei Cloud and discuss the threats and limitations of our approach.

4.5.1 Industrial Experience

In this section, we share our experience in applying ISOLATE to the real-world cloud system of Huawei Cloud, a full-stack cloud system that consists of an infrastructure layer, a platform layer, and an application layer, aiming to demonstrate the practical usefulness of ISOLATE. Huawei Cloud serves hundreds of millions of cloud service tenants, offering them low-latency and high-performance services that span computing, storage, networking, and database solutions. Among them, ELB is a crucial service that is tasked with the

automatic distribution of incoming network traffic across a multitude of targets, including Elastic Cloud Server (ECS) instances, containers, and IP addresses across different Availability Zones (AZs). Relational Database Service (RDS) is another reliable and scalable managed DB service that frees up developers from handling time-consuming database administration tasks. In addition, the API Gateway Service (APIG), is an API management service that serves as a single point of entry into cloud systems, sitting between the user and a collection of backend services. It receives requests from an application user, routes the request to the appropriate services, gathers the appropriate data, and combines the results for the user in a single package. The Object Storage Service (OBS) provides stable, secure, efficient, and easy-to-use object storage. It enables storage and retrieval of any amount of data at any time, facilitating cloud storage for applications, big data analysis, and backup and archiving scenarios. The reliability of these services is among the most crucial concerns for Huawei Cloud and its tenants.

ISOLATE has been successfully incorporated into the performance issue detection system of large-scale online service systems in Huawei Cloud since *November 2022*, specifically, the overall pipeline of deployment is shown in Figure 4.10. The clusters in Huawei Cloud represent a collection of nodes that work together to provide various services, ensuring high availability, reliability, and scalability. Particularly, there are four types of nodes in Huawei Cloud: Management node, network node, compute node, and storage node, where the details of them are elaborated in Table 4.7. For clusters in cloud service systems, it has been a common practice for monitoring metrics used to profile the runtime status [82, 164, 218]. Thus, software reliability engineers usually collect tens of monitoring metrics (like CPU usage, network traffic, disk I/O, request rates, and memory usage) in nodes of the cluster through monitoring tools like Grafana, Prometheus, etc [2]. Then, these monitoring metrics are stored in the Data Lake of Huawei Cloud, a highly scalable and flexible storage

Table 4.7: A Summary of the Node Types in the Huawei Cloud

Node Type	Description
Management Nodes	Management nodes are used to deploy FusionSphere OpenStack, an enterprise-level platform to enhance computing, storage, network management, installation and maintenance, security, and reliability while supporting multiple infrastructure virtualization technologies at the resource pool layer. Management nodes use UVP as the host OS. The Computing cloud services, storage cloud services, network cloud services, common components, and management domain components are deployed on VMs.
Network Nodes	The network node uses the UVP as the host OS. The virtual Router, L3NAT, L3 service, and VPN components are deployed on VMs.
Compute Nodes	Compute nodes can be divided into two subtypes. The first is the KVM compute node for general-purpose Elastic Computing Services (ECS) in tenant VMs. The second is the KVM compute node for GPU-accelerated ECSs, which provision GPUs for deep learning jobs in tenant VMs. These two types of KVM compute nodes use the UVP as the host OS, and FusionSphere OpenStack (role compute) is also deployed.
Storage Nodes	Distributed storage nodes in Huawei typically support Elastic Volume Service (EVS). After the deployment of Huawei Distributed Block Storage, this node is utilized by the EVS service to provision EVS instances inthe tenant EVS disks.

system that consists of the Data Lake Storage, the Data Warehouse, and the Data Lake Governance Center (DGC). Data Lake Storage is the actual storage space where all the data, including the monitoring metrics, is stored, while the Data Warehouse is an enterprise system used for reporting and data analysis. On top of these two components, the DGC is responsible for managing the data stored in the data lake, which oversees the lifecycle of the data, from ingestion and storage to usage and deletion. With Huawei Cloud's data lake, real-time data analysis is enabled, *i.e.*, as soon as monitoring metrics are collected and stored in the data lake, they can be immediately accessed and analyzed by the performance issue diagnosis system empowered with ISOLATE. The results of ISOLATE provide not only the timing of a performance issue but also the metrics that are most related to this issue. Then, alerts will be triggered immedi-

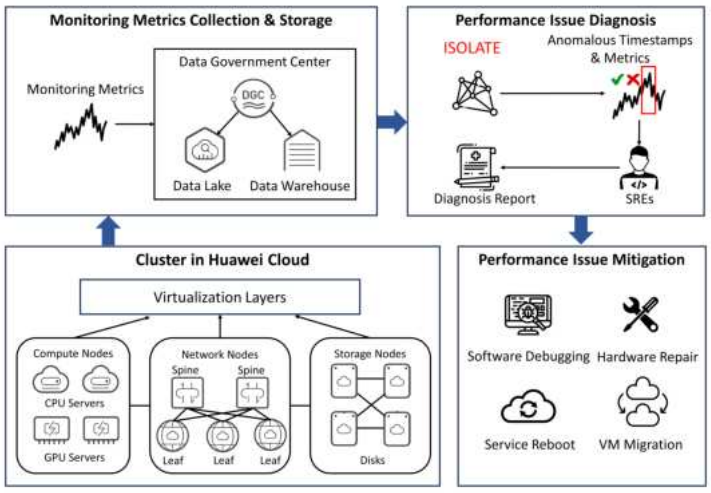


Figure 4.10: The Pipeline of Deploying ISOLATE in Huawei Cloud System

ately and sent to the SREs for further investigation. The SREs first inspect the alert and the associated metrics to understand the mechanism of the problem and find the root cause. Once the root cause has been identified, the SREs prepare a diagnosis report, including a detailed description of the performance issue, the associated metrics, the identified root cause, and potential mitigation strategies. For instance, if the performance issue is a memory leak, engineers may need to perform software debugging involving tracing the program's execution and implementing appropriate code fixes to prevent the leak from happening again. As another example, if the problem is network congestion, the mitigation strategies may include hardware repair, such as replacement of the NICs.

Due to frequent service updates and changes in user behavior, monitoring metric patterns may also evolve, a phenomenon known as concept drift. This drift can gradually weaken the effectiveness of ISOLATE over time. To alleviate this issue, ISOLATE is retrained on a weekly basis with newly collected data. Figure 4.11 illustrates the performance issue identification accuracy of both the retrained and offline versions over 20 weeks. A noticeable downward trend in the accuracy of the offline version can be observed, at-

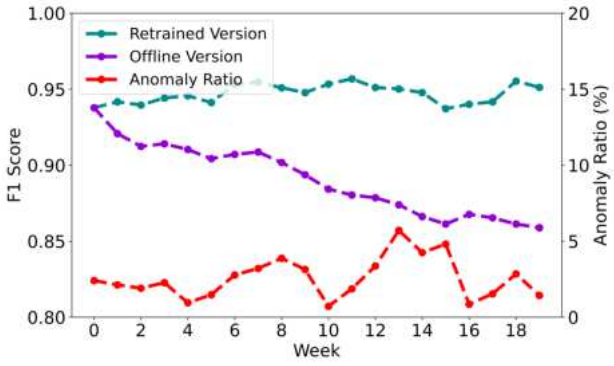


Figure 4.11: The Performance of deploying ISOLATE in Huawei Cloud System

tributed to pattern drift. Conversely, the version that is periodically retrained displays relatively stable performance. This adaptability to new patterns in online deployment scenarios underscores the robustness and effectiveness of ISOLATE within dynamically evolving cloud systems. More specifically, take one of the services $\mathcal{R}$ for example, the time proportion of undergoing performance issues during this period is 3.4%, and the online version of ISOLATE achieves a precision of 0.904, recall of 0.951, and F1 of 0.926. Another interesting observation is that though the anomaly ratio analyzed by engineers ranges from 0.7% to 5.6% during the 20-week period, the performance remains relatively stable in the retrained version of ISOLATE, demonstrating that the performance of ISOLATE is not sensitive to the anomaly ratio. It should be noted that the weekly retraining enhances the adaptiveness of our method, with an additional cost of human labeling of the potential performance anomaly data. However, it has been a common practice in cloud systems that on-call SREs manually verify some reported suspicious performance anomaly [24, 124, 204]. Specifically, according to the interview with SREs, they typically spend approximately one hour per week checking these data and they consider it to be affordable.

The ability to timely alert operations engineers after a performance issue happens is also of great significance. The delay time of ISOLATE in Huawei Cloud is the time delay between the detection

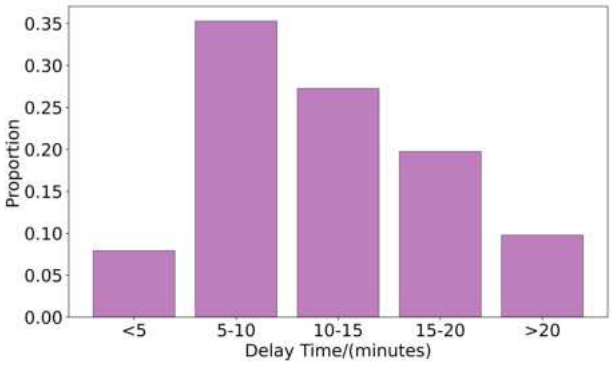


Figure 4.12: The Delay Time of ISOLATE in Identifying Performance Issues

of performance issues and the time at which the issues are confirmed by customer tickets [122]. The delay time of ISOLATE is shown in Figure 4.12, where we can observe that most of the performance issues can be captured 5 to 20 minutes after the event, and SREs can take mitigation strategies to avoid continuously compromising the user experience. Consequently, the potential negative impacts on the QoS can be significantly mitigated, enhancing system reliability and ensuring high availability.

4.5.2 Threats to Validity

Internal threats. The correctness of the implementation of baselines constitutes one of the internal threats to our study's validity. For the baselines, we utilized the open-sourced code released by the authors of the papers or packages on GitHub, like [249]. As for our proposed approach, the source code has been reviewed meticulously by the authors, as well as several experienced software engineers, to minimize the risk of errors and increase the overall confidence in our results. For parameter selection, we conducted extensive experiments with different parameters to find the most suitable configurations for both baselines and our proposed method. We chose the parameters based on the best results obtained in these experiments. To make our results reproducible, we have also made our code and

partial data available.

External threats. The external threats to the validity of our study mainly lie in the generalizability of our experimental results. We conduct experiments on the large-scale online systems of two regions within a prominent cloud service company. In addition to this, our approach is also evaluated on a publicly available dataset containing monitoring metrics from an Internet company, further expanding the scope of our evaluation. While the diversity of the experimental settings provides some confidence in the generality of our findings, it is essential to acknowledge that results might vary when applied to different cloud service providers, industries, or specific use cases. Nevertheless, we believe that our experimental results, obtained from these multiple sources, can demonstrate the generality and effectiveness of our proposed approach, ISOLATE.

4.5.3 Limitations of ISOLATE

While ISOLATE achieves satisfactory performance in detecting performance issues, its practical application in real-world environments reveals several limitations and areas for future enhancement. This section discusses these challenges and the practical strategies employed to mitigate them.

Firstly, ISOLATE, in its current form, is not inherently adaptive to the continuous evolution of metric patterns and correlations in dynamic cloud systems, which are subject to rapid software and application upgrades. This phenomenon, known as concept drift, can render the trained model obsolete, leading to false positives when newly established normal behaviors are incorrectly flagged as anomalous. In practice, this limitation is effectively mitigated through a strategy of periodic model retraining. By retraining the model on a regular schedule, for instance, on a weekly basis, the framework can continuously learn the new baseline behaviors of the cloud system. This cyclical updating process ensures the model remains synchronized

with the system's current state, thereby maintaining its detection accuracy over time.

Secondly, the computational complexity associated with graph neural networks can become a concern when scaling to enterprise-level cloud systems that generate a vast number of monitoring metrics. This potential for high computational cost is addressed by ISOLATE's targeted scope of deployment. Rather than attempting to model an entire system in a single monolithic graph, our approach is deployed at the node or service level, where the number of interdependent metrics is typically manageable (e.g., fewer than 100). This deliberate design choice not only contains computational demands and enhances efficiency but also aligns with the operational needs of Site Reliability Engineers (SREs). Performance issues detected at this more granular level often serve as valuable early warnings for systemic problems, providing actionable insights for prompt intervention.

Thirdly, as a deep learning-based method, ISOLATE lacks the direct, step-by-step transparency of traditional models like decision trees, which can be a limitation where full model interpretability is required. However, the model is not entirely a "black box." A degree of post-hoc interpretability is provided by the learned graph attention mechanism. These attention weights explicitly model the strength of learned correlations between metrics. When ISOLATE flags a performance issue, engineers can inspect these weights to identify which specific metric relationships have deviated from their established norms. This ability to pinpoint correlation violations offers granular, diagnostic insight, making the model partially interpretable and aiding in the initial localization of a performance issue's root cause.

4.6 CONCLUSION

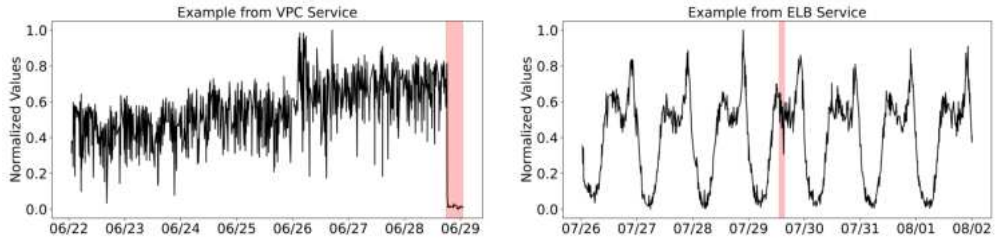
In this chapter, we propose ISOLATE, a novel framework to mine correlations among metrics, detect performance issues, and localize the correlation-violation metrics. Specifically, ISOLATE leverages a graph neural network with graph attention to capture the complex correlations between a variety of metrics and a label-conditional VAE model to distinguish normal and abnormal patterns. We also propose to utilize the positive unlabeled learning strategy to overcome the impacts of noisy data. Extensive experiments on one public dataset and two industrial datasets show that ISOLATE achieves 0.945 F1-Score on anomaly detection and 0.920 Hit@3 in terms of localizing correlation-violation metrics, outperforming all the baselines. Furthermore, our framework has been successfully incorporated into Huawei Cloud's performance issue monitoring and detection system. Both the codes and data are released to facilitate future research.

□ **End of chapter.**

Chapter 5

Adaptive AutoML-based Anomaly Detection

A common practice in the reliability engineering of cloud services involves the collection of monitoring metrics, followed by comprehensive analysis to identify performance issues. However, existing methods often fall short of detecting diverse and evolving anomalies across different services. Moreover, there exists a significant gap between the technical and business interpretation of anomalies, *i.e.*, a detected anomaly may not have an actual impact on system performance or user experience. To address these challenges, we propose ADAMAS, an adaptive AutoML-based anomaly detection framework aiming to achieve practical anomaly detection in production cloud systems. To improve the ability to detect cross-service anomalies, we design a novel unsupervised evaluation function to facilitate the automatic searching of the optimal model structure and parameters. ADAMAS also contains a lightweight human-in-the-loop design, which can efficiently incorporate expert knowledge to adapt to the evolving anomaly patterns and bridge the gap between predicted anomalies and actual business exceptions. Furthermore, through monitoring the rate of mispredicted anomalies, ADAMAS proactively reconfigures the optimal model, forming a continuous loop of system improvement. Extensive evaluation on one public and two industrial datasets shows that ADAMAS outperforms all



(a) Network Interrupt Anomaly in VPC Service (b) Packets Loss Anomaly in ELB Service

Figure 5.1: Examples of anomaly patterns of different services in cloud company $\mathcal{X}$

baseline models with a 0.891 F1-score. The ablation study also proves the effectiveness of the evaluation function design and the incorporation of expert knowledge.

5.1 Introduction

In recent years, many traditional software systems have been migrated to cloud computing platforms, which are managed effectively as *cloud services*. They serve hundreds of millions of users around the world on a 24/7 basis [49, 51, 125]. Due to their increasing scale and complexity, performance issues become inevitable [53, 54]. A common practice of reliability engineering involves the gathering of system monitoring metrics, also known as Key Performance Indicators (KPIs), from software and hardware components (*e.g.*, virtual machines and network devices). The collected data are then analyzed to identify any potential performance issues [25, 26, 251]. However, the overwhelming volume of monitoring metrics [24] in modern cloud services renders manual performance analysis infeasible [108]. As a result, many automated anomaly detection methods have been proposed [25, 103, 215, 242], aiming at revealing the unusual patterns of the metrics that reflect potential performance anomalies.

Modern service technologies, *e.g.*, microservices and serverless

functions, decouple software into sophisticated and fine-grained units [128]. Thus, cloud services tend to exhibit great diversity and dynamism not only in functionalities but also in anomaly patterns [247]. This situation poses two significant challenges for practical anomaly detection in production systems. First, it is difficult for a single method to effectively identify the entire spectrum of anomalies across different services, *i.e.*, the No Free Lunch Theorem [156, 167, 186]. For example, Fig. 5.1 presents two metrics with performance anomalies from a Virtual Private Cloud (VPC) service and an Elastic Load Balancing (ELB) service. A prevalent approach to anomaly detection involves first learning the normal pattern of a metric time series and then identifying anomalies when data points deviate from this established norm [127, 167]. While this strategy can work well in the first example, it may not be as effective for anomalies that violate time series periodicity, as illustrated in the second example, where the magnitude of deviation is not prominent. On the other hand, approaches [120, 164] that leverage the periodicity for anomaly detection face a different dilemma. The second challenge is related to the data-driven nature of anomaly detection. Existing approaches obtain the best model based on the experimental datasets. However, an anomaly detected on a technical level might not necessarily translate to a performance issue that impacts the overall business performance or customer experience. Consequently, engineers need to deal with a lot of false positives. Due to the frequent service updates and user behavior changes, the anomaly patterns of services may evolve accordingly, *i.e.*, concept drift [199], which further compounds the problem.

While efforts have been devoted to addressing these challenges, they suffer from important limitations that hinder their practical applications. For instance, [160, 230] employ the technique of Automated Machine Learning (AutoML) to prevent designing service-specific anomaly detection solutions [172, 195]. However, existing AutoML-based approaches often require a considerable amount

of labeled data or cannot properly evaluate the model performance, hindering the search for the optimal model architecture and parameters. In real-world systems, obtaining sufficient labeled data for each cloud service is often challenging [39]. On the other hand, the human-in-the-loop mechanism [19, 179] has been proven effective in incorporating domain knowledge to mitigate the gap between the predicted anomalies and the actual business exceptions. However, existing approaches heavily rely on human expertise to provide high-quality feedback, which impacts the scalability and efficiency. Simply using limited feedback data to retrain the models can hardly guarantee performance.

In this paper, we propose ADAMAS (Adaptive Domain-Aware Performance Anomaly detection for cloud Services Systems), an AutoML-based anomaly detection framework adaptive to different cloud services with diverse and evolving abnormal patterns. Specifically, ADAMAS consists of two stages, *i.e.*, a *Label-free Configuration Search stage* and a *Feedback-based Adaptive Learning stage*. In the first stage, ADAMAS employs Bayesian Optimization [80] to automatically search the best model architecture with parameters for anomaly detection. During the search, ADAMAS utilizes the proposed Noise-Free Mean squared error with Kurtosis (NFMK) evaluation function to estimate model performance without labels. In the second stage, ADAMAS adopts a lightweight, adaptive learning approach to efficiently incorporate expert knowledge. Specifically, during online serving, engineers can provide feedback on whether the detected anomaly is a false positive. Given the manually labeled data, ADAMAS employs Metric Stream Clustering (MSC) to group similar anomalous patterns into clusters and leverage historical feedback to identify true performance issues or false positives, significantly reducing the feedback required. To close the loop of continuous service updates, ADAMAS triggers model retraining when the cumulative mispredicted samples exceed a threshold. In this process, the domain knowledge introduced by human feedback

will guide the configuration search.

To evaluate the performance of ADAMAS, we conduct extensive experiments on one public and two industrial datasets. The results demonstrate that ADAMAS outperforms all baselines with an F1-score of 0.891. An ablation study demonstrates that compared to other evaluation functions, NFMK is more effective, and the MSC design can help ADAMAS achieve better performance with less feedback. A case study further shows how ADAMAS works in industrial cloud systems. To sum up, our main contributions are as follows:

- We design and implement ADAMAS, a domain-aware AutoML-based anomaly detection framework. It addresses two practical challenges in this field, namely, the diverse anomaly patterns across different services and the gap between the predicted anomalies and the actual performance issues. It eliminates the dependency on labels in the model searching process and significantly reduces the amount of human feedback for expert knowledge integration.
- We conduct extensive experiments with public data as well as industrial data collected from Company $\mathcal{X}$. Furthermore, our framework has been successfully incorporated into $\mathcal{X}$'s performance monitoring and anomaly detection system.

5.2 Background

In this section, we first introduce some background about metric-based performance anomaly detection in modern cloud systems. An example from an industrial scenario that motivates this work is also present. Then, we give a brief introduction to the general workflow of AutoML.

5.2.1 Performance Anomaly Detection in Cloud Service Systems

In modern cloud service systems, software reliability engineers (SREs) usually collect a large number of metrics that track the health status of the services (*e.g.*, CPU utilization, memory usage and network traffic) with monitoring tools like Grafana, Splunk, *etc.* [2]. Monitoring metrics provide the most granular information, which can be used to derive other types of cloud monitoring data. For example, alerts are usually triggered when metrics cross a threshold defined by the engineers [246]. In cloud systems, there are usually redundant components that provide fault tolerance and self-healing [25] capabilities. Thus, most of the service outages manifest themselves as performance anomalies (also known as fail-slow failures [131, 155]) before fail-stop failures, which can be detected through analysis of monitoring metrics. Previous studies [25, 135] have demonstrated that performance anomalies of similar types tend to trigger similar symptoms on the monitoring metrics. Thus, a particular type of performance anomaly can be characterized as an anomaly pattern.

However, we observe that not all anomaly patterns manifest in monitoring metrics are true performance anomalies. An industrial example from Cloud Company $\mathcal{X}$ is shown in Fig. 5.2, where the traffic of a load balancer in the Relational Database Service (RDS) is monitored. In the red area, the metric suddenly dips to nearly zero. It persists until system maintainers replace the network devices with new ones, which is confirmed as a network interrupt failure. In the green area, as load balancers can handle unpredictable traffic surges [57] by scaling up the hardware and software resources, the service can recover from a burst of requests due to its auto-scaling feature without manual intervention in the green area. Thus, this anomaly pattern is not a true anomaly because the fluctuating requests are expected behaviors. Due to the recurring nature of performance anomalies and failures [111], these false positives can bring

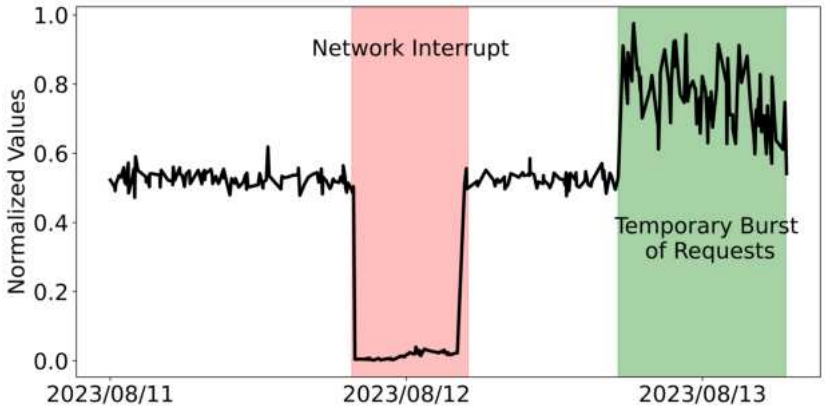


Figure 5.2: An Motivating Example from Company $\mathcal{X}$

meaningless trouble to engineers if not properly handled. A natural solution is to automatically differentiate the types of performance anomalies and determine the true anomaly by considering the context of the specific service.

5.2.2 Automated Machine Learning

In a large-scale cloud system, there are many services with different anomaly patterns that need different configurations to achieve optimal performance. Though there are many on-hand anomaly detection methods, in software reliability engineers' view, an "out of the box" tool is more desired. Automated Machine Learning (AutoML), the process that makes machine learning easier by avoiding tedious manual hyperparameter tuning for both machine learning experts and non-experts, provides great benefits to cloud system operators. Typically, AutoML can be formulated as a Combined Algorithm Selection and Hyper-parameter (CASH) problem [9].

A general AutoML workflow is shown in Fig. 5.3. The optimizer fetches model configurations based on a search space and the observations from the evaluator. Next, the evaluator measures the model trained with configuration passed from the optimizer on some objective functions, and the observation is used to update the opti-

mizer. The model configuration refers to both the model selection and model hyperparameters. After several iterations, the model that achieves the best performance will be output as the output model.

There are extensive studies on AutoML that are dedicated to improving the efficiency and the effectiveness of the optimizer [3, 4, 107, 250]. However, to the best of our knowledge, none of the existing AutoML methods possess the merit of domain knowledge that plays a critical role in differentiating true anomalies and false positives. Furthermore, existing AutoML frameworks lack the adaptability to new anomaly patterns in online services, which hinders their application in dynamic, evolving cloud systems.

5.3 Methodology

In this section, we present ADAMAS, which is a Bayesian Optimization-based approach for identifying performance issues based on monitoring metrics in cloud systems. First, we will give a formal definition of the problem. Then, we give the overview of ADAMAS. We illustrate the core design of ADAMAS in detail in the remaining sub-sections. We use the term metrics in two different contexts. To avoid confusion, we use monitoring metrics to refer to the collected time series that quantifies the health status of cloud systems, *e.g.*, CPU usage. While evaluation metrics refer to the indicators measuring the quality of anomaly detection models, *i.e.*, the F1 score.

5.3.1 Problem Formulation

The objective of our work is to detect performance anomalies in software systems, especially large-scale cloud systems with monitoring metrics. Specifically, a monitoring metric can be seen as a time series $X \in R^n = [x_1, x_2, ..., x_n]$, where n denotes the number of observations collected, *i.e.*, the length of a time series. Typically, a sliding window of historical values $X_t^l = [x_{t-l+1}, x_{t-l+2}, ..., x_t]$ is

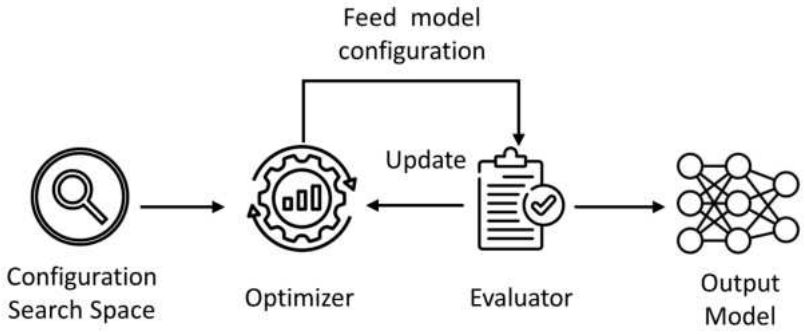


Figure 5.3: A General Overview of AutoML

used for modeling current observation, where l is the length of the sliding window. Our goal is to determine whether or not the given observation X_t^l is anomalous, *i.e.*, whether there is an occurrence of performance issues at the observation.

An AutoML framework typically contains several models with parameters. Given the set of monitoring metrics anomaly detection model $M = \{m_1, m_2, ..., m_k\}$, the set of parameter search space $\theta_i = \{\theta_1^i, \theta_1^i, ..., \theta_{n_i}^i\}$, we call the combination of a model m with parameter θ as a configuration $c \in \mathcal{C}$. With the objective function $\mathcal{L}(m, \theta, X)$ that evaluates the performance of configurations, the goal of AutoML is to find the optimal configuration c^* as follows:

$$c^* = \arg\max_{c \in \mathcal{C}} \mathcal{L}(c; X) \tag{5.1}$$

Normalization is performed on each monitoring metric to unify the range of them and improve the robustness of our method. We normalize the monitoring metrics with the Min-max normalization first.

5.3.2 Overview

The overall architecture of ADAMAS is shown in Fig. 5.4, which consists of two phases, namely, *label-free configuration search* and

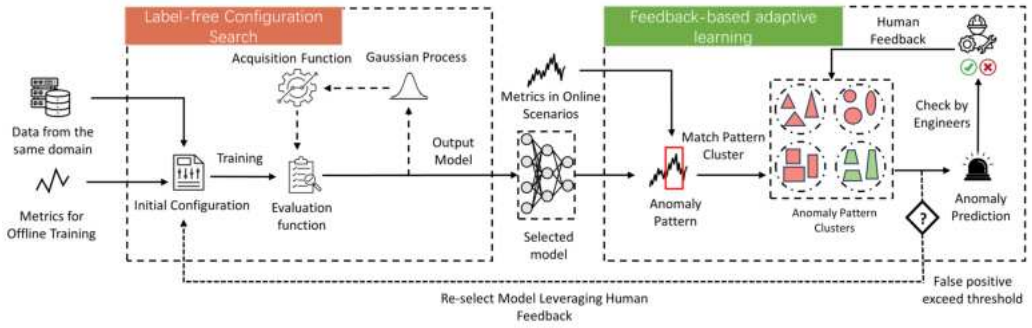


Figure 5.4: An Overview of Proposed Framework ADAMAS

feedback-based adaptive learning. In the configuration search phase, we apply Bayesian Optimization, a widely used AutoML technique, to iteratively search for new configurations and update the belief over the search space with the evaluation performance of the configuration. Bayesian Optimization typically consists of three main components: surrogate model, objective function, and acquisition function [210]. In our framework, the surrogate model that represents the estimated belief of performance over the observed configurations is the most widely used Gaussian Process (GP) [44]. The objective function is utilized to evaluate the performance of configurations. In cloud systems, performance anomaly detection based on monitoring metrics is usually unsupervised due to the lack of high-quality labels. Thus, an objective function that approximates the performance without labels is needed. To this end, we propose the NFMK objective function, which is an excellent approximation of the F1 score compared with other MSE-based objective functions. The acquisition function is used to search for the next configuration to observe with the exploration-exploitation trade-off. In our case, we deploy the commonly used max-value entropy Search, an acquisition function with a fast convergence rate. The initialized configuration can be either from the optimal configurations of the same service, which serves as a warm start of BO-based search, or random configurations.

In the Feedback-based Adaptive Learning phase, we leverage the output model from the search phase to detect anomalies in a scenario where monitoring metrics arrive in streams, *i.e.*, one instance at a time. To fill in the gap that many predicted anomalies are not true anomalies and reduce the number of false positives, we incorporate expert knowledge with human feedback from on-site engineers when an anomaly happens. However, the human effort will be unaffordable if each anomaly alert is checked manually; we try to solve this with a streaming metrics cluster method MSC. In this way, the historical feedback from a similar anomaly pattern can be leveraged, which prevents additional labeling by engineers. Moreover, when the false positive exceeds a threshold, a retraining and configuration search will be triggered to make our framework adaptive to the system change.

5.3.3 Label-free Configuration Search

Generally, an AutoML framework trains different models with different parameters to search for the best configuration. Before the search process, the framework randomly selects the initial models and hyperparameters as the cold start due to the lack of prior knowledge about the monitoring metrics. If there exist explored optimal configurations from metrics in the same service, these configurations can serve as a warm start [159] as anomaly patterns in the monitoring metrics of the same service are similar. After obtaining the configurations, the objective function will evaluate the performance of the current configuration, and the surrogate model (GP in our case) updates the belief over the search space. We design an objective function, Noise-Free Mean squared error with Kurtosis (NFMK), that serves as an excellent approximation to evaluation functions like the F1 score without labels. This is because, in the case of metrics anomaly detection, performance anomalies are rare, and the labels are hard to obtain, which prohibits the exact evalua-

tion of labels like the F1 score.

Objective Function

The idea of approximating an evaluation function with ground truth is based on a basic property of anomaly. If a monitoring metric segment cannot be reconstructed well by the model, it is more likely to be an anomaly [182]. Specifically, by using Mean Squared Error (MSE), we can evaluate the reconstruction ability of a model. However, collecting anomaly-free metric data is challenging and requires tedious manual efforts. So, the training data often contains noises, which manifest large errors in the MSE between the original and reconstructed monitoring metrics. To ensure the model can differentiate potential anomalies from normal patterns, we use the difference in kurtosis before and after noise removal. This helps measure the model's sensitivity to noise and its ability to discern outliers. Based on this observation, we try to capture the noise (*e.g.*, some mild performance anomaly ignored by engineers [53]) and calculate the MSE of the noise-free part. To make the noise part that potentially represents performance anomalies more distinguishable, we computed the difference in kurtosis before and after removing the noise. Specifically, as an effective estimator that measures the distance between the predicted and the raw monitoring metrics, MSE can be written as follows:

$$MSE(X, \hat{X}) = \sum_{i=1}^{n} \frac{\|X_i - \hat{X}_i\|^2}{n} \tag{5.2}$$

where X and $\hat{X}$ are raw and reconstructed monitoring metrics, respectively, and the length of the metric data is n, X_i and $\hat{X}_i$ are the raw and reconstructed sliding windows at the *i*-th point. Given an anomaly ratio r estimated by engineers according to historical observation, a threshold α is the r quantile of reconstruction error in each point. The points that exceed the threshold α are considered

noise, while others are normal. The noise indicator A_t can be written as follows:

$$A_t = \begin{cases} 1, & |X_t - \hat{X}_t| \geq \alpha, \\ 0, & |X_t - \hat{X}_t| < \alpha. \end{cases} \tag{5.3}$$

Typically, a good anomaly detection method can fit the pattern that represents the normal behavior of monitoring metrics. Suppose the noise-free part of monitoring metric X is represented as X_N ($X_N = \{X_t | A_t = 1\}$), then the MSE term without noise data is $MSE(X_N, \hat{X_N})$. When a model has a strong ability to differentiate anomalies from normal patterns, the objective function should change significantly after noise removal. This is based on the assumption that a higher kurtosis corresponds to greater fluctuation, indicating the presence of anomalies. Thus, if a model is proficient in anomaly detection and the original kurtosis is large, there will be a dramatic drop in kurtosis after the noise is removed. In contrast, if the data has a small kurtosis, this suggests fewer outliers. In such cases, both the kurtosis before and after noise removal would be small, and the MSE term dominates the objective function. This encourages the model to fit the monitoring metric well. The formulation of kurtosis is as follows:

$$Kurt(X) = \frac{\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^4}{(\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2)^2} \tag{5.4}$$

Note that we take the reciprocal of the kurtosis part to make it unified with the MSE part. This way, we derive NFMK as our objective function and minimize it to search for the optimal configuration. The NFMK function in our framework is:

$$\begin{aligned} NFMK(X, \hat{X}) =& MSE(X_N, \hat{X_N}) + (Kurt(X - \hat{X}) \\ & - Kurt(X_N - \hat{X_N}))^{-1} \end{aligned} \tag{5.5}$$

Acquisition Function

The acquisition function is the utility function that guides the selection of the next configuration that should be explored to reach the optimum of the objective function [202]. Among many acquisition functions, entropy search-based acquisition functions [70–72] motivated by information theory are proven to achieve a fast convergence rate that dramatically improves the efficiency of the adaptive learning process [12]. In our framework, we apply the Max-value Entropy Search (MES) [205] as the acquisition function. MES uses the information about the maximum value of objective function $y^* = \mathcal{L}(c^*, X)$ at optimal configuration c^*, where the gain in mutual information between the maximum y^* and the next configuration can be approximated by evaluating the entropy of the predictive distribution as follows:

$$\begin{aligned} MES(c) &= H(p(y|D_t, c)) - \mathbb{E}(H(p(y|D_t, c, y^*))) \\ &\approx \frac{1}{K} \sum_{y^* \in Y^*} \frac{\gamma_{y^*}(c)\phi(\gamma_{y^*}(c))}{2\Phi(\gamma_{y^*}(c))} - \log \Phi(\gamma_{y^*}(c)) \end{aligned} \tag{5.6}$$

$$\gamma_{y^*}(c) = \frac{y^* - \mu_t(c)}{\sigma_t(c)} \tag{5.7}$$

where D_t represents the observations until the *i*-th iteration, H denotes the entropy, ϕ is the probability density function and Φ is the cumulative density function of normal distribution. In the first term of Equation 5.6, the mean and variance of $p(y|D_t, c)$ are $\mu_t(c)$ and $\sigma_t(c)$. The expectation of the second term in Equation 5.6 is approximated using Monte Carlo methods through sampling a set of K function maxima Y^*, where the sampling is from an approximation via a Gumbel distribution.

5.3.4 Feedback-based Adaptive Learning

After the configuration search phase, an optimal model will be applied to detect performance anomalies in online scenarios where monitoring metrics arrive in streams. However, there are inevitably many false positives as some anomaly patterns on monitoring metrics are not considered performance anomalies. It is difficult for the offline model to discriminate these false positives from true performance anomalies without domain knowledge from system experts. To this end, we propose to incorporate the knowledge of experts through a human-in-the-loop to facilitate the anomaly detection process. It has been a common practice in cloud systems that on-call engineers manually verify every reported suspicious anomaly to mitigate the problem [24, 204]. Though the precision of the performance anomaly detection model can be tremendously improved with human feedback, the human effort devoted to labeling is non-negligible, especially in large-scale cloud systems that collect thousands of monitoring metrics on the fly. We observe that similar types of performance anomalies tend to trigger similar patterns on the monitoring metric, similar to [25, 135]. A straightforward idea is clustering similar anomaly patterns into one cluster and leveraging the feedback on historically similar patterns. In this way, the engineers only need to label the anomaly patterns in a cluster once.

Since the monitoring metrics are generated in a streaming manner, overwhelming data can exceed storage capacities, making clustering methods that require full data unsuitable [25]. Thus, we have no access to all metric patterns, unlike offline clustering approaches. We propose a streaming metrics clustering method, Metric Stream Clustering (MSC), that continuously clusters all incoming anomalous metric segments. Particularly, MSC is presented in Algorithm 1. First of all, noise in the NFMK term containing potential performance anomalies can be utilized to construct initialized clusters, which is denoted as X_0 for ease of presentation. We apply DB-

SCAN [41], a widely-used density-based clustering algorithm [169] to construct the initialized clusters. The mean vectors, cluster sizes, and radii are denoted as μ_0, S_0, R_0. Our core idea of metric stream clustering is that given a new metric segment X_t, we determine whether it can be attributed to an existing known anomaly cluster. The cluster will include X_t as an element and then update its parameters. Otherwise, a new anomaly cluster containing X_t itself will be created, an unseen metric pattern that on-site engineers should label. Particularly, we search for the index of the closest cluster idx_t and check whether the distance between the mean vector of the cluster and the metric pattern is smaller than the radius of the cluster. If yes, X_t will be considered a member of cluster idx_t. Otherwise, a new cluster containing X_t with a small radius δ represents the unseen pattern. When a new anomaly pattern is absorbed, the cluster's attributes should be updated. The mean vector and cluster size can be updated straightforwardly through the formulation in lines 11-12. However, the radius cannot be directly calculated as only the attributes of clusters are retained, which is common practice in online learning. We estimate it by analyzing the best and worst case of the radius update. On the one hand, the best case can be trivially derived in that the radius remained unchanged at $R_t[idx_t]$. On the other hand, the worst case is shown in Fig. 5.5, where the new radius reaches its maximum value when X_t lies on the opposite side of the cluster center μ_t with respect to the furthest point that yields the radius R_t, resulting in the largest radius update. We update the radius with the mean of the best and worst case, shown in line 13.

With the proposed clustering method, MSC, we can seamlessly integrate valuable feedback from on-site engineers to ADAMAS. Specifically, when the offline model identifies an unseen suspicious anomaly pattern, it issues a query to cloud experts for confirmation. It should be noted that the number of suspicious anomalies is far less than the number of all metric streams. Moreover, the expert is only required to label once for an anomalous cluster, where all the pat-

Algorithm 1: Metric Stream Clustering

Input: X_0, X_t, μ_t, S_t and R_t // Metrics and parameters of clusters
Output: μ_{t+1}, S_{t+1} and R_{t+1} // Updated parameters

if t=0 **then**
  // Initialization Phase
  $\mu_0, S_0, R_0 \leftarrow$DBSCAN$(X_0)$
else
  // Continuous Clustering Phase
  $\mu_{t+1}, S_{t+1}, R_{t+1} \leftarrow \mu_t, S_t, R_t$
  $D_t \leftarrow$ PairWiseDistance(X_t, μ_t)
  $idx_t \leftarrow$ MinIndex(D_t)
  if $D_t[idx_t] < R_t[idx_t]$ **then**
    // Add X_t to the nearest cluster and update parameters
    $\mu_{t+1}[idx_t] \leftarrow (\mu[idx_t] \times S_t[idx_t] + X_t)/(S_t[idx_t] + 1)$
    $S_{t+1}[idx_t] \leftarrow S_t[idx_t] + 1$
    $R_{t+1}[idx_t] \leftarrow |\mu_{t+1}[idx_t] - \mu_t[idx_t]|/2 + R_t[idx_t]$
  else
    // A new cluster is created
    $\mu_{t+1} \leftarrow$ Append μ_t with X_t
    $S_{t+1} \leftarrow$ Append S_t with 1
    $R_{t+1} \leftarrow$ Append R_t with δ // A small radius will be assigned to new cluster
  end if
end if
return μ_{t+1}, S_{t+1} and R_{t+1}

terns are considered anomalous if there is an element that is labeled as anomalous in the cluster. This further drastically reduces the required human effort. Consequently, the number of queries to experts is considerably lower compared to the total amount of metrics data, making our solution highly feasible and scalable. A problem with our strategy of incorporating human-in-the-loop is that though overwhelming false positives can be alleviated, the offline model remains unchanged and can not adapt to the evolving metric patterns. We try to solve this problem by setting an engineer-specified threshold that, when the false positive clusters exceed it, a model retraining will be triggered. Only when many accumulated false positives cannot be

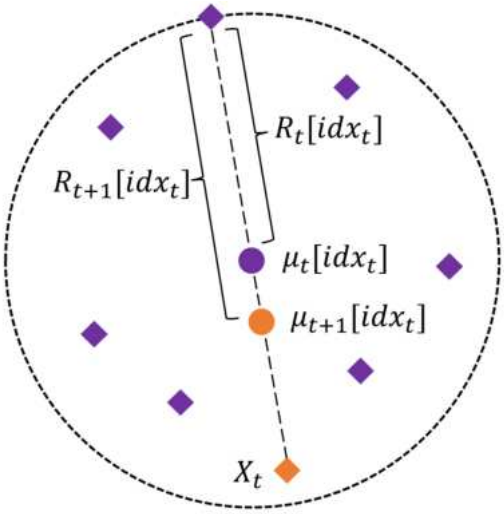


Figure 5.5: The Worst Case of Cluster Radius Update

properly differentiated by offline models should we make our model adaptive to these patterns through retraining; otherwise, the offline model works well toward a satisfactory performance. In this way, the computation cost that the regular retraining adopted by most frameworks consumes is significantly reduced. Specifically, we can take the incorporated domain knowledge to guide the configuration search process through the following objective function:

$$\begin{aligned} NFMK(X', \hat{X}') =& MSE(X'_N, \hat{X'_N}) + (Kurt(X' - \hat{X}') \\ & - Kurt(X'_N - \hat{X'_N}))^{-1} \end{aligned} \tag{5.8}$$

where X' and $\hat{X}'$ represent the raw and reconstructed online monitoring metric segments. $\hat{X'_N}$ represents the normal patterns not identified as anomalies and the false positives, which is a bit different from Equation 5.5. The new model is updated through training on the X' and selecting based on the objective function. It should be noted that X' represents the most recent metric segments, as past metric segments are discarded due to the overwhelming volume of metrics.

5.4 Experiments

In this section, we evaluate the effectiveness of ADAMAS using both a publicly available dataset and real-world monitoring metrics datasets collected from large-scale cloud systems in $\mathcal{X}$, a world-leading cloud company. In particular, we aim to answer the following research questions (RQs).

- RQ1: What is the effectiveness of ADAMAS compared with other baselines?
- RQ2: What is the effectiveness of each component in ADAMAS?
- RQ3: What is the sensitivity of ADAMAS to each hyperparameter?

5.4.1 Experiment Settings

Datasets

To evaluate the effectiveness of ADAMAS, we conduct experiments on a public dataset and two industrial datasets collected from large-scale cloud systems in Company $\mathcal{X}$, which confirm its practical significance. The statistics of the datasets are summarized in Table 5.1.

Public dataset. The AIOps18 dataset was released by an international AIOps competition in 2018, composed of multiple monitoring metrics collected from the web services of large-scale IT companies. Particularly, the dataset contains service metrics and machine metrics. The service metrics record the performance of the services, *e.g.*, response time and traffic, while the machine metrics reflect the health state of physical machines, including CPU usage and network throughput.

Industrial dataset. To fully evaluate the effectiveness in real-world production scenarios, we collect monitoring metrics from various services (*e.g.*, VPC, ELB, and RDS) in two availability zones

Table 5.1: Statistics of All Datasets

Dataset	#Metrics	#Points	#Ratio
AIOps18	29	5922913	2.26%
Industry A	506	5100480	4.67%
Industry B	535	5392800	3.46%

(AZs) of a global cloud service provider $\mathcal{X}$. These metrics, including the service CPU usage, NIC throughput, received packets of load balancers, *etc.*, closely monitor the health status of services. For each of these metrics, we collect one week of data with a sampling interval of one minute. We rely on the corresponding issue reports of performance anomalies, which contain the start and end times of problems identified by on-site engineers or customers of an online service, to label the data.

Implementation

We implement four widely used metric performance anomaly detection methods by cloud system operators in our ADAMAS framework. Table 5.2 shows the details of the algorithms and hyperparameters. We run all the experiments on a Linux server with Intel Xeon Gold 6140 CPU @ 2.30GHZ and Tesla V100 PCIe GPU. The proposed model is implemented under the PyTorch and BoTorch [10] framework and runs on the GPU. The value of γ corresponds to the NFMK function and is set to 0.98, and the value of δ in MSC is 0.01. Due to the policy of company $\mathcal{X}$, we will release our data upon acceptance. Both the artifacts and data are available on https://github.com/WenweiGu/ADAMAS.

Evaluation Metrics

The anomaly detection problem is modeled as a binary classification problem, so the widely used binary classification measurements can

be applied to evaluate the models. We employ Precision: $PC = \frac{TP}{TP+FP}$, Recall: $RC = \frac{TP}{TP+FN}$, F1 score: $F1 = 2 \cdot \frac{PC \cdot RC}{PC+RC}$. Specifically, TP is the number of abnormal samples the model correctly discovered; FP is the number of normal samples incorrectly classified as anomalies; FN is the number of anomalous samples that failed to be detected by the model. F1 score is the harmonic mean of the precision and recall, which symmetrically represents both precision and recall in a metric.

In real-world applications, anomalies will last for a while, leading to consecutive anomalies in the monitoring metrics. Therefore, it is acceptable for the model to trigger an alert for any point in a contiguous anomaly segment if the delay is within the acceptable range. Thus, we adopt the widely used evaluation strategy following [7, 112, 182, 215, 230]. In practice, different monitoring metrics have varying sampling frequencies, ranging from milliseconds (*e.g.*, latency) to minutes (*e.g.*, average load). Hence, to handle metrics with different sampling intervals, we use the number of timestamps rather than a fixed time interval. Particularly, the range is set to 10 timestamps based on the experiences of engineers.

5.4.2 RQ1: The Effectiveness of ADAMAS

To study the effectiveness of ADAMAS, we compare its performance with various state-of-the-art baselines on a public dataset and two industrial datasets collected from industry. These baselines include (1) Traditional machine learning-based approaches, isolation forest (IForest) [121] and ADSketch [25]; (2) Deep learning-based approaches, DAGMM [255], VAE [215], LSTM [76, 242], SCRNN [164] and Maat [103]; (3) AutoML-based approaches, Hyperband [107], AutoAD [160] and AutoKAD [230]. For a fair comparison, the models and the hyperparameter space of the Hyperband, AutoAD, and AutoKAD are the same as ADAMAS.

The results are shown in Table 5.3, where the best F1 scores

Table 5.2: Implemented Algorithms and their hyperparameters

Algorithm	Parameters	Range
Autoencoder	encoding dimension	10∼50
	hidden dimension	100∼200
	decoding dimension	100∼200
VAE	hidden dimension	20∼200
	latent dimension	10∼50
LSTM	hidden dimension	50∼200
Transformer	hidden dimension	20∼500
	head number	1∼8
	layer number	1∼6
Common	window length	100∼300
	batch size	128∼2048
	learning rate	$1^{-5} \sim 1^{-2}$

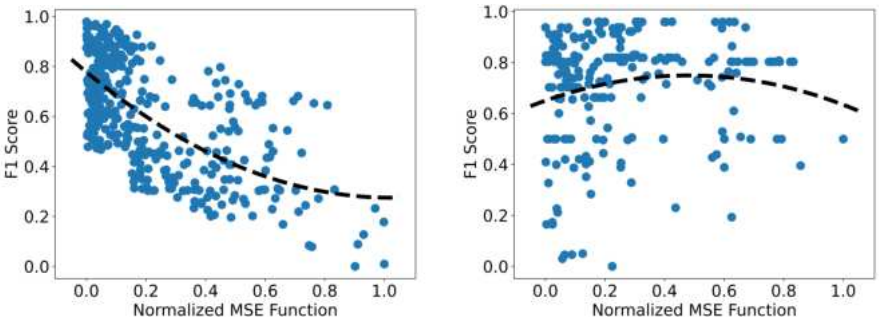
are marked with boldface. We can see the average F1 score of ADAMAS outperforms all baseline methods in three datasets. We observe that the improvement of effectiveness achieved by ADAMAS is more significant in two industrial datasets, especially on precision. This is because modern large-scale cloud systems widely employ microservices architectures [227] and possess the ability of fault tolerance and self-healing [45], and some anomalies manifested in monitoring metrics can be mitigated without intervention, which is not considered as performance anomalies. Based on a random sampling of 100 metrics from the datasets, software reliability engineers manually analyzed these samples and determined that 28.7% of the anomalies were data anomalies but not business anomalies. These data anomalies are non-negligible due to their potential to generate a significant number of false positives, which may distract the SREs. Through seamlessly incorporating human feedback, ADAMAS drastically reduces the false positives that waste the human effort of engineers. We also notice that though ADAMAS achieves near-perfect precision in both industrial datasets, there ex-

Table 5.3: Experimental Results of Different Anomaly Detection Methods

Methods	AIOps18			Dataset A			Dataset B		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
IForest	0.581	0.467	0.517	0.483	0.787	0.541	0.445	0.802	0.515
DAGMM	0.530	0.608	0.535	0.521	0.777	0.535	0.376	0.771	0.440
VAE	0.592	0.483	0.543	0.504	0.810	0.566	0.418	0.779	0.465
LSTM	0.598	0.706	0.530	0.621	0.730	0.638	0.534	0.726	0.553
ADSketch	0.744	0.670	0.677	0.691	0.732	0.658	0.583	0.745	0.618
Maat	0.753	0.687	0.692	0.595	0.818	0.656	0.568	0.856	0.641
SRCNN	0.741	0.656	0.674	0.619	0.707	0.640	0.597	0.756	0.636
Hyperband	0.836	0.647	0.732	0.640	0.743	0.673	0.615	0.764	0.683
AutoAD	0.798	0.665	0.710	0.611	0.784	0.662	0.539	0.848	0.614
AutoKAD	0.861	0.694	0.769	0.675	0.798	0.685	0.662	0.846	0.692
ADAMAS	**0.975**	**0.763**	**0.848**	**0.997**	**0.832**	**0.897**	**0.998**	**0.878**	**0.929**
Effect size	4.472	2.237	3.091	12.034	0.883	8.316	13.170	1.395	9.292

ists a few false positives in the AIOps18 dataset. The reason is that we apply MSC in order to reduce the effort of human labeling, which assigns all the metric patterns in a cluster with the same label. The symptoms in the two industrial datasets are similar for the same types of performance anomalies. In contrast, the AIOps18 dataset may contain different types of performance issues that demonstrate the same patterns, resulting in imperfect precision. However, we believe that ADAMAS will not introduce much burden on the cloud operators as only 2.5% of the prediction results are false positives. Furthermore, we find that the recall on two industrial datasets is consistently higher compared with the AIOps18 dataset. We attribute this phenomenon to the good monitoring mechanism established in Company $\mathcal{X}$ that makes most of the performance anomalies perceptible through analysis of monitoring metrics. We also calculated Cohen's d to measure the effect size of the differences between our approach and baselines.

In terms of baselines, we observe that AutoML-based baselines typically achieve better results compared with others, consistent with recent studies [160, 230]. This indicates that in cloud systems, there is no golden algorithm that performs consistently best on all data from various services, and we should employ AutoML to achieve



(a) Correlation of NFMK (b) Correlation of MSE

Figure 5.6: The correlation between two evaluation functions and F1 score

Table 5.4: Experimental Results of the Ablation Study

Methods	AIOps18			Dataset A			Dataset B		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
ADAMAS-OFF	0.843	0.741	0.778	0.728	0.793	0.730	0.715	0.845	0.751
ADAMAS-Jacob	0.956	0.715	0.813	0.972	0.753	0.857	0.976	0.813	0.870
ADAMAS-Snip	0.941	0.696	0.807	0.963	0.722	0.835	0.967	0.769	0.851
ADAMAS-Synflow	0.923	0.704	0.785	0.952	0.736	0.829	0.956	0.794	0.862
ADAMAS-F05	0.876	0.757	0.791	0.897	0.824	0.841	0.881	0.859	0.858
ADAMAS-F15	0.904	0.760	0.815	0.921	0.828	0.858	0.921	0.854	0.877
ADAMAS-F30	0.935	0.752	0.823	0.946	0.827	0.865	0.954	0.857	0.883
ADAMAS	**0.975**	**0.763**	**0.848**	**0.997**	**0.832**	**0.897**	**0.998**	**0.878**	**0.929**

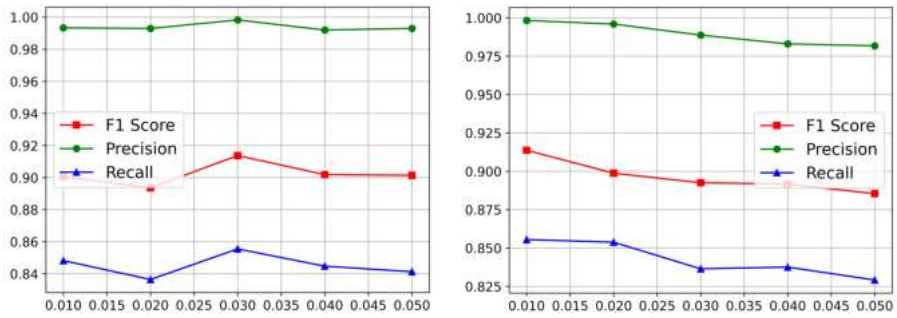
better performance rather than rely on a single algorithm. Among the machine learning-based and deep learning-based baselines, ADSketch and Maat perform best across three datasets with average F1 scores of 0.647 and 0.663. Specifically, ADSketch detects performance anomalies through pattern sketching, and Maat is based on the denoising diffusion model that resists noisy data during the training phase, both achieving relatively good performance. However, all these baselines purely mine the abnormal behaviors from the monitoring metrics without incorporating the crucial domain knowledge from on-site engineers, resulting in suboptimal results.

5.4.3 RQ2: The Effectiveness of Each Components of ADAMAS

In this research question, we conduct an ablation study on ADAMAS to show the effectiveness of its components. In particular, we derive seven baseline models based on removing the feedback from MSC, replacing the NFMK function with other widely used proxy functions from the AutoML community, and replacing it with random feedback. We further visualize the correlation between NFMK and F1 score to demonstrate the contribution of our design.

- **ADAMAS-OFF** This baseline removes the human feedback provided by on-site engineers and the retraining of ADAMAS. We merely utilize the model output by AutoML in the configuration search phase to detect the anomalies.
- **ADAMAS-Jacob, Snip, Synflow** These three variants replace the NFMK function with three widely used proxy functions, namely, Jacob Covariant [143], Snip [106] and Synflow [187]. Specifically, the original loss function value required by Snip is MSE loss in our context.
- **ADAMAS-F05, F15, F30** These three baselines randomly sample 5%, 15%, and 30% of the whole testing metric. Feedback on these sampled metric segments is utilized.

Table 5.4 shows the experimental results of ADAMAS and its variants. On one hand, we can observe performance drops when replacing our NFMK function with other proxy functions. Among these three proxies, Jacob Covariant performs the best. The assumption behind Jacob Covariant is that a lower correlation indicates a better network, which works well when the input data show various anomaly patterns. However, it may fall short when the input batch contains similar anomaly features. Another proxy, Snip, requires the original loss function, which in our case is MSE loss. This proxy metric approximates the loss change when a certain parameter



(a) Parameter Sensitivity of γ (b) Parameter Sensitivity of δ

Figure 5.7: Parameter Sensitivity of ADAMAS

is removed and encourages searching for the model with the lowest loss. In our scenario, where training data may contain noise, models with low MSE can still be ineffective in differentiating anomaly patterns. As a result, we can observe a significant drop on the recall. Similarly, the Synflow function computes the loss as the product of all parameters without requiring the original loss. However, it does not inherently consider the specific pattern of input data. Generally speaking, these proxies are not specially designed for anomaly detection, and they neglect the unique characteristics of the task, thus leading to unsatisfactory performance.

On the other hand, human feedback through MSC helps to improve the effectiveness of ADAMAS as it performs the best compared with the variants without feedback or with random feedback. We observe that in the variant without human feedback, not only does the precision drop markedly, but also the recall decreases a bit. We attribute this to the design of retraining during the adaptive learning phase because the human-labeled evolving anomaly patterns are utilized to update our searched model, enhancing its ability to capture unseen anomalies. Compared with the results in Table 5.3, we find that even without human feedback, our method still outperforms all the baselines, which shows the effectiveness of the design of our configuration search phase. It should be noted that, according to our experiments, the feedback ratio required by ADAMAS for three

datasets is 5.9%, 6.7%, and 4.3%, respectively. Compared to baselines with random feedback, ADAMAS can achieve higher performance with less human feedback than utilizing even 30% feedback of the whole metric, which demonstrates the effectiveness of our design of the MSC algorithm, where anomaly patterns in the same cluster will only be labeled by engineers once.

Our proposed NFMK function tries to fit the noise-free metric patterns and prevent overfitting through a regularization term. To demonstrate this, we visualize the correlation of the NFMK function versus the F1 score compared with MSE, shown in Fig. 5.6. For the MSE function, we can observe a trend of decline in the correlation when the MSE is small enough. This indicates the model overfits the metrics, *i.e.*, it fits both the normal and anomalous pattern well, resulting in a drop in performance. In fact, an effective performance anomaly detection method generally reconstructs normal patterns well while differentiating anomalies with high reconstruction errors, while our NFMK function takes this into account. According to our assumption, the smaller the NFMK function is, the better the performance is. We can observe an obvious negative correlation between the NFMK function and the F1 score, which demonstrates the effectiveness of the NFMK function as an estimator of the F1 score.

5.4.4 RQ3: Parameter Sensitivity of ADAMAS

In ADAMAS, there are only two parameters to tune, namely, the threshold γ that determines the ratio of noise in the NFMK function and δ that represents the radius of a new cluster in MSC. We hereon evaluate the sensitivity of ADAMAS to these two parameters on two industrial datasets. We change the value of γ and δ while keeping all other parameters unchanged in our experiments to guarantee fairness. Specifically, we choose the value of γ in the range from 0.95 to 0.99, while the value of δ is selected, ranging from 0.01 to 0.05. Fig. 5.7 presents the experimental results, where we observe that

ADAMAS is relatively stable under different settings. Therefore, ADAMAS exhibits robustness to these two parameters, eliminating the need for meticulous parameter tuning. This aligns well with our objective of sparing non-ML expert engineers' effort in parameter tuning. It should be noted that there is a consistent decline in performance with the increase of the δ. This can be attributed to the fact that when the radius of a new pattern becomes excessively large, it results in numerous new anomalous patterns merging into a single cluster. However, these patterns may not represent the same type of performance anomaly, thus reducing the accuracy.

5.4.5 Case Study

Since October 2023, ADAMAS has been effectively incorporated into the cloud service systems of Company $\mathcal{X}$, a leading company that provides cloud service to millions of users worldwide. ADAMAS can be seamlessly integrated into the existing cloud monitoring data analytics pipeline, such as Apache Kafka [25], InfluxDB [194], Datadog [68]. We focus on the deployment of ADAMAS in the Object Storage Service (OBS), which is a scalable solution offering cloud storage through a RESTful web services interface. Fig. 5.8 reflects the feedback required by ADAMAS over a 25-week deployment period, according to the weekly troubleshooting reports produced by SREs. It can be observed that the required feedback experiences a significant decrease after the initial week, thanks to our design of MSC. Following a service update in the sixteenth week, new anomaly patterns begin to emerge, resulting in an increase in required feedback. However, within a week, the required feedback decreases to a relatively low level again, effectively reducing the workload for SREs. Furthermore, we notice two drops in the accumulated false positives. This occurs due to our design of retraining when the accumulated false positives exceed a predetermined threshold (0.6 in our normalized values), which makes ADAMAS

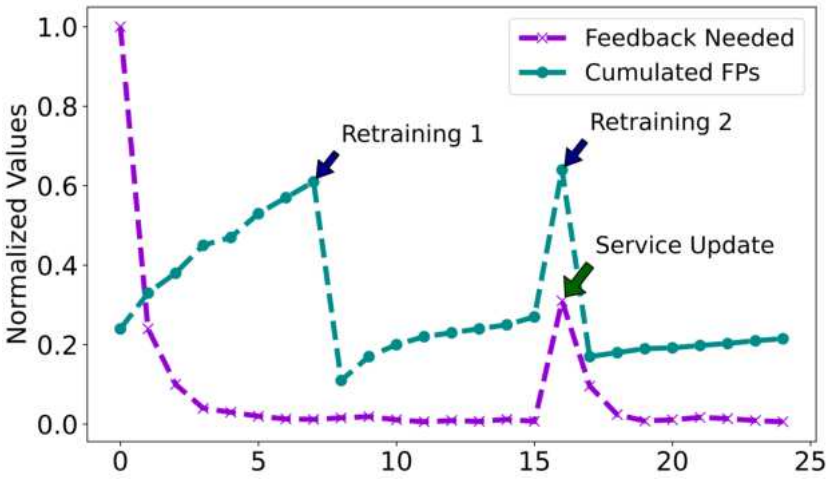


Figure 5.8: Industrial deployment in OBS service

adaptive to the evolving anomaly patterns of the cloud system.

Fig. 5.9 presents a case study illustrating how ADAMAS harnesses human feedback to minimize false positives. This example focuses on a metric monitoring the data transfer rate within the OBS provided by Company $\mathcal{X}$. A transient spike in the metric may be attributed to expected events like an influx in user requests or planned data migration. These events are not typically categorized as performance anomalies. On the contrary, a sudden dip to near zero in the data transfer rate could suggest network congestion or service malfunction, necessitating prompt troubleshooting. In this case, the green areas highlight four false positives; nonetheless, only the first anomaly pattern will undergo manual inspection by engineers, as these four false positives share a similar pattern and would be clustered together. When encountering a true performance anomaly, as denoted by the red area, an alert will be triggered. Engineers can then confirm that the service is experiencing performance degradation. When this anomaly pattern recurs again, engineers can recognize performance anomalies even without the effort of manually checking. Thus, we posit that our design of ADAMAS can reduce false positives and facilitate maintaining cloud systems.

5.5 Discussion

In this section, we discuss the reduction of human efforts with ADAMAS, its overhead, and some potential threats to the validity.

5.5.1 The Reduction of Human Efforts

We give an estimation of how much time ADAMAS can reduce the amount of human effort. The feedback ratio required by ADAMAS for the three datasets is 5.9%, 6.7%, and 4.3%. Based on interviews with software engineers who routinely maintain the systems, about 30% of data should be reviewed manually to maintain accurate anomaly detection without ADAMAS. We estimate that without ADAMAS, SREs should manually check approximately six times the cases. According to the interview with SREs, each SRE typically checks around 100 anomalies per week and spends about 2-3 minutes per anomaly, amounting to approximately 4 hours per week in total. ADAMAS can potentially reduce this effort to around 40 minutes. Thus, ADAMAS can reduce more than 3 hours of manual effort for each SRE per week. We aim to gather more data to accurately assess the exact time savings, which will be the focus of our future work.

5.5.2 The overhead of ADAMAS

The execution time for ADAMAS mainly consists of three components: the computation time for searching optimal configuration (the most time-consuming one), the time spent by humans providing feedback, and the time required for clustering. According to our investigation, when we search for 20 iterations and train each model with 50 epochs, the average execution times on three datasets for ADAMAS and Hyperband are 937s and 354s, respectively. It should be noted that the efficiency can be enhanced when utilizing the computational power available in cloud service systems. Fur-

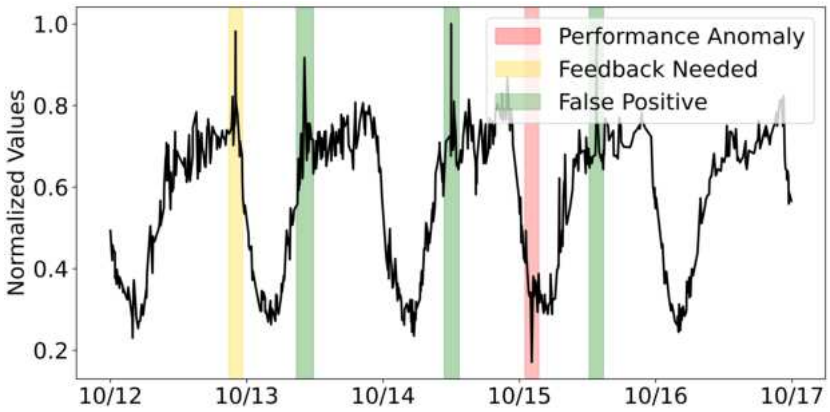


Figure 5.9: Case Study in OBS service

thermore, since this phase is conducted offline, its impact on real-time operations is insignificant. Thus, the overhead is affordable for industrial deployment.

5.5.3 Threats to Validity

Internal threats. Implementing baselines constitutes one of the internal threats to our study's validity. To address the threat of implementation, we directly utilized the open-sourced code released by the authors of the papers. As for our proposed approach, the source code undergoes rigorous peer code review by the authors and experienced engineers to minimize the risk of errors in our results. To mitigate the parameter setting threat, we fine-tuned the baseline methods utilizing a grid-search approach and derived the optimal results. To make our results reproducible, we have also made our code and partial data available.

External threats. The external threats to the validity of our study mainly lie in the generalizability of our experimental results. We conduct experiments on large-scale cloud systems within a prominent cloud service company. In addition to this, our approach is also evaluated on a publicly available dataset containing monitoring metrics from the web services of a large-scale IT company, further

expanding the scope of our evaluation. Therefore, we believe the evaluation is representative and convincing, demonstrating that our framework ADAMAS could be applied to typical cloud systems.

5.5.4 Limitations of ADAMAS

While ADAMAS demonstrates an advancement in adaptive anomaly detection for cloud services, its design presents opportunities for further research and enhancement. This section outlines two primary limitations of the current framework and proposes corresponding directions for future work.

A key limitation lies in the scope of the human-in-the-loop mechanism, which is primarily designed to address false positives. By allowing engineers to validate flagged anomalies, ADAMAS effectively reduces the noise from insignificant alerts. However, it currently lacks a proactive strategy for identifying false negatives—critical anomalies that the model fails to detect. These missed events can pose a greater risk to system reliability than false positives. A promising future direction is to enhance this feedback loop to also uncover potential false negatives. This could be implemented by periodically sampling data points that, while classified as normal, receive a high anomaly score from the model, placing them near the decision boundary. Presenting these ambiguous, high-risk "normal" instances to engineers for labeling would create a more comprehensive feedback system, allowing ADAMAS to learn from its under-sensitivity and improve its recall over time.

A second area for future enhancement concerns the computational efficiency of the AutoML-based search process. In its current form, ADAMAS evaluates each potential model configuration by conducting a full training cycle, a process that can be resource-intensive and time-consuming. This overhead may slow down the framework's ability to adapt in highly dynamic environments. To address this, future work could focus on developing a more efficient

performance estimation strategy. One approach is to design a meta-learning model that can predict the final performance of a given hyperparameter configuration without requiring full model training, perhaps by extrapolating from its initial learning curve. An alternative, complementary approach would be to perform the model search on strategically sampled subsets of the data, thereby reducing the cost of each evaluation. Success in this area would significantly lower the computational barrier to adoption, making ADAMAS more agile and cost-effective for continuous, real-time reconfiguration in production cloud systems.

5.6 Conclusion

In this chapter, we propose ADAMAS, an adaptive AutoML-based performance anomaly detection framework incorporated with domain knowledge. Specifically, ADAMAS consists of a configuration search stage and a feedback-based Adaptive Learning stage. In the first stage, a novel unsupervised evaluation function, NFMK, is proposed to guide the configuration search. In the second stage, we incorporate human feedback to differentiate true anomalies from all predicted anomalies. A streaming metrics clustering algorithm, MSC, is proposed to leverage the historical feedback from on-site engineers to decrease the human effort. Furthermore, when the number of mispredicted anomalies exceeds a threshold, retraining will be triggered to make our framework adaptive to evolving patterns due to rapid software updates. Extensive experiments on a public dataset and two industrial datasets show that ADAMAS achieves 0.891 F1-Score on anomaly detection, outperforming all the baselines. Furthermore, a case study demonstrates how ADAMAS is deployed into the cloud service system of Company $\mathcal{X}$.

□ **End of chapter.**

Chapter 6

Efficient KPI Root Cause Localization

To ensure the reliability of cloud systems, their runtime status reflecting the service quality is periodically monitored with monitoring metrics, *i.e.*, KPIs (key performance indicators). When performance issues happen, *root cause localization* pinpoints the specific KPIs that are responsible for the degradation of overall service quality, facilitating prompt problem diagnosis and resolution. To this end, existing methods generally locate root-cause KPIs by identifying the KPIs that exhibit a similar anomalous trend to the overall service performance. While straightforward, solely relying on the similarity calculation may be ineffective when dealing with cloud systems with complicated interdependent services. Recent deep learning-based methods offer improved performance by modeling these intricate dependencies. However, their high computational demand often hinders their ability to meet the efficiency requirements of industrial applications. Furthermore, their lack of interpretability further restricts their practicality. To overcome these limitations, we propose KPIRoot, an effective and efficient method for root cause localization integrating both advantages of similarity analysis and causality analysis, where similarity measures the trend alignment of KPI and causality measures the sequential order of variation of KPI. Furthermore, we leverage symbolic aggregate

approximation to produce a more compact representation for each KPI, enhancing the overall analysis efficiency of the approach. The experimental results show that KPIRoot outperforms seven state-of-the-art baselines by 7.9%∼28.3%, while time cost is reduced by 56.9%. Moreover, we share our experience of deploying KPIRoot in the production environment of a large-scale cloud provider Cloud $\mathcal{H}$[1].

6.1 Introduction

Large-scale cloud systems, such as Microsoft Azure, Amazon Web Services, and Google Cloud Platform, have revolutionized the computing infrastructure landscape, providing scalable, flexible, and cost-effective services to worldwide users on a 7×24 basis [117, 223]. However, the inherent complexity and scale of cloud service systems make performance issues (*e.g.*, slow application response time, service outages, and resource overload) an inevitability [100, 119], which may lead to potential violations of Service Level Agreements (SLAs), causing user dissatisfaction and financial losses [123]. Thus, promptly identifying and resolving performance issues has become a significant concern for both cloud vendors and users [126].

Cloud vendors usually collect real-time key performance indicators (KPIs) to monitor the health status of their services [183]. Anomaly detection is conducted over these KPIs to identify performance issues based on this KPI data [74, 245, 248]. For example, if the resource utilization rate is continuously high, it may indicate an imminent service overload and performance degradation. However, due to the scale of cloud systems, it is infeasible to analyze the KPI of each instance (*e.g.*, VM and container) individually. Since a cloud service typically consists of many instances, a common way is to monitor specific KPIs that can reflect the overall performance of the service, *e.g.*, latency, error count, and traffic, which we refer to

[1]Due to the company policy, we anonymize the name as Cloud $\mathcal{H}$.

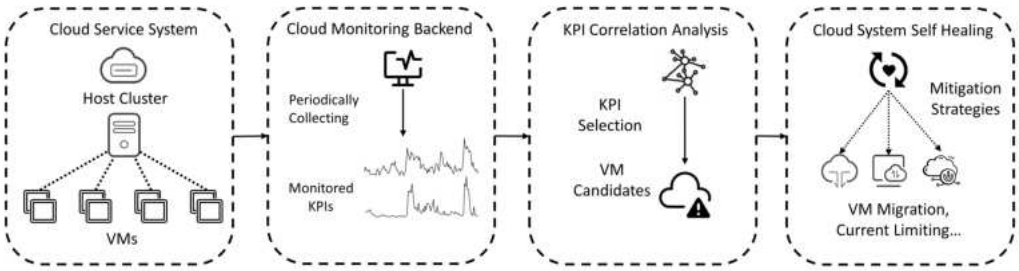


Figure 6.1: The Overall Pipeline of Root Cause Localization in Cloud $\mathcal{H}$

as *alarm KPIs*. Automated performance issue detection can thus be realized through configuring alerting rules or performing anomaly detection algorithms on such alarm KPIs. These underlying KPIs of individual instances or VMs within a cloud service may not be directly analyzed due to the scale of cloud systems. However, their collective behavior significantly influences the alarm KPIs.

When a performance issue is detected (*i.e.*, the alarm KPI is abnormal), it is crucial to identify the root cause (*e.g.*, which underlying instances cause the abnormal performance of the service). However, pinpointing the root cause is a non-trivial task since the monitored alarm KPI is highly aggregated and often derived [218], *i.e.*, the correlation between the underlying KPIs and the alarm KPI is complicated and hard to understand. Even experienced software reliability engineers (SREs) can struggle to pinpoint the specific KPIs that contribute to the root cause. Such a manual approach is like finding a needle in a haystack, which is tedious and time-consuming. Hence, the automated root cause localization method is an urgent requirement for prompt performance issue resolution.

In particular, a practical root cause localization approach for KPIs from cloud systems should meet the *efficiency* and *interpretability* requirements [226]. Specifically, due to the huge volume of underlying KPIs and the tight time-to-resolve pressure, the approach needs to be able to process large amounts of data (*e.g.*, thousands of KPIs) efficiently (*e.g.*, in seconds). Furthermore, the approach should pro-

duce interpretable results to help engineers take effective remedy actions, which is essential in the maintenance of cloud systems. Existing root cause localization methods typically adopt statistics or deep learning models. Statistic-based methods adopt Kendall, Spearman, and Pearson correlation to compute the linear relationships between KPIs and find the root cause [222]. However, these methods have high computational costs to calculate the correlation for every KPI pair and also suffer from low accuracy in handling complicated KPIs from cloud systems [221]. Some recent studies [218] adopt deep learning models (*e.g.*, graph neural networks) to model the KPI relationships for root cause localization. However, such methods suffer from high computation costs and lack interpretability [233, 235].

To address the above limitations, we propose KPIRoot, an effective and efficient root cause localization approach to identify the root cause underlying KPIs when an anomaly in the monitored alarm KPI is detected in cloud systems. To meet the efficiency requirement, KPIRoot first adopts the Symbolic Aggregate Approximation (SAX) representation to downsample the time-series data of KPIs and facilitate extracting the anomaly segments. Through filtering out the normal KPI data, KPIRoot can focus on the anomaly patterns instead of the whole time series, which greatly optimizes efficiency. Then, KPIRoot conducts both the similarity and causality analysis to localize the root cause KPIs. Specifically, the underlying KPIs with a high similarity of anomaly patterns to the alarm KPI are more likely to trigger the alert and be the root causes. On the other hand, causality analysis is used to validate the cause and effect in the temporal dimension, *i.e.*, the anomaly pattern of root cause KPIs should happen before that of the alarm KPI. Finally, KPIRoot combines the similarity and causality analysis results to produce a correlation score for each underlying KPI. The higher the score, the greater the possibility that the KPI is the root cause. The time complexity of KPIRoot is $\mathcal{O}(\sqrt{n})$ (n is the length of the KPIs), which allows it to process thousands of KPIs in seconds, thus facilitating

the resolution of real-time performance issues.

To evaluate the effectiveness of our proposed KPIRoot, we conducted extensive experiments based on large-scale real-world KPI data from a large cloud vendor. The experimental results demonstrate that KPI can pinpoint root cause KPIs more accurately compared with seven baselines with an F1-score of 0.850 and Hit Rate@10 of 0.917. On the other hand, KPIRoot largely reduces the computation cost with an execution time of around 5 seconds, which facilitates engineers in diagnosing root causes in real-time. In particular, we have successfully deployed our approach in the cloud service system of Cloud $\mathcal{H}$ since *Nov 2022* and successfully localized the true root cause of ten performance issues of emergency level with 100

We summarize the main contributions of this work as follows:

- We introduce KPIRoot, an effective and efficient method to localize the underlying KPIs that cause the anomaly. KPIRoot adopts the SAX representation for downsampling and combines both the similarity and causality of anomaly patterns of KPIs to identify the root cause. Such designs meet the practical requirements of efficiency and interpretability, making KPIRoot feasible to be deployed in large-scale cloud systems.
- Extensive experiments on three industrial datasets collected from Cloud $\mathcal{H}$'s large-scale cloud system demonstrate the effectiveness of KPIRoot, *i.e.*, 0.85 F1-score and 0.917 Hit@10 rate. The average execution time of KPIRoot is around 5 seconds, significantly outperforming seven state-of-the-art baselines.
- We have successfully deployed KPIRoot into the troubleshooting system of a large-scale cloud service system of Cloud $\mathcal{H}$ since *Nov 2022*. It has successfully analyzed ten emerging performance issues with 100% accuracy, and none of the issues affected the customer. The success stories of our deployment confirm the applicability and effectiveness of our method.

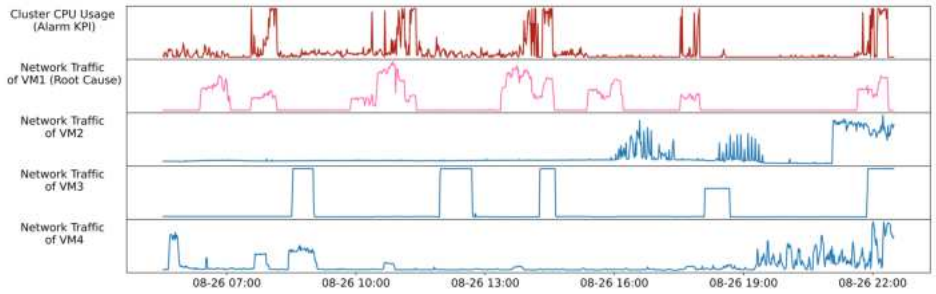


Figure 6.2: An Industrial Case in Cloud $\mathcal{H}$

6.2 Background and Motivation

In this section, we briefly introduce the KPI-based root cause analysis in cloud service systems and show the root cause localization practice in a large-scale cloud service system of Cloud $\mathcal{H}$ with a real case.

6.2.1 KPI-based Root Cause Localization in Cloud Systems

Ensuring optimal performance and reliability in cloud systems is of great importance. Performance anomalies like hardware malfunctions, network overloads, and security violations can significantly influence the performance of cloud systems and violate SLA [173]. Consequently, the need for run-time status and performance monitoring of cloud systems is in demand. Key Performance Indicators (KPIs) serve as informative tools that monitor the overall status of various components of cloud systems [27], providing helpful insights that aid in the identification of potential anomalies [180], and even proactively predicting these performance issues before they escalate into catastrophic failures [190]. Some common KPIs in cloud systems include CPU usage, memory usage, network bandwidth, latency, error rates, and service QPS (queries per second).

The cloud service system has become increasingly huge in scale and produces larger volumes of monitoring data. The highly inter-

connected nature of cloud systems incurs problems that performance failures can spread from one component to other components. Consequently, the failure diagnosis, root cause localization, and performance debugging in large cloud systems are more complex than before [163, 197]. In real-world applications, monitoring a large number of KPIs is computationally intensive, thus a more practical way is monitoring the aggregated KPI and configuring alerts.

Specifically, in large-scale cloud service clusters, large amounts of virtual machines (VMs) operate concurrently to provide tenants with various services. A special KPI is the "*alarm KPI*" that triggers alerts when a performance issue like an overload of CPU usage in the entire cluster happens. In large-scale cloud systems, service may consist of large amounts of VMs working together to respond to cloud users' demands [209]. Given the scale of these systems, individual monitoring of each VM becomes infeasible. Instead, software reliability engineers often utilize alarm KPIs as a more effective approach to oversee the overall performance of the service. When the alarm KPI indicates abnormal activity, it becomes crucial to identify which VMs are the root causes. The root cause refers to the specific VMs that trigger the anomaly within the alarm KPI. For instance, if the alarm KPI is triggered due to a fairly high CPU usage, the root cause could be the particular VMs that directly cause the resource overload. Such a setup allows for the proactive identification of performance issues. In addition to the alarm KPI, other KPIs monitor the bytes per second (bps) and packets per second (pps) of each VM in the cluster [101]. These KPIs offer valuable insights into the data traffic of each user, serving as indicators of their workload.

The overall pipeline of root cause localization using monitoring KPI in Cloud $\mathcal{H}$ is shown in Fig.6.1. Cloud service providers typically have many data centers spread across different regions. Each region consists of multiple, isolated locations known as availability zones to ensure low latency and high availability [93]. Users can create their VMs in any region that best fits their needs. Then, the

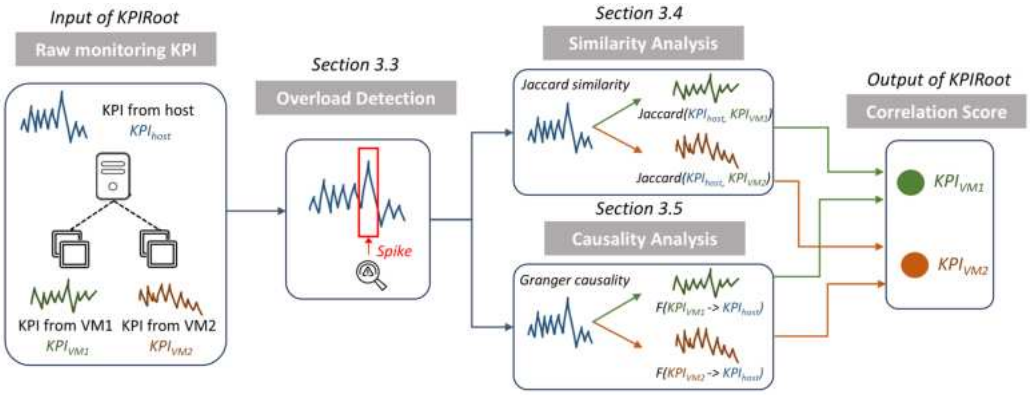


Figure 6.3: The Overview of Our Proposed Method KPIRoot

behavior of both the host CPU cluster and the VMs is continuously monitored and recorded through KPIs, including CPU usage, memory usage, and netflow throughput. Next, KPI correlation analysis is conducted to understand the dependencies between each VM and the host cluster. Based on the KPI correlation analysis, mitigation strategies such as VM migration or throttling are enacted to alleviate the overload in the system. In our paper, we focus on the third and most significant part, namely root cause analysis, and propose KPIRoot.

6.2.2 A Motivating Example

In a cloud system, there exist intrinsic correlations between the KPIs of individual VMs and the alarm KPI [52], which is a crucial part of RCA. Take the CPU usage in cloud systems as an example, the correlation is based on the fundamental principle of resource allocation within a cloud system that each VM is allocated a portion of the cluster's resources like CPU [31]. When a VM's workload increases, it consumes more CPU resources, thereby affecting the overall CPU usage. However, the relationship between the KPIs of individual VMs and the overall CPU usage of the cluster is complex and non-linear [196]. This complexity is due to the sophis-

ticated architecture of modern cloud systems and the principles of resource allocation they employ. In other words, these mechanisms ensure that the resource usage of one VM does not significantly impact others, thereby preventing a single VM from monopolizing the CPU [252]. Thus, the bulge of the workload KPI of a single VM does not necessarily lead to alarm KPI trigger alerts.

To effectively identify the root cause of performance anomaly, we capture the correlations between the VM KPIs and the alarm KPI that depicts the contribution of VMs to the detected performance anomaly. This correlation often manifests in a similar waveform between the VM's KPIs and the alarm KPI. For example, a sudden surge in a VM's data traffic would likely lead to an increased demand for CPU resources, which would be reflected as a spike in the KPI of the cluster's CPU usage [13]. The KPI correlation analysis approach aiming to mine the inherent correlations in KPI data can be leveraged to pinpoint the root causes of system alerts. In our case, similarity and causality analysis are adopted. Firstly, similarity analysis allows us to identify which VMs are behaving similarly to the overall system's performance, as reflected by the alarm KPI. Therefore, similarity analysis can help narrow down the potential root causes of the anomaly. Secondly, causality analysis is critical as it allows us to determine which changes in VM KPIs occurred before the anomaly, thus providing clues as to which VMs might have triggered the anomaly.

An industrial case in a real-world cloud system cluster of Cloud $\mathcal{H}$ is shown in Fig.6.2. There is an alarm KPI monitoring the overall CPU usage of the cluster, and several VM KPIs monitoring the network traffic of individual VMs. For the purpose of discussion, we focus on four of the VM KPIs. We can observe that the waveforms of VM2 and VM4 have weak alignments with the fluctuations in the alarm KPI, indicating a lower correlation, and thus are unlikely to be significant contributors to the CPU overload. The KPI of VM1 and VM3 exhibit a high degree of similarity to the alarm KPI, indicating

they are potential root causes for the anomaly. However, to ascertain the true root cause of the CPU overload, time series causality, *i.e.*, chronological order of events should also be taken into consideration. As confirmed by the SREs, it is VM1, not VM3, which is the true root cause of the CPU overload. This is because the spike in VM1's KPI precedes the CPU overload anomaly, while the spike in VM3's KPI happens slightly after the anomaly, indicating that it is an outcome, not a cause of the anomaly. Indeed, in a cloud system, a VM's increase in resource consumption usually precedes the CPU overload due to temporal causality, which is why we take temporal causality into consideration in our method.

6.3 METHODOLOGY

In this section, we present KPIRoot, an automated approach for root cause localization with monitoring KPIs in cloud systems. We first formulate the problem we target. Then we provide an overview of the proposed method. Next, we elaborate on each part of our method, *i.e.*, anomaly segment detection, similarity analysis, and causality analysis. We finally analyze the complexity of our proposed algorithm.

6.3.1 Problem Formulation

The goal of our work is to identify the root causes of performance anomalies like CPU overload in large-scale cloud systems, based on the alarm KPI and observed individual KPIs. The root causes are the VMs that influence the system service quality. By throttling the throughput of these VMs, we can alleviate the system-level anomaly and restore service quality. Given the alarm KPI that monitors the status of the host cluster $X_{host} \in R^n$ and the monitored KPIs of VMs, *e.g.*, the netflow of them $X_i \in R^n, i \in \{1, 2, ..., m\}$, where N denotes the number of observations collected at an equal interval and

m is the number of monitored VMs. To determine the true root cause of the detected anomaly, a correlation score $c_i \in [0, 1]$ that represents the contribution of a VM KPI to the anomaly is calculated. Then the root causes can be obtained by ranking the correlation score and KPIs with the top K scores are deemed as root causes.

6.3.2 Overview

The overview of KPIRoot is shown in Fig.6.3, which consists of three key components, namely, anomaly segment detection, similarity analysis, and causality analysis. Given the raw monitoring KPI, to make the RCA more efficient and meet the real-time requirement of industrial deployment, we propose to adopt SAX representation to downsample the raw KPI. Then KPIRoot detects the potential anomaly segments in the downsampled alarm KPI of the host cluster (Section.6.3.3). In this step, a score that describes the variation trend of KPI will be computed, an anomaly segment will automatically extracted around the spike. Then KPIRoot conducts a similarity analysis to compute the similarity between VM KPIs and the alarm KPI during the anomaly period (Section.6.3.4). This analysis provides insights into how each VM influences the host cluster by measuring the alignment of the KPI trends. A causality analysis is then conducted (Section.6.3.5) to identify the cause-and-effect between the VM KPIs and the alarm KPI. In our case, we utilize Granger causality. The results from the similarity and causality analyses are then combined to compute a correlation score for each KPI.

6.3.3 Anomaly Segment Detection

To make KPIRoot efficient and meet the industrial requirement of real-time identification, we propose to adopt Symbolic Aggregate Approximation (SAX) [116]. SAX has several advantages in KPI analysis: First, SAX allows for a significant reduction in the dimension of the raw KPI, which can make subsequent similarity compu-

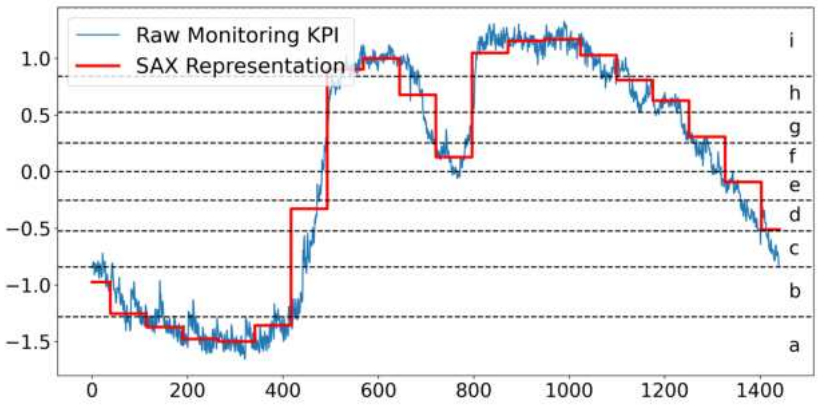


Figure 6.4: An Illustration of SAX Representation

tation more efficient [147]. Second, SAX can effectively filter out the noise and highlight the significant patterns in the KPIs by aggregating several consecutive data points into a single "symbol" [170]. Specifically, the raw KPI x of length n will be represented as a w-dimensional vector $P = \{p_1, p_2, ..., p_w\}$, where the j^{th} element can be calculated as follows:

$$p_i = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} x_j \tag{6.1}$$

In other words, to reduce the dimension of KPI from n to w, the KPI is divided into w equal-sized subsequences. The mean value of the subsequence is calculated and a vector of these values becomes the Piecewise Aggregate Approximation (PAA) representation [55]. Indeed, PAA representation is intuitive and simple, yet shows an approximate performance compared with more sophisticated dimension reduction representations like Fourier transforms and wavelet transforms [116]. Before converting it to the PAA, we normalize each KPI to have a mean of zero and a standard deviation of one.

In the industrial scenario, a fixed threshold method (e.g., CPU usage higher than 80%) is commonly used to detect system resource

usage anomalies. However, fixed thresholds can be limiting as they do not adapt to changes in the system's behavior over time. Typically, an anomaly refers to a state where the system's resources, such as CPU, memory, or network bandwidth, are being utilized at their maximum capacity and will cause performance issues for the system. However, in a dynamic cloud system, at which threshold an anomaly occurs can shift. Specifically, during periods of low demand, a sudden spike in resource usage might be considered an anomaly. However, during peak demand periods, the system might be designed to handle much higher resource usage, thus the same usage level would not be considered an anomaly. Furthermore, the individual preferences of engineers make the setting of universally acceptable static thresholds complex. What might be a suitable threshold for one engineer could be too high or too low for another, leading to potential issues being overlooked or an excessive number of false alarms [244].

Technically, anomaly segment detection can be formulated as a spike detection problem. By detecting an uprush in workload, the early warning of potential system anomaly can be identified and root cause localization will be enabled. A score that describes the variation trend of a KPI is computed as follows:

$$r_i = \frac{\sum_{k=i}^{i+l-1} p_k}{\sum_{j=i-l}^{i-1} p_j} \tag{6.2}$$

where l denotes the historical lags taken into consideration. If the value r_i is greater than a large threshold γ, it suggests that the usage of resources as indicated by the KPI starts to undergo a spike and we denote the start point of overload as t_s. Once the KPI value drops below the value of t_s, it signifies that the overload ends; the endpoint of the overload is denoted as t_e. In other words, $x_{t_e} < x_{t_s}$ and $x_{t_e-1} > x_{t_s}$.

Our approach allows for the detection of anomaly segments by

considering the variation trend of KPI, effectively marking the beginning and endpoint of anomaly segments. This can be a particularly beneficial preprocess step for subsequent correlation analysis.

6.3.4 Similarity Analysis

Motivated by [221], we propose to compute the similarity of the alarm KPI and VM KPIs to measure the degree of the root cause. The intuition behind this is that if a VM is responsible for triggering an overload, its KPI should exhibit a significant similarity with the host cluster's KPI, especially during periods of overload. If a VM is indeed the root cause of an overload, it is expected that its resource usage pattern would reflect the pattern of the host resource usage.

Although there exist some approaches that can be used to calculate the similarity of monitoring KPIs, such as AID [221], HALO [238], and CMMD [218], however, in real-time cloud computing systems, timely root cause localization is paramount. Traditional algorithms such as Dynamic Time Warping (DTW) might not be suitable for such scenarios due to their high time complexity, which can be prohibitive for processing large volumes of data in a real-time manner.

Thus we transform the KPIs into symbolic sequences and then compute the similarity between these sequences using the Jaccard similarity coefficient. To obtain the discrete representation with symbols, a discretization technique that will produce symbols with equal probability is desired. As proved by [116], the normalized KPIs have nearly Gaussian distributions. It's easy to pick equal-sized areas under the Gaussian distribution curve using lookup tables for the cut line coordinates, slicing the under-the-Gaussian-curve area. Suppose we have α symbols in the SAX representation, then the breakpoints refer to a sort of numbers $\beta = \{\beta_1, \beta_2, ..., \beta_\alpha\}$ such that the area under normalized Gaussian distribution curve between β_i to β_{i+1} is equal to $\frac{1}{\alpha}$. The PAA representation element in Section.6.3.3 between β_i to β_{i+1} will be assigned with the i^{th} symbol

shown as follows:

$$s_i = alphabet_l, \quad if\ \beta_l \leq p_i \leq \beta_{l+1} \tag{6.3}$$

where, $alphabet_i$ denotes the i^{th} symbol and s_i denotes the i^{th} element of the SAX representation S. An example of SAX representation of a monitoring KPI with $w = 20, \alpha = 9$ is shown in Fig.6.4.

We adopt the Jaccard similarity coefficient rather than other similarity measures because of its advantages when dealing with symbolic sequences like the SAX representation [67]. Moreover, Jaccard similarity is easy to compute and can effectively capture the similarity between two symbolic sequences regardless of their lengths. This makes it very suitable for our case, where the lengths of the symbolic sequences could vary. Then, the Jaccard similarity can be computed as follows:

$$Jaccard(S_{host}, S_i) = \frac{|S_{host} \cap S_i|}{|S_{host} \cup S_i|} \tag{6.4}$$

where S_{host} is the SAX representation of the host cluster's KPI and S_i is the SAX representation of individual VM KPI X_i.

6.3.5 Causality Analysis

The Symbolic Aggregate Approximation (SAX) method is effective in reducing the dimension of raw KPI, however, the computation of SAX representation-based similarity does not provide any insights into the causality between VM KPIs and alarm KPIs. As mentioned by [142], the ability of Granger causality analysis to analyze the correlation between KPIs can be a key factor for improving the accuracy of the root cause localization. By using Granger Causality in conjunction with SAX representation, we can not only analyze large

quantities of time series data effectively but also gain insights into the potential causality between different KPIs. That is why we take Granger Causality [178] as a supplement.

Granger Causality is a statistical hypothesis test used to determine if one KPI is useful in forecasting another KPI [6]. For instance, if a VM KPI undergoes an uprush and causes the alarm KPI to trigger alerts, *i.e.*, the change in the VM KPI precedes the changes in the alarm KPI, then Granger causality exists from the alarm KPI to the VM KPI. It should be noted that Granger Causality is unidirectional, which means that if VM KPI Granger causes alarm KPI, it does not imply that alarm KPI Granger causes VM KPI. In our case, we are interested in understanding how VM KPIs influence the alarm KPI of the host cluster, so we focus on the Granger causality from the VM KPIs to the alarm KPI. Specifically, assuming that the two KPIs can be well described by Gaussian autoregressive processes, the autoregression (AR) of alarm KPI without and with information from VM KPI can be written as follows:

$$p^t_{alarm} = \hat{a}_0 + \sum_{j=1}^{q} \hat{a}_j p^{t-j}_{alarm} + \hat{\varepsilon}_t \tag{6.5}$$

$$p^t_{alarm} = a_0 + \sum_{j=1}^{q} a_j p^{t-j}_{alarm} + \sum_{j=1}^{q} b_j p^{t-j}_{i} + \varepsilon_t \tag{6.6}$$

where the first equation uses the past values of the PAA representation of host KPI X^{host} while the second includes the past values of the PAA representation of both X^{host} and X^{vm}. Furthermore, $\hat{a_j}$ is the autoregression coefficients for X^{host}, while a_j and b_j are the autoregression coefficients for X^{host} with the contribution of both X^{host} and X^{vm}'s historical values. Both $\hat{\varepsilon}_t$ and ε_t are residual terms assumed to be Gaussian and q is model order which represents the

amount of past information that will be included in the prediction of the future sample. Then we conduct the F-statistic test:

$$F_{vm \to host} = \frac{\sum_{t=t_s+q}^{t_e} (\hat{\varepsilon}_t^2 - \varepsilon_t^2)/q}{\sum_{t=t_s+q}^{t_e} \varepsilon_t^2/(t_e - t_s - 2q - 1)} \tag{6.7}$$

where $\hat{\varepsilon}_t^2$ and ε_t^2 represent the mean square error (MSE) of the AR model of host KPI without and with information from VM KPI. t_s and t_e are the start point and end point of the detected overload. The F-statistic test follows an F-distribution with q and $t_e - t_s - 2p - 1$ degrees of freedom under the null hypothesis that the VM KPI does not Granger-cause the host KPI. The calculated F-statistic can be a good indicator of the VM KPI Granger-causality to the host KPI.

After both the similarity and causality analyses are performed, KPIRoot combines these two scores to create a more comprehensive correlation score for each VM KPI. Specifically, the correlation score is a weighted sum of similarity score and causality score:

$$c_i = \lambda \times Jaccard(S_{host}, S_i) + (1 - \lambda) \times F_{vm \to host} \tag{6.8}$$

where c_i is the correlation score between the i^{th} VM KPI and the alarm KPI. The balance weight λ is a hyperparameter. In our experiments, this parameter is set to be 0.9.

6.3.6 Complexity Analysis

The proposed method KPIRoot is summarized in Algorithm.2. The computation of our method mainly lies in the similarity and causality analysis. In industrial practice, $w \approx \sqrt{n}$, which means the lengths of SAX representation of KPIs are roughly $\sqrt{n}$. So, the time complexity of obtaining SAX representation is $\mathcal{O}(\sqrt{n})$. On one hand, the time complexity of Jaccard similarity is directly proportional to

Algorithm 2: KPI Root Cause Localization

Require: The alarm KPI of the host X_{alarm}; The KPIs of VMs $X_i, i \in \{1, 2, ..., m\}$;

Ensure: The correlation scores of VM KPIs that correlate to the anomaly of alarm KPI c_i

for $i = 1; i \leq w; i++$ **do**
  $p^i_{alarm} = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} x^j_{alarm}$
end for
// Anomaly Segment Detection
$t_s = \{t | r_t = \frac{\sum_{k=t}^{t+l-1} p^k_{alarm}}{\sum_{j=t-l}^{t-1} p^j_{alarm}} > \gamma\}$
$t_e = \{min(t) | p_{t_e} < p_{t_s} \text{ and } p_{t_e-1} > p_{t_s}\}$
$p_{alarm} = p_{alarm}[t_s : t_e]$
$s^i_{alarm} = \{alphabet_l,\ s.t.\ \beta_l \leq p^i_{alarm} \leq \beta_{l+1}\}$
for $i = 1; i \leq m; i++$ **do**
  // Similarity Analysis
  for $k = 1; k < m; k++$ **do**
    $p^k_i = \frac{w}{n} \sum_{j=\frac{n}{w}(k-1)+1}^{\frac{n}{w}k} x^k_i$
    $p_i = p_i[t_s : t_e]$
    $s^k_i = \{alphabet_l,\ s.t.\ \beta_l \leq p^k_i \leq \beta_{l+1}\}$
  end for
  $Jaccard(S_{host}, S_i) = \frac{|S_{host} \cap S_i|}{|S_{host} \cup S_i|}$
  // Causality Analysis
  for $t = t_s + q; t < t_e; t++$ **do**
    $p^t_{alarm} = \hat{a}_0 + \sum_{j=1}^{q} \hat{a}_j p^{t-j}_{alarm} + \hat{\varepsilon}_t$
    $p^t_{alarm} = a_0 + \sum_{j=1}^{q} a_j p^{t-j}_{alarm} + \sum_{j=1}^{q} b_j p^{t-j}_i + \varepsilon_t$
  end for
  $F_{vm \to host} = \frac{\sum_{t=t_s+q}^{t_e} (\hat{\varepsilon}^2_t - \varepsilon^2_t)/q}{\sum_{t=t_s+q}^{t_e} \varepsilon^2_t / (t_e - t_s - 2q - 1)}$
  $c_i = \lambda \times Jaccard(S_{host}, S_i) + (1 - \lambda) \times F_{vm \to host}$
end for
return c_i

the KPI length, so the complexity of similarity analysis is $\mathcal{O}(\sqrt{n})$. On the other hand, the complexity of Granger causality mainly depends on the autoregression of P_{host}, which is $\mathcal{O}(\sqrt{n} \times q^3)$, where q is the time lag of Granger causality (usually very small). Thus, the complexity of KPIRoot is $\mathcal{O}(\sqrt{n} \times (q^3 + 2))$. As a comparison, the time efficiency of methods like AID (based on DTW) is $\mathcal{O}(n^2)$, let alone more complex deep learning-based methods like CMMD. Therefore, KPIRoot is a more suitable method for industrial applications that demand real-time root cause localization.

6.4 EVALUATION

To fully evaluate the effectiveness of our proposed approach, KPIRoot, we use three real-world monitoring KPI datasets from the cloud service systems of Cloud $\mathcal{H}$. Particularly, we aim to answer the following research questions (RQs):

- RQ1: How effective is KPIRoot compared with KPI root cause localization baselines?
- RQ2: How effective is each component of KPIRoot in root cause localization?
- RQ3: How efficient is KPIRoot in localizing root cause KPIs compared to baselines?

Table 6.1: Statistics of Industrial Dataset

Industrial	Dataset A	Dataset B	Dataset C
Host Clusters	16	6	7
VM Number	120∼803	21∼26	41∼57
KPI Length	5,928,480	17,040	37,200
Root Causes	4∼36	3∼8	2∼15

6.4.1 Experiment Setting

Datasets

To confirm the practical significance of KPIRoot, we collect three datasets from large-scale online services in three Available Zones (AZs) of Cloud $\mathcal{H}$. The statistics of three industrial datasets are shown in Table 6.1. Various VM KPIs and alarm KPIs monitor the status of the service. The VM KPIs typically measure the healthy status of each VM, including resource usage metrics like CPU, memory, I/O, and bandwidth usage. The alarm KPI monitors the runtime status at the host cluster level, which is usually positively correlated to the VM KPIs. Both the artifacts and data are available on https://github.com/WenweiGu/KPIRoot.

Evaluation Metrics

In the following experiments, the F1-score is utilized to evaluate the performance of root cause localization results. We employ Precision: $PC = \frac{TP}{TP+FP}$, Recall: $RC = \frac{TP}{TP+FN}$, F1 score: $F1 = 2 \cdot \frac{PC \cdot RC}{PC+RC}$. To be specific, TP is the number of correctly localized VM KPIs; FP is the number of incorrectly predicted VM KPIs; FN is the number of root cause VM KPIs that failed to be predicted by the model. F1 score is the harmonic mean of the precision and recall. In real-world applications, since the number of root cause KPIs is unknown, software engineers will first investigate top k recommended results by root cause localization methods. Hit Rate@k is a widely used metric to measure whether the correct root causes (in our case, the root cause VM KPIs) are within the recommended top k results. We adopt Hit Rate@5 and Hit Rate@10 as evaluation metrics in our experiments.

6.4.2 Experimental Results

RQ1 The effectiveness of KPIRoot

To answer this research question, we compare the performance of KPIRoot with three statistical correlation measurements based methods, namely, Kendall correlation, Spearman correlation, and CloudScout [222], a DTW distance-based method AID [221], a graph centrality-based method LOUD [141], a conditional entropy-based method HALO [238] and a graph neural network based method CMMD [218]. The results are shown in Table 6.2, where the best F1 scores, Hit@5 and Hit@10 are all marked with boldface, while the second-best results are underlined. We can see that the average F1 scores, Hit@5 and Hit@10 of KPIRoot outperform all baseline methods in three datasets. In Dataset B and Dataset C, we can observe that the improvement achieved by KPIRoot is more significant than in Dataset A. This is because Dataset B and Dataset C focus on KPIs (*e.g.*, requests rate) related to the load balancer that manages the distribution of network traffic across physical machines, which makes the VM request rate anomalies inherently precede host cluster anomalies. It should be noted that, as indicated in Table 6.2, the number of root causes is often larger than 5. Therefore, not all root causes can be captured within the top 5 predictions. Given this, achieving Hit@5 scores of over 70% is significant enough, as it means our method is correctly identifying a large portion of the root causes within just the top 5 predictions. Furthermore, we observe that the F1 score and Hit@10 are high enough for industrial application, further demonstrating their effectiveness.

We can observe that baseline models like Kendall, Spearman, CloudScout, and AID have worse performance. These coefficient-based methods fundamentally measure the similarity between the shape of KPIs. However, high similarity does not necessarily imply causality because a high similarity can occur due to a shared underlying cause, rather than one KPI directly influencing another KPI.

Though CMMD has the ability to capture complex, nonlinear relationships between KPIs through graph attention neural networks and achieves a Hit@10 of 0.801∼0.848, it still falls short of considering the causality between VM KPIs and the host cluster KPI. HALO computes the conditional entropy between VM KPIs and the host KPI, which somehow alleviates the defect of neglecting the causality between KPIs. In contrast, our method incorporates both the similarity analysis through SAX representation similarity and causality analysis through the Granger causality test, leading to better root cause localization accuracy. The LOUD method applies graph centrality to pinpoint the root causes of issues. However, the way in which the graph is constructed can significantly impact the results. As a result, the LOUD method fails to deliver optimal performance, making it less effective in accurately identifying the root causes of problems in our context.

Table 6.2: Experimental Results of Different Root Cause Localization Methods

Methods	Dataset A			Dataset B			Dataset C		
	F1 Score	Hit@5	Hit@10	F1 Score	Hit@5	Hit@10	F1 Score	Hit@5	Hit@10
Kendall	0.651	0.562	0.728	0.605	0.594	0.770	0.657	0.635	0.727
Spearman	0.681	0.587	0.753	0.619	0.591	0.737	0.681	0.598	0.715
CloudScout	0.699	0.612	0.788	0.673	0.607	0.772	0.715	0.612	0.706
LOUD	0.736	0.652	0.813	0.736	0.625	0.824	0.709	0.653	0.829
AID	0.746	0.652	0.749	0.673	0.618	0.794	0.665	0.613	0.729
HALO	0.734	0.651	0.842	0.632	0.569	0.811	0.719	0.635	0.789
CMMD	0.776	0.632	0.833	0.679	0.594	0.848	0.721	0.667	0.801
KPIRoot	**0.859**	**0.731**	**0.909**	**0.860**	**0.749**	**0.946**	**0.829**	**0.713**	**0.895**

RQ2 The effectiveness of components in KPIRoot

To answer this research question, we conducted an ablation study on KPIRoot. Particularly, we compare two baseline models, removing the similarity and causality analysis part of KPIRoot to investigate the contribution of these two designs.

- *KPIRoot w/o Similarity* This baseline is a variant of KPIRoot that calculates the correlation score between KPIs merely based on the

Granger causality test.

- *KPIRoot w/o Causality* This baseline removes the causality analysis part and computes the correlation score based on the SAX representation similarity.

Table 6.3 shows the performance comparison between KPIRoot and its variants. In summary, the effectiveness of KPIRoot is enhanced with the utilization of similarity analysis and causality analysis, with the former making a more significant contribution. Indeed, the variant without the Granger causality test is better than all correlation coefficient-based methods. SAX representation captures the shape and trends in the data, rather than just the raw values, which is less sensitive to noise. Thus it allows for the detection of patterns that could be missed by other methods that focus only on pointwise correlations. In contrast, Pearson, Kendall, and Spearman correlations are susceptible to noise because they perform pointwise calculations. As such, outliers within the KPIs can have a significant impact on the results of the correlation coefficients. The variants without SAX representation similarity can still yield relatively satisfactory performance because Granger causality predicts the future values of a KPI based on its own past and the past of another KPI, which makes it powerful for identifying the potential causal relationships between two KPIs. While the Granger causality test may not capture the comprehensive and complex relationships between KPIs, it is still effective for identifying potential causality thus providing valuable insights for root cause analysis.

Table 6.3: Experimental Results of the Ablation Study of KPIRoot

Methods	Dataset A			Dataset B			Dataset C		
	F1 Score	Hit@5	Hit@10	F1 Score	Hit@5	Hit@10	F1 Score	Hit@5	Hit@10
KPIRoot *w/o* Similarity	0.735	0.646	0.797	0.709	0.694	0.777	0.659	0.627	0.780
KPIRoot *w/o* Causality	0.801	0.694	0.858	0.748	0.706	0.869	0.731	0.675	0.823
KPIRoot	**0.859**	**0.731**	**0.909**	**0.860**	**0.749**	**0.946**	**0.829**	**0.713**	**0.895**

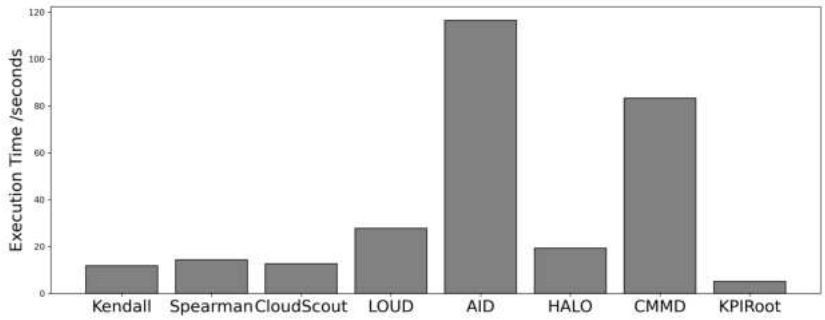


Figure 6.5: Root Cause Localization Time for All Methods

RQ3 The efficiency of KPIRoot

In this section, we evaluate the efficiency of KPIRoot in large-scale cloud systems of Cloud $\mathcal{H}$. The average running time of each method is shown in Fig. 6.5, from which we can observe that KPIRoot is the most efficient with an average execution time of around only 5 seconds, which suggests that KPIRoot is capable of providing real-time root cause analysis, meeting the requirements of large-scale cloud systems where timely identification of root causes is critical. The observed result is aligned with the time complexity analysis detailed in Section 6.3.6. As for methods like AID and CMMD, their performances are less than satisfactory due to their inherent computational complexities. AID, with its time complexity of $\mathcal{O}(n^2)$, suffers from an average runtime of more than one hundred seconds. On the other hand, CMMD, which applies graph attention neural networks, requires high computational resources, which also leads to a slower execution time and makes it less efficient. Therefore, both AID and CMMD fail to deliver the desired levels of efficiency, particularly in large-scale, real-time environments. Baseline methods like Kendall and Spearman may seem appealing due to their lower computation times. However, these apparent gains are offset by their inferior accuracy levels. As a result, their use can lead to inaccurate root-cause diagnoses and subsequently ineffective problem-solving solutions.

In summary, the evaluation results highlight KPIRoot's superior-

ity in terms of both efficiency and accuracy, which makes KPIRoot a highly promising tool for conducting real-time root cause analysis within large-scale cloud systems.

6.5 Industrial Experience

In this section, we share our experience of deploying KPIRoot in the cloud system of Cloud $\mathcal{H}$, a full-stack cloud system that consists of an infrastructure layer, a platform layer, and an application layer. To support a large number of customers, each of our services is supported by multiple clusters with tens of hundreds of virtual instances (*e.g.*, virtual router) or devices. The collective workload of each cluster is continuously monitored using an alarm KPI. When abnormal traffic impacts these services, for instance, due to overwhelming requests overloading a service, an anomaly is swiftly detected based on the alarm KPI. This triggers a root cause analysis procedure to pinpoint the specific nodes (e.g., VMs) and take prompt mitigating actions. In our previous practice, manual inspection is feasible given the limited scale of each cluster. So, we can check each specific KPI of the node, compare it with the alarm KPI (with similarity comparison tools), and find the root cause. However, this process proved to be error-prone and labor-intensive, particularly as the scale of each service expanded. On average, it took between thirty minutes to one hour to identify and mitigate the root causes.

To alleviate these issues, we have deployed KPIRoot in Cloud $\mathcal{H}$ since *Nov 2022*. Specifically, KPIRoot operates by automatically fetching KPIs collected from the monitoring backends and applying the algorithm to calculate the correlation score in real time. Using KPIRoot, the potential root causes are returned to engineers. In addition, visualization tools are provided, making it easier for engineers to understand the system's behavior and performance.

In Fig. 6.6, we demonstrate the practical application of the root cause analysis tool KPIRoot in real industrial scenarios. While our

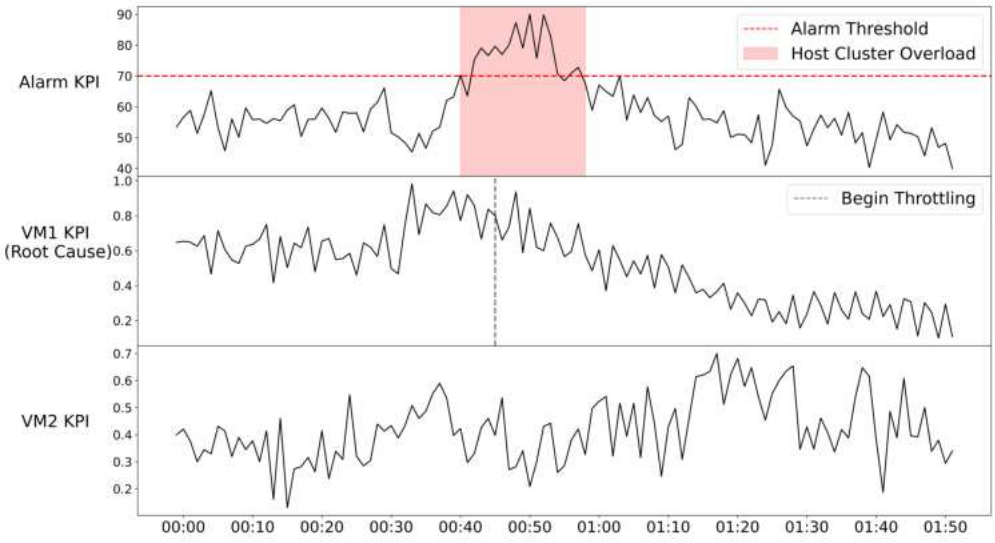


Figure 6.6: Case Study of KPIRoot

system may encompass tens to hundreds of KPIs, as outlined in Table 6.1, for the sake of conciseness, we showcase only two KPIs in this illustration. In this case, we initially received an alert indicating that the overall traffic for the host cluster had abruptly surpassed the predefined threshold. This requires immediate measures to pinpoint the root cause and throttle its throughput to avoid resource exhaustion within the cluster. However, this is quite challenging given the large number of KPIs needed to check, and the root-cause KPI may not be readily identifiable visually, as its shape similarity may not correspond directly with the alarm KPI. Given that, the root cause analysis takes tens of minutes to one hour to check manually, leading to delayed mitigation of the sudden traffic spike. With KPIRoot, the root cause of KPI can be quickly localized, generally within five minutes. With this result, we throttle the throughput of VM1 immediately after the alarm KPI is fired. As shown in Figure 6.6, the overall traffic is limited, and the alarm KPI returns back to a normal range quickly.

KPIRoot has been deployed in all major regions of our com-

pany, covering eighteen critical network services, *e.g.*, Linux Virtual Server (LVS), NGINX, Network Address Translation (NAT), and DNS services. It has been serving in our production environment for more than ten months, reducing the average root cause localization time from 30 minutes to 5 minutes. Following the deployment of the KPIRoot service, the feedback from engineers has been overwhelmingly positive. In terms of computational efficiency, KPIRoot has reduced the computational load significantly compared to previous methods. The system can perform real-time RCA, identifying potential issues quickly and allowing engineers to take immediate action. In terms of accuracy, KPIRoot's design of combining similarity and causality analysis has proven highly precise in identifying root causes. This leads to more effective problem resolution and significantly reduced revenue loss.

6.6 Discussion

In this section, we discuss the difference between our approach and existing root cause analysis approaches for microservice systems and why they are not applicable in our industrial scenario. Besides, we discuss the influence of SAX representation in KPIRoot. Finally, we identified some potential threats to the validity of our study.

6.6.1 Root Cause Analysis for Microservice System

Our objective shares some similarities with root cause analysis in microservice systems, however, there are several main differences in terms of the application scenarios. Firstly, rather than localizing the root causes of application/service failures in microservice systems, where these applications are at the same level, our problem is top-down root localization. When we observe an anomaly at the system level, we investigate and analyze the underlying VM instance-level information. Secondly, due to VM isolation, each VM

instance operates independently and is isolated from other VMs and the host system. This leads to sparse or even non-existent invocation dependency among them, making the construction of a service dependency graph as done in existing works very challenging.

Existing Methods like FRL-MFPG [21] and ServiceRank [134] rely on the construction of a service dependency graph and the execution of a second-order random walk, which can become highly time-consuming with complexity exceeding $O(n^2)$. As for HRLHF [200], the large graph size makes causal discovery computationally intensive. Furthermore, the delay incurred by waiting for engineers to provide human feedback poses an additional obstacle for real-time localization. However, the analysis delay should be less than the sampling interval, *e.g.*, 1 minute in our practical scenarios, making these methods unsuitable for industrial deployment.

6.6.2 The Influence of SAX Representation

The anomaly segment detection serves as a precursor to root cause localization and may influence the subsequent task. Since SAX representation is a downsampling method, it may induce the omission of the original KPI information. However, the alarm KPI is carefully selected by experienced engineers based on their experience. These KPIs typically exhibit distinct anomaly patterns when system-level issues occur, making them highly reliable system health indicators, even if SAX may cause some content of information loss. Given the detected anomalies in alarm KPIs, our target is to find the VMs causing performance anomalies, which should also have obvious patterns in the VM KPI. So the SAX representation has little influence on the localization part.

6.6.3 Threats to Validity

We have identified the following potential threats to the validity of our study:

Internal threats. The implementation of baselines and parameter settings constitutes one of the internal threats to our work's validity. To mitigate these threats, we utilized the open-sourced code released by the authors of the papers or packages on GitHub for all baselines. As for our proposed approach, the source code has been reviewed meticulously by the authors, as well as several experienced software engineers, to minimize the risk of errors and increase the overall confidence in our results. For parameter settings, as our algorithm KPIRoot has few parameters, we find the most suitable configurations based on the best results obtained in different parameters.

External threats. Our experiments are conducted based on real-world datasets collected from Cloud $\mathcal{H}$ over more than two years. The evaluation requires engineers to inspect and label the root cause KPIs manually. Label noises are inevitable during the manual labeling process. However, alleviation strategies taken by engineers further ensure the accuracy of labeled root causes. Therefore, we believe the amount of noise is small and does not have a significant impact on the experiment results. On the other hand, the results may vary between different cloud service providers, industries, or specific use cases. Nevertheless, we believe that our experimental results, obtained from large-scale online systems within a prominent cloud service company serving millions of users, can demonstrate the generality and effectiveness of our proposed approach, KPIRoot.

6.6.4 Limitations of KPIRoot

While KPIRoot demonstrates state-of-the-art performance and practical utility in localizing root causes, its current design presents several avenues for future improvement. This section details two key limitations and proposes corresponding enhancements to broaden its applicability and accuracy.

First, the current efficacy of KPIRoot is primarily centered on performance issues that manifest as trend variations, where root-

cause KPIs exhibit anomalous trends similar to the overall service KPI. However, performance degradation in complex cloud systems can arise from various anomaly types, such as sudden spikes, transient glitches, or shifts in data variance, which may not present as clear correlative trends. This focus implicitly limits KPIRoot's ability to diagnose issues whose root causes have a different morphological signature. A significant direction for future work is to decouple anomaly detection from the core root cause analysis logic. This could be achieved by integrating a more comprehensive, multi-pattern anomaly detection module to identify a broader spectrum of anomalous behaviors across all KPIs. The similarity and causality analysis of KPIRoot could then be applied specifically to this pre-filtered set of anomalous KPIs, making the entire framework more robust and versatile in handling diverse, real-world failure scenarios.

Second, a limitation arises from the information compression inherent in the SAX representation. While SAX effectively improves computational efficiency by converting a time-series segment into a single symbol, it discards the sub-window dynamics. For instance, two KPI segments might be assigned the same high-value symbol, but one could be on a sharp upward trajectory while the other is declining. This loss of intra-window trend information can be critical for precise causality assessment. To address this, a promising future enhancement is to enrich the symbolic representation itself. We propose augmenting the standard SAX alphabet with a secondary set of symbols that explicitly encode the intra-window trend *e.g.*, increasing, decreasing, or stable. This dual-component symbolic representation would provide a more nuanced input to the causality analysis module, potentially improving localization accuracy without fully sacrificing the significant efficiency gains afforded by the symbolic approach.

6.7 CONCLUSION

In this chapter, we propose KPIRoot, an effective and efficient framework for root cause analysis in practical cloud systems with monitoring KPIs. Specifically, KPIRoot integrates the strength of similarity analysis and causality analysis, offering a more comprehensive correlation evaluation of KPI, thus enhancing the accuracy of root cause localization. Additionally, the utilization of SAX representation of KPI significantly improves the efficiency of the method. Extensive experiments on three industrial datasets show that KPIRoot achieves 0.850 F1-Score and 0.916 Hit@10 with the highest efficiency, outperforming all the baselines. Moreover, the successful deployment of our approach in large-scale industrial applications further demonstrates its practicality.

□ **End of chapter.**

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Cloud systems play a crucial role in supporting daily activities by offering scalable and accessible computing resources, making their reliability essential for ensuring seamless and uninterrupted services. However, achieving high reliability in these systems is challenging due to their vast scale and inherent complexity. This thesis explores our research efforts to tackle these challenges, presenting innovative approaches and solutions designed to enhance the reliability of cloud systems in the face of increasing demands and complex architectures.

In chapter 4, we address the critical challenge of identifying performance issues in cloud systems, which are often comprised of various components like microservices that can lead to service-level agreement violations and financial losses. Traditional methods, which focus on independent analysis of metrics, fail to account for complex dependencies among components. While some graph neural network-based approaches consider temporal and relational information, they struggle to detect correlation violations in metrics that indicate underlying issues. Additionally, the vast volume of components results in a myriad of noisy metrics, complicating engineers' ability to accurately identify performance issues. To overcome these limitations, we propose ISOLATE, a learning-based approach that

harnesses relational and temporal features of metrics to pinpoint performance issues. ISOLATE employs a graph neural network with attention to characterize metric relations and utilizes a GRU and a convolution network to extract long-term and multi-scale temporal patterns. The approach also incorporates a positive unlabeled learning strategy to mitigate noisy data impact by tagging pseudo labels from confirmed negative examples. Our extensive evaluation reveals ISOLATE's superior performance, achieving a 0.945 F1-score and 0.920 Hit rate@3, and its practical success in identifying performance issues in Huawei Cloud underscores its effectiveness.

In chapter 5, we tackle the challenge of practical anomaly detection in cloud systems, where traditional methods struggle to detect diverse and evolving anomalies across services and fail to bridge the gap between technical and business interpretations of anomalies. To address these issues, we introduce ADAMAS, an adaptive AutoML-based framework designed to enhance anomaly detection capabilities in production environments. ADAMAS features a novel unsupervised evaluation function to optimize model structure and parameters for cross-service anomaly detection and integrates a lightweight human-in-the-loop design to incorporate expert knowledge, adapting to evolving anomaly patterns while connecting predicted anomalies to actual business exceptions. By monitoring misprediction rates, ADAMAS dynamically reconfigures its optimal model, establishing a continuous improvement loop. Evaluations show ADAMAS's superiority over baseline models with a 0.891 F1-score, and the ablation study confirms the effectiveness of its evaluation function and expert knowledge integration.

In chapter 6, we confront the challenge of root cause localization in cloud systems, essential for diagnosing and resolving performance issues efficiently. Existing methods typically focus on identifying KPIs with trends similar to overall service performance, which is insufficient for systems with complex interdependent services. Although deep learning-based approaches offer improved modeling

of dependencies, they suffer from high computational demands and lack interpretability. To address these limitations, we propose KPIRoot, a method integrating similarity analysis and causality analysis for effective root cause localization. KPIRoot measures trend alignment and sequential variation order of KPIs, leveraging symbolic aggregate approximation for compact KPI representation, enhancing efficiency. Our experimental results demonstrate KPIRoot's superiority, outperforming seven state-of-the-art baselines by 7.9% to 28.3%, while reducing time cost by 56.9%. The deployment of KPIRoot in Huawei Cloud showcases its practical efficacy in large-scale production environments.

7.2 Future Work

Ensuring the reliability of software systems is a significant task. This thesis primarily focuses on enhancing the reliability management of large-scale cloud systems. As large language models (LLMs) are reshaping the world and gaining significant traction in various industries, new research problems and challenges arise in the era of LLMs. My future work focuses on enhancing the reliability of LLM systems.

Large language models (LLMs) have emerged as epochal technologies, with recent advancements significantly enhancing their capabilities [84]. These models have demonstrated remarkable potential across various domains, including machine translation, conversational agents, and even AIOps-related tasks, such as log parsing [85]. The scaling law [91] dictates that model size and training data size are critical factors that determine the model's capability. Consequently, there has been a rapid increase in dataset sizes and the number of parameters in LLMs, *i.e.*, with trillions of parameters on trillions of tokens. For example, the GPT-4 model [1] demonstrates this growth with its 1.8T parameters. Training LLMs with such huge sizes is a daunting task, requiring distributed model train-

ing on large-scale GPU clusters. It involves a huge amount of computation, communication, and storage resources, as well as software support for the training tasks. Consequently, the possibility of task failures can be high. According to the experience of ByteDance [36], the economic loss for a customer can reach more than $ 1,700 in a 128-machine task lasting 40 minutes. If the training process is frequently interrupted by such faults, operating expenses and time costs will increase significantly, which will cause customer dissatisfaction. Thus, the reliability of LLM training systems is crucial, and there is a critical need for intelligent monitoring and diagnosis for LLM training systems.

To improve the reliability of LLM training systems, we will focus on predicting failures of high-bandwidth memory as our future work.

7.2.1 Failure Prediction of High-bandwidth Memory in LLM Training Platforms

With the emergence of large language models (LLMs), *e.g.*, GPT-4 and LLaMA, there has been a considerable increase in the scale of ultra-massive datasets, leading to a growing demand for rapid computation [95]. There is an exponentially growing trend of AI accelerators, *e.g.*,, neural-network processing units (NPU), that focus on increasing computing performance [105]. However, the growing exponent for GPUs or NPUs is substantially larger than that for dynamic random-access memories (DRAMs) [46]. Thus, the latency of data movement between memory and AI accelerators is becoming the bottleneck in large-scale AI model training [92]. The increasing gap between the computing power and the memory bandwidth is known as the memory wall, especially in modern LLM training systems [211]. Recently, high-bandwidth memory (HBM) has gained tremendous attention as a promising solution to alleviate memory bottlenecks fundamentally [95, 185, 232] and has established itself

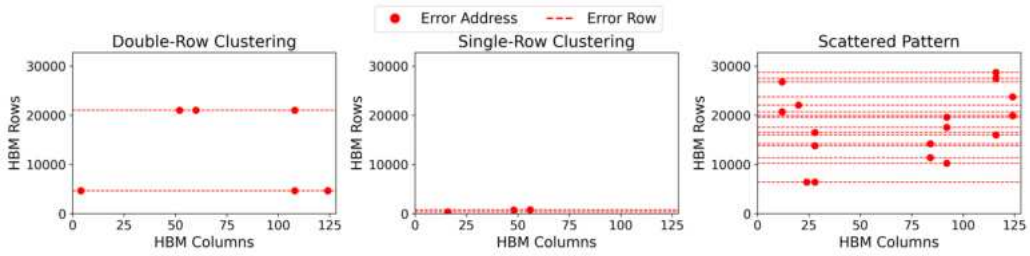


Figure 7.1: Examples of Bank-level Failure Patterns

as the memory solution for LLM training.

In modern LLM training systems, hardware failures, especially memory failures, are among the most significant reasons for training job crashes or slowdowns [14, 28, 40, 43, 213, 236]. Unfortunately, due to its stacking structure, HBMs not only exhibit the errors of DRAM but also suffer from new errors, *e.g.*, TSV faults [8, 151]. Besides, conventional error correction codes (ECC) are insufficient to correct malfunctions of sub-wordline drivers (SWDs) in HBMs [149], a primary cause of errors, making HBM even more vulnerable to unexpected errors. Thus, it is of great significance to proactively predict HBM failures to ensure the reliability of the high-performance computing platform that serves for AI model training.

As an initial exploration, we have conducted an empirical study to understand the characteristics of HBMs in the LLM training platform, especially in the sudden uncorrectable error required action (UER), the bank-level failure pattern, and the locality of UER rows, which motivates a preliminary method design.

Sudden UER Ratio. As described in the literature [228, 229], there are two types of UERs: sudden UERs, which result from component malfunctions that immediately corrupt data, and non-sudden UERs, which are predictable and initially appear as CEs and UEOs but evolve into UERs over time. Sudden UERs are typically considered unpredictable, meaning existing in-row failure prediction frameworks cannot effectively address them. In Table 7.1, we present the ratio of sudden UERs observed in an industrial dataset,

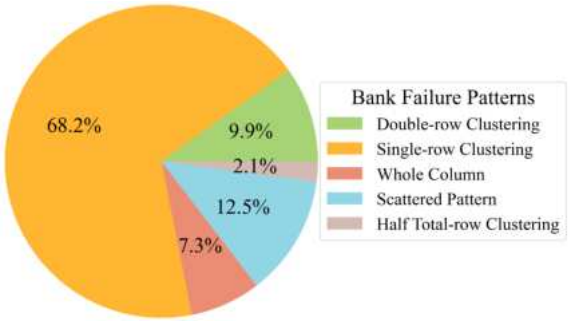


Figure 7.2: Bank Failure Pattern Distribution

Table 7.1: In-row Predictable Ratio of UERs

Micro-level	Sudden UER	Non-sudden UER	Predictable Ratio
NPU	243	175	41.86%
HBM	246	175	41.56%
SID	260	180	40.91%
PS-CH	311	185	37.29%
BG	434	252	36.73%
Bank	760	314	29.23%
Row	4980	229	4.39%

which includes more than 10,000 NPUs and 80,000 HBMs. Our analysis reveals that the behavior of sudden UERs in HBMs is markedly different from that of traditional DDR4 and DDR5 memory systems. Specifically, the ratio of sudden UERs increases drastically as we move from the NPU level to the row level, with sudden row UERs accounting for more than 95% of all UERs. This underscores the limitations of existing in-row prediction methods, rendering them impractical for managing sudden UERs at the row level.

Bank-Level Failure Patterns. Our empirical analysis has identified several distinct failure patterns at the bank level: double-row clustering pattern (including half total-row clustering), single-row clustering pattern, scattered pattern, and a special case of scattered pattern known as the whole column pattern with the error dispread

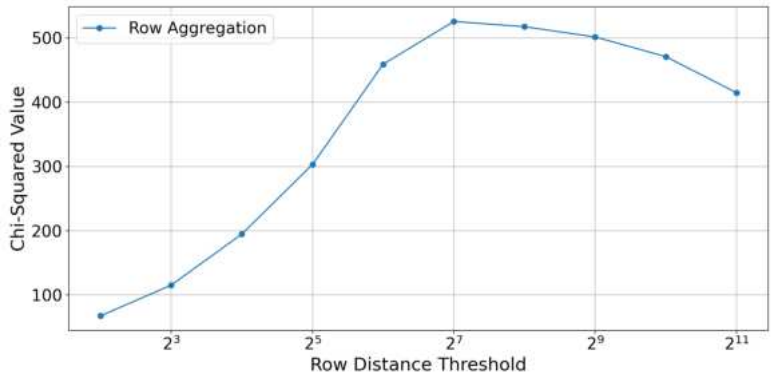


Figure 7.3: Statistic Significance of Difference Distance Thresholds

in nearly all the rows. Figure 7.1 illustrates examples of these failure patterns. The single-row clustering pattern, comprising 68.2% of observed UER banks, is characterized by errors concentrated within a contiguous, narrow area, facilitating easier failure prediction due to its spatial locality. Double-row clustering, which includes the half total-row clustering variant, accounts for 9.9% of UERs. This pattern features two clusters of UERs with a consistent interval between them, making prediction manageable by leveraging the predictable spacing between clusters. The scattered pattern is more complex, with UERs distributed irregularly across the bank, representing 12.5% of UERs. Within this category, column failure—a special case where UERs appear across all rows of a column—accounts for 7.3%. This pervasive distribution necessitates bank-sparing techniques, as the unpredictable nature of errors complicates row-level failure mitigation.

Our analysis of the industrial dataset, as depicted in Figure 7.2, reveals that while scattered patterns pose challenges, the prevalence of aggregation patterns (78.1% combined) indicates that cross-row failure prediction remains feasible for the majority of cases. This insight underscores the practical applicability of targeted prediction strategies in managing memory failures within high-performance computing systems.

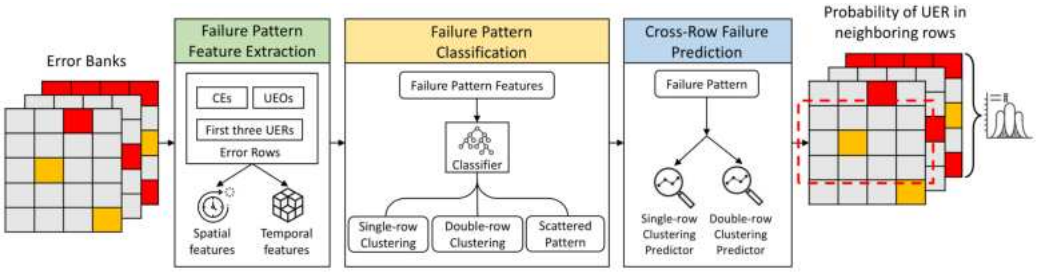


Figure 7.4: The Overview of Our Proposed Method Cordial

Locality of Cross-row UER. Since the single-row clustering pattern is characterized by UERs concentrated within a narrow and contiguous area, this spatial concentration can be advantageous for cross-row UER prediction, as it suggests that subsequent UERs are likely to occur in the vicinity of the existing UER row. Thus, we explore the locality of cross-row UERs to determine the effective range within which predictions can be made. To quantify this locality, we compute the chi-square statistic of subsequent UERs occurring within various row distance thresholds from the current UER row. These thresholds range from 4 to 2048 rows. Our analysis indicates that the strongest statistical significance is achieved at a threshold of 128 rows, as shown in Figure 7.3. This suggests that predicting UERs within a 128-row range is both manageable and effective, enabling us to focus our prediction efforts and redundant resources on the most likely areas for subsequent UERs.

Based on insights from our empirical study, we propose the Cross-row Failure Prediction Method Based on Bank-level Error Locality (Cordial). Cordial is designed to predict failures in a cross-row manner, effectively addressing the limitations of existing in-row prediction methods.

The overall workflow of Cordial is illustrated in Figure 7.4, which consists of three stages: failure pattern feature extraction, failure pattern classification and cross-row failure prediction. We first collect the raw error log from the baseboard management controller

Table 7.2: Performance of Different Failure Prediction Methods

Methods	Precision	Recall	F1 Score	ICR (%)
Neighbor Rows	0.322	0.393	0.347	13.31%
Cordial-LGBM	0.642	0.504	0.563	18.60%
Cordial-XGB	0.732	0.509	0.591	18.87%
Cordial-DT	**0.806**	**0.580**	**0.662**	**19.58%**

(BMC) and then generate a set of spatial and temporal features (*e.g.*, the average row difference between two successive UER rows) with all CEs, UEOs and the first three UERs for each bank. Then, we use the generated features to train tree-based predictors and output the failure pattern of the current bank. We finally utilize the cross-row UER predictors to anticipate whether there are UER rows in the neighboring rows for aggregation patterns and conduct isolation for these predicted error rows. Otherwise, all banks with scattered row patterns will be isolated directly.

7.2.2 Preliminary Evaluation Results

The preliminary experimental result is shown in Table 7.2. As previously mentioned, the sudden UER ratio can reach as high as 95.61% at the row level, indicating that existing methods are ideally capable of predicting only 4.39% of UERs. Therefore, to ensure a fair comparison, we benchmark our approach against an industrial baseline. This baseline method isolates the eight rows adjacent to an identified UER row, aiming to prevent further propagation of errors within the immediate vicinity. Compared to this baseline, our method demonstrates significantly improved performance across various metrics, including weighted precision, recall, and F1 score of all prediction blocks, as well as in practical industrial evaluation metrics such as Isolation Coverage Rate (ICR), which is shown in Table 7.2. Notably, the strong performance of Random Forest aligns with the re-

sults of the failure pattern classification, reinforcing its effectiveness. Though our method achieves a 19.58% isolation coverage rate due to the inherent randomness of UER rows that adds complexity to the problem, it is substantially higher compared to traditional in-row failure prediction (19.58% over 4.39%). This randomness underscores the challenge of accurately predicting failure patterns, yet our framework's intelligent block selection strategy effectively addresses these difficulties. By adopting our new paradigm of cross-row failure prediction, we believe industries can achieve more robust and proactive management of memory failures, ultimately improving system stability and performance.

7.2.3 Limitations of Our Preliminary Approach

We further recognize three main limitations of our preliminary method, Cordial, which we will address accordingly in our future work:

- First, in Cordial, the bank group information is not leveraged. However, our observations indicate that, in cases of multi-bank error HBM, there is a significant correlation between bank-level failure patterns and error rows across different banks within the same bank group. This information could enhance both failure pattern prediction and row-level failure prediction. To address this limitation, we propose incorporating bank group-related features into the prediction model, which allows for a more precise identification of error patterns across related banks.

- Second, Cordial relies on block-level prediction, which can be overly coarse-grained. The resource cost to isolate potential error rows can significantly exceed that of directly predicting a single block that encompasses the true UER. We propose leveraging a temporal-spatial regressor instead of a multi-block classification approach. This method would refine the prediction

granularity, reducing unnecessary isolation of non-error rows and optimizing resource usage.

- Third, the block size used in Cordial's prediction is fixed. However, the potential range for covering errors can vary with different patterns, and this inflexibility can affect prediction accuracy and increase the isolation resource budget. To address this, we suggest developing a dynamic block sizing mechanism that adapts to different error patterns, thereby improving prediction accuracy and optimizing resource allocation for isolation.

□ **End of chapter.**

Chapter 8

List of Publications

1. Wenwei Gu, Jiazhen Gu, Renyi Zhong, Wenyu Zhang, Ming Li and Michael R. Lyu. "Cordial: Cross-row Failure Prediction Method Based on Bank-level Error Locality for HBMs." In Proceedings of the 55th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2025), Industry Track.

2. Zhihan Jiang, Rui Ren, Guangba Yu, Yulun Wu, Wenwei Gu, Yichen Li, Yujie Huang, Cong Feng, Zengyin Yang, Yongqiang Yang and Michael R. Lyu. "LLMPrism: Black-box Performance Diagnosis for Production LLM Training Platforms." In Proceedings of the 55th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2025), Industry Track.

3. Renyi Zhong, Yichen Li, Jinxi Kuang, Wenwei Gu, Yintong Huo, and Michael R. Lyu. "LogUpdater: Automated Detection and Repair of Specific Defects in Logging Statements." In ACM Transactions on Software Engineering and Methodology (TOSEM).

4. Wenwei Gu, Jiazhen Gu, Jinyang Liu, Zhuangbin Chen, Jianping Zhang, Jinxi Kuang, Cong Feng, Yongqiang Yang and Michael R. Lyu. "ADAMAS: Adaptive Domain-Aware Per-

formance Anomaly Detection in Cloud Service Systems." In Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE 2025).

5. Wenwei Gu, Jinyang Liu, Zhuangbin Chen, Jianping Zhang, Yuxin Su, Jiazhen Gu, Cong feng, Zengyin Yang, Yongqiang Yang and Michael R. Lyu. "Identifying Performance Issues in Cloud Service Systems Based on Relational-Temporal Features." In ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 34, Issue 3.

6. Wenwei Gu, Xinying Sun, Jinyang Liu, Yintong Huo, Zhuangbin Chen, Jianping Zhang, Jiazhen Gu, Yongqiang Yang and Michael R. Lyu. "KPIRoot: Efficient Monitoring Metric-based Root Cause Localization in Large-scale Cloud Systems." In Proceedings of the 35th International Symposium on Software Reliability Engineering (ISSRE 2024).

7. Jianping Zhang, Wenwei Gu, Yizhan Huang, Zhihan Jiang, Weibin Wu and Michael R. Lyu. "Curvature-Invariant Adversarial Attacks for 3D Point Clouds." In Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI 2024).

8. Yun Peng, Shuqing Li, Wenwei Gu, Yichen Li, Wenxuan Wang, Cuiyun Gao and Michael R. Lyu. "Revisiting, Benchmarking and Exploring API Recommendation: How Far Are We?" In IEEE Transactions on Software Engineering (TSE), Volume 49, Issue 4.

Chapter 4 is an adapted reprint of the publication "Identifying Performance Issues in Cloud Service Systems Based on Relational-Temporal Features." that was published in the ACM Transactions on Software Engineering and Methodology Volume 34, Issue 3 (TOSEM). The thesis author is the primary author of this publication. This is a joint work with Jinyang Liu, Jianping Zhang, Jiazhen Gu, and

Michael R. Lyu from The Chinese University of Hong Kong. Additionally, Zhuangbin Chen and Yuxin Su from the School of Software Engineering, Sun Yat-sen University, also played a significant role. The collaboration extends to Cong Feng, Zengyin Yang, and Yongqiang Yang from the Computing and Networking Innovation Lab at Huawei Cloud Computing Technology Co., Ltd.

Chapter 5 is an adapted reprint of the publication "ADAMAS: Adaptive Domain-Aware Performance Anomaly Detection in Cloud Service Systems." that was published in the IEEE/ACM 47th International Conference on Software Engineering (ICSE 2025). The thesis author is the primary author of this publication. This is a joint work with Jiazhen Gu, Jinyang Liu, Jianping Zhang, Jinxi Kuang, and Michael R. Lyu from The Chinese University of Hong Kong. Additionally, Zhuangbin Chen from the School of Software Engineering, Sun Yat-sen University, also played a significant role. The collaboration extends to Cong Feng and Yongqiang Yang from the Computing and Networking Innovation Lab at Huawei Cloud Computing Technology Co., Ltd.

Chapter 6 is an adapted reprint of the publication "KPIRoot: Efficient Monitoring Metric-based Root Cause Localization in Large-scale Cloud Systems." that was published in the 35th International Symposium on Software Reliability Engineering (ISSRE 2024). The thesis author is the primary author of this publication. This is a joint work with Jinyang Liu, Yintong Huo, Jianping Zhang, Jiazhen Gu, and Michael R. Lyu from The Chinese University of Hong Kong. Additionally, Zhuangbin Chen from the School of Software Engineering, Sun Yat-sen University, also played a significant role. The collaboration extends to Xinying Sun and Yongqiang Yang from the Computing and Networking Innovation Lab at Huawei Cloud Computing Technology Co., Ltd.

□ **End of chapter.**

Bibliography

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] S. Agarwal, S. Chakraborty, S. Garg, S. Bisht, C. Jain, A. Gonuguntla, and S. Saini. Outage-watch: Early prediction of outages using extreme event regularizer. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 682–694, 2023.

[3] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[4] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang. {CherryPick}: Adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 469–482, 2017.

[5] Y. Amannejad, D. Krishnamurthy, and B. Far. Detecting performance interference in cloud-based web services. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 423–431. IEEE, 2015.

[6] A. Arnold, Y. Liu, and N. Abe. Temporal causal modeling with graphical granger methods. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 66–75, 2007.

[7] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga. Usad: Unsupervised anomaly detection on multivariate time series. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 3395–3404, 2020.

[8] K. Bae and J. Park. Efficient tsv fault detection scheme for high bandwidth memory using pattern analysis. In *2020 International SoC Design Conference (ISOCC)*, pages 19–20. IEEE, 2020.

[9] M. Bahri, F. Salutari, A. Putina, and M. Sozio. Automl: state of the art with a focus on anomaly detection, challenges, and research directions. *International Journal of Data Science and Analytics*, 14(2):113–126, 2022.

[10] M. Balandat, B. Karrer, D. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy. Botorch: A framework for efficient monte-carlo bayesian optimization. *Advances in neural information processing systems*, 33:21524–21538, 2020.

[11] M. S. Bali and S. Khurana. Effect of latency on network and end user domains in cloud computing. In *2013 International Conference on Green Computing, Communication and Conservation of Energy (ICGCE)*, pages 777–782. IEEE, 2013.

[12] S. Belakaria, A. Deshwal, and J. R. Doppa. Max-value entropy search for multi-objective bayesian optimization. *Advances in neural information processing systems*, 32, 2019.

[13] A. Beloglazov and R. Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance

efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420, 2012.

[14] I. Boixaderas, D. Zivanovic, S. Moré, J. Bartolome, D. Vicente, M. Casas, P. M. Carpenter, P. Radojković, and E. Ayguadé. Cost-aware prediction of uncorrected dram errors in the field. In *SC20: International Conference for High-Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2020.

[15] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini. Anomaly detection using autoencoders in high performance computing systems. In *Proceedings of the AAAI Conference on artificial intelligence*, volume 33, pages 9428–9433, 2019.

[16] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[17] S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

[18] C. Cangea, P. Veličković, N. Jovanović, T. Kipf, and P. Liò. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287*, 2018.

[19] C. Chai, L. Cao, G. Li, J. Li, Y. Luo, and S. Madden. Human-in-the-loop outlier detection. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 19–33, 2020.

[20] X. Chen, J. Shi, J. Chen, P. Wang, and W. Wang. High-precision online log parsing with large language models. In

Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings, pages 354–355, 2024.

[21] Y. Chen, D. Xu, N. Chen, and X. Wu. Frl-mfpg: Propagation-aware fault root cause location for microservice intelligent operation and maintenance. *Information and Software Technology*, 153:107083, 2023.

[22] Z. Chen, D. Chen, X. Zhang, Z. Yuan, and X. Cheng. Learning graph structures with transformer for multivariate time-series anomaly detection in iot. *IEEE Internet of Things Journal*, 9(12):9179–9189, 2021.

[23] Z. Chen, Z. Jiang, Y. Su, M. R. Lyu, and Z. Zheng. Tracemesh: Scalable and streaming sampling for distributed traces. In *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*, pages 54–65. IEEE, 2024.

[24] Z. Chen, Y. Kang, L. Li, X. Zhang, H. Zhang, H. Xu, Y. Zhou, L. Yang, J. Sun, Z. Xu, et al. Towards intelligent incident management: why we need it and how we make it. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1487–1497, 2020.

[25] Z. Chen, J. Liu, Y. Su, H. Zhang, X. Ling, Y. Yang, and M. R. Lyu. Adaptive performance anomaly detection for online service systems via pattern sketching. In *Proceedings of the 44th International Conference on Software Engineering*, pages 61–72, 2022.

[26] Z. Chen, J. Liu, Y. Su, H. Zhang, X. Wen, X. Ling, Y. Yang, and M. R. Lyu. Graph-based incident aggregation for large-scale online service systems. In *2021 36th IEEE/ACM In-*

ternational Conference on Automated Software Engineering (ASE), pages 430–442. IEEE, 2021.

[27] Q. Cheng, D. Sahoo, A. Saha, W. Yang, C. Liu, G. Woo, M. Singh, S. Saverese, and S. C. Hoi. Ai for it operations (aiops) on cloud platforms: Reviews, opportunities and challenges. *arXiv preprint arXiv:2304.04661*, 2023.

[28] Z. Cheng, S. Han, P. P. Lee, X. Li, J. Liu, and Z. Li. An in-depth correlative study between dram errors and server failures in production data centers. In *2022 41st International Symposium on Reliable Distributed Systems (SRDS)*, pages 262–272. IEEE, 2022.

[29] M. B. Chhetri, Q. B. Vo, and R. Kowalczyk. Cl-slam: Cross-layer sla monitoring framework for cloud service-based applications. In *Proceedings of the 9th International Conference on Utility and Cloud Computing*, pages 30–36, 2016.

[30] G. Chu, J. Wang, Q. Qi, H. Sun, Z. Zhuang, B. He, Y. Jing, L. Zhang, and J. Liao. Anomaly detection on interleaved log data with semantic association mining on log-entity graph. *IEEE Transactions on Software Engineering*, 2025.

[31] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 153–167, 2017.

[32] H. Dai, H. Li, C.-S. Chen, W. Shang, and T.-H. Chen. Logram: Efficient log parsing using n n-gram dictionaries. *IEEE Transactions on Software Engineering (TSE)*, 48(3):879–892, 2020.

[33] Z. Dang, S. He, P. Hong, Z. Li, X. Zhang, X.-H. Sun, and G. Chen. Nvalloc: rethinking heap metadata management

in persistent memory allocators. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 115–127, 2022.

[34] U. Degenbaev, J. Eisinger, K. Hara, M. Hlopko, M. Lippautz, and H. Payer. Cross-component garbage collection. *Proceedings of the ACM on Programming Languages*, 2(OOPSLA):1–24, 2018.

[35] A. Deng and B. Hooi. Graph neural network-based anomaly detection in multivariate time series. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 4027–4035, 2021.

[36] Y. Deng, X. Shi, Z. Jiang, X. Zhang, L. Zhang, Z. Zhang, B. Li, Z. Song, H. Zhu, G. Liu, et al. Minder: Faulty machine detection for large-scale distributed model training. *arXiv preprint arXiv:2411.01791*, 2024.

[37] M. Du and F. Li. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 859–864. IEEE, 2016.

[38] M. Du, F. Li, G. Zheng, and V. 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.

[39] C. Duan, T. Jia, H. Cai, Y. Li, and G. Huang. Afalog: A general augmentation framework for log-based anomaly detection with active learning. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pages 46–56. IEEE, 2023.

[40] J. Duan, S. Zhang, Z. Wang, L. Jiang, W. Qu, Q. Hu, G. Wang, Q. Weng, H. Yan, X. Zhang, et al. Efficient training of large language models on distributed infrastructures: A survey. *arXiv preprint arXiv:2407.20018*, 2024.

[41] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[42] R. Fu, Z. Zhang, and L. Li. Using lstm and gru neural network methods for traffic flow prediction. In *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 324–328. IEEE, 2016.

[43] Y. Gao, X. Shi, H. Lin, H. Zhang, H. Wu, R. Li, and M. Yang. An empirical study on quality issues of deep learning platform. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 455–466. IEEE, 2023.

[44] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. *Advances in neural information processing systems*, 31, 2018.

[45] P. Garraghan, R. Yang, Z. Wen, A. Romanovsky, J. Xu, R. Buyya, and R. Ranjan. Emergent failures: Rethinking cloud reliability at scale. *IEEE Cloud Computing*, 5(5):12–21, 2018.

[46] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer. Ai and memory wall. *IEEE Micro*, 2024.

[47] E. J. Ghomi, A. M. Rahmani, and N. N. Qader. Load-balancing algorithms in cloud computing: A survey. *Journal of Network and Computer Applications*, 88:50–71, 2017.

[48] S. Ghosh, M. Shetty, C. Bansal, and S. Nath. How to fight production incidents? an empirical study on a large-scale cloud service. In *Proceedings of the 13th Symposium on Cloud Computing (SoCC)*, pages 126–141, 2022.

[49] J. Gu, C. Luo, S. Qin, B. Qiao, Q. Lin, H. Zhang, Z. Li, Y. Dang, S. Cai, W. Wu, et al. Efficient incident identification from multi-dimensional issue reports via meta-heuristic search. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 292–303, 2020.

[50] J. Gu, C. Luo, S. Qin, B. Qiao, Q. Lin, H. Zhang, Z. Li, Y. Dang, S. Cai, W. Wu, et al. Efficient incident identification from multi-dimensional issue reports via meta-heuristic search. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 292–303, 2020.

[51] J. Gu, J. Wen, Z. Wang, P. Zhao, C. Luo, Y. Kang, Y. Zhou, L. Yang, J. Sun, Z. Xu, et al. Efficient customer incident triage via linking with system incidents. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1296–1307, 2020.

[52] W. Gu, J. Liu, Z. Chen, J. Zhang, Y. Su, J. Gu, C. Feng, Z. Yang, and M. Lyu. Performance issue identification in cloud systems with relational-temporal anomaly detection. *arXiv preprint arXiv:2307.10869*, 2023.

[53] W. Gu, J. Liu, Z. Chen, J. Zhang, Y. Su, J. Gu, C. Feng, Z. Yang, Y. Yang, and M. R. Lyu. Identifying performance issues in cloud service systems based on relational-temporal

features. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2024.

[54] W. Gu, X. Sun, J. Liu, Y. Huo, Z. Chen, J. Zhang, J. Gu, Y. Yang, and M. R. Lyu. Kpiroot: Efficient monitoring metric-based root cause localization in large-scale cloud systems. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*, pages 403–414. IEEE, 2024.

[55] C. Guo, H. Li, and D. Pan. An improved piecewise aggregate approximation based on statistical features for time series mining. In *Knowledge Science, Engineering and Management: 4th International Conference, KSEM 2010, Belfast, Northern Ireland, UK, September 1-3, 2010. Proceedings 4*, pages 234–244. Springer, 2010.

[56] X. Guo, X. Peng, H. Wang, W. Li, H. Jiang, D. Ding, T. Xie, and L. Su. Graph-based trace analysis for microservice architecture understanding and problem diagnosis. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 1387–1397, 2020.

[57] Z. Guo, Y. Xu, Y.-F. Liu, S. Liu, H. J. Chao, Z.-L. Zhang, and Y. Xia. Aggreflow: Achieving power efficiency, load balancing, and quality of service in data center networks. *IEEE/ACM Transactions on Networking*, 29(1):17–33, 2020.

[58] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen. Logmine: Fast pattern recognition for log analytics. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 1573–1582, 2016.

[59] S. Han and S. S. Woo. Learning sparse latent graph representations for anomaly detection in multivariate time series.

In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2977–2986, 2022.

[60] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu. An evaluation study on log parsing and its use in log mining. In *2016 46th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 654–661. IEEE, 2016.

[61] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu. Towards automated log parsing for large-scale log data analysis. *IEEE Transactions on Dependable and Secure Computing*, 15(6):931–944, 2017.

[62] P. He, J. Zhu, Z. Zheng, and M. R. Lyu. Drain: An online log parsing approach with fixed depth tree. In *Proceedings of the IEEE International Conference on Web Services (ICWS)*, pages 33–40. IEEE, 2017.

[63] S. He, B. Feng, L. Li, X. Zhang, Y. Kang, Q. Lin, S. Rajmohan, and D. Zhang. Steam: Observability-preserving trace sampling. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1750–1761, 2023.

[64] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu. A survey on automated log analysis for reliability engineering. *ACM computing surveys (CSUR)*, 54(6):1–37, 2021.

[65] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, and D. Zhang. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 60–70, 2018.

[66] S. He, J. Zhu, P. He, and M. R. Lyu. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 207–218. IEEE, 2016.

[67] X. He, C. Shao, and Y. Xiong. A non-parametric symbolic approximate representation for long time series. *Pattern Analysis and Applications*, 19:111–127, 2016.

[68] Y. He, R. Guo, Y. Xing, X. Che, K. Sun, Z. Liu, K. Xu, and Q. Li. Cross container attacks: The bewildered {eBPF} on clouds. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5971–5988, 2023.

[69] Z. He, P. Chen, X. Li, Y. Wang, G. Yu, C. Chen, X. Li, and Z. Zheng. A spatiotemporal deep learning approach for unsupervised anomaly detection in cloud systems. *IEEE Transactions on Neural Networks and Learning Systems*, 34(4):1705–1719, 2020.

[70] P. Hennig and C. J. Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(6), 2012.

[71] J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. *Advances in neural information processing systems*, 27, 2014.

[72] M. W. Hoffman and Z. Ghahramani. Output-space predictive entropy search for flexible global optimization. In *NIPS workshop on Bayesian Optimization*, pages 1–5, 2015.

[73] T. Huang, P. Chen, and R. Li. A semi-supervised vae-based active anomaly detection framework in multivariate time series for online systems. In *Proceedings of the ACM Web Conference 2022*, pages 1797–1806, 2022.

[74] T. Huang, P. Chen, J. Zhang, R. Li, and R. Wang. A transferable time series forecasting service using deep transformer model for online systems. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1–12, 2022.

[75] Z. Huang, P. Chen, G. Yu, H. Chen, and Z. Zheng. Sieve: Attention-based sampling of end-to-end trace data in distributed microservice systems. In *2021 IEEE International Conference on Web Services (ICWS)*, pages 436–446. IEEE, 2021.

[76] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining (KDD)*, pages 387–395, 2018.

[77] Y. Huo, Y. Su, C. Lee, and M. R. Lyu. Semparser: A semantic parser for log analytics. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 881–893. IEEE, 2023.

[78] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth. Performance anomaly detection and bottleneck identification. *ACM Computing Surveys (CSUR)*, 48(1):1–35, 2015.

[79] O. Ibidunmoye, A.-R. Rezaie, and E. Elmroth. Adaptive anomaly detection in performance metric streams. *IEEE Transactions on Network and Service Management*, 15(1):217–231, 2017.

[80] M. N. Injadat, F. Salo, A. B. Nassif, A. Essex, and A. Shami. Bayesian optimization with machine learning algorithms towards anomaly detection. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2018.

[81] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning (ICML)*, pages 448–456. PMLR, 2015.

[82] M. S. Islam, W. Pourmajidi, L. Zhang, J. Steinbacher, T. Erwin, and A. Miranskyy. Anomaly detection in a large-scale cloud platform. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 150–159. IEEE, 2021.

[83] M. Jiang, M. A. Munawar, T. Reidemeister, and P. A. Ward. System monitoring with metric-correlation models: problems and solutions. In *Proceedings of the 6th international conference on Autonomic computing*, pages 13–22, 2009.

[84] Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, Y. Peng, X. Li, C. Xie, S. Nong, et al. Megascale: Scaling large language model training to more than 10,000 gpus. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 745–760, 2024.

[85] Z. Jiang, J. Liu, Z. Chen, Y. Li, J. Huang, Y. Huo, P. He, J. Gu, and M. R. Lyu. Lilac: Log parsing using llms with adaptive parsing cache. *Proceedings of the ACM on Software Engineering*, 1(FSE):137–160, 2024.

[86] Z. Jiang, J. Liu, J. Huang, Y. Li, Y. Huo, J. Gu, Z. Chen, J. Zhu, and M. R. Lyu. A large-scale evaluation for log parsing techniques: How far are we? In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 223–234, 2024.

[87] Z. M. Jiang, A. E. Hassan, P. Flora, and G. Hamann. Abstracting execution logs to execution events for enterprise applica-

tions (short paper). In *2008 The Eighth International Conference on Quality Software*, pages 181–186. IEEE, 2008.

[88] H. Jin, Z. Li, H. Liu, X. Liao, and Y. Zhang. Hotspot-aware hybrid memory management for in-memory key-value stores. *IEEE Transactions on Parallel and Distributed Systems*, 31(4):779–792, 2019.

[89] M. Jin, H. Y. Koh, Q. Wen, D. Zambon, C. Alippi, G. I. Webb, I. King, and S. Pan. A survey on graph neural networks for time series: Forecasting, classification, imputation, and anomaly detection. *arXiv preprint arXiv:2307.03759*, 2023.

[90] A. Kane and N. Shiri. Multivariate time series representation and similarity search using pca. In *Advances in Data Mining. Applications and Theoretical Aspects: 17th Industrial Conference, ICDM 2017*, pages 122–136. Springer, 2017.

[91] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[92] F. Karimzadeh, M. Imani, B. Asgari, N. Cao, Y. Lin, and Y. Fang. Memory-based computing for energy-efficient ai: Grand challenges. In *2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–8. IEEE, 2023.

[93] P. Kaushik, A. M. Rao, D. P. Singh, S. Vashisht, and S. Gupta. Cloud computing and comparison based on service and performance between amazon aws, microsoft azure, and google cloud. In *2021 International Conference on Technological Advancements and Innovations (ICTAI)*, pages 268–273. IEEE, 2021.

[94] Z. A. Khan, D. Shin, D. Bianculli, and L. Briand. Guidelines for assessing the accuracy of log message template identification techniques. In *Proceedings of the 44th International Conference on Software Engineering (ICSE)*, pages 1095–1106, 2022.

[95] K. Kim and M.-j. Park. Present and future, challenges of high bandwidth memory (hbm). In *2024 IEEE International Memory Workshop (IMW)*, pages 1–4. IEEE, 2024.

[96] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[97] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[98] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[99] R. Kiryo, G. Niu, M. C. Du Plessis, and M. Sugiyama. Positive-unlabeled learning with non-negative risk estimator. *Advances in neural information processing systems (NeurIPS)*, 30, 2017.

[100] J. Kuang, J. Liu, J. Huang, R. Zhong, J. Gu, L. Yu, R. Tan, Z. Yang, and M. R. Lyu. Knowledge-aware alert aggregation in large-scale cloud systems: a hybrid approach. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*, pages 369–380, 2024.

[101] M. Latah and L. Toker. Artificial intelligence enabled software-defined networking: a comprehensive overview. *IET networks*, 8(2):79–99, 2019.

[102] V.-H. Le and H. Zhang. Log parsing with prompt-based few-shot learning. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 2438–2449. IEEE, 2023.

[103] C. Lee, T. Yang, Z. Chen, Y. Su, and M. R. Lyu. Maat: Performance metric anomaly anticipation for cloud services with conditional diffusion. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 116–128. IEEE, 2023.

[104] J. Lee, I. Lee, and J. Kang. Self-attention graph pooling. In *International conference on machine learning (ICML)*, pages 3734–3743. PMLR, 2019.

[105] J. Lee, J. M. Lee, Y. Oh, W. J. Song, and W. W. Ro. Snakebyte: A tlb design with adaptive and recursive page merging in gpus. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1195–1207. IEEE, 2023.

[106] N. Lee, T. Ajanthan, and P. Torr. Snip: single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*. Open Review, 2019.

[107] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[108] L. Li, X. Zhang, S. He, Y. Kang, H. Zhang, M. Ma, Y. Dang, Z. Xu, S. Rajmohan, Q. Lin, et al. Conan: Diagnosing batch failures for cloud systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software En-*

gineering in Practice (ICSE-SEIP), pages 138–149. IEEE, 2023.

[109] X. Li, G. Yu, P. Chen, H. Chen, and Z. Chen. Going through the life cycle of faults in clouds: Guidelines on fault handling. In *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, pages 121–132. IEEE, 2022.

[110] Z. Li, C. Luo, T.-H. Chen, W. Shang, S. He, Q. Lin, and D. Zhang. Did we miss something important? studying and exploring variable-aware log abstraction. *arXiv preprint arXiv:2304.11391*, 2023.

[111] Z. Li, N. Zhao, M. Li, X. Lu, L. Wang, D. Chang, X. Nie, L. Cao, W. Zhang, K. Sui, et al. 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*, pages 996–1008, 2022.

[112] Z. Li, Y. Zhao, J. Han, Y. Su, R. Jiao, X. Wen, and D. Pei. Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 3220–3230, 2021.

[113] Z. Li, Y. Zhao, R. Liu, and D. Pei. Robust and rapid clustering of kpis for large-scale anomaly detection. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2018.

[114] D. Lin, R. Raghu, V. Ramamurthy, J. Yu, R. Radhakrishnan, and J. Fernandez. Unveiling clusters of events for alert and incident management in large-scale enterprise it. In *Proceedings of the 20th ACM SIGKDD international conference*

on Knowledge discovery and data mining, pages 1630–1639, 2014.

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

[116] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 2–11, 2003.

[117] Q. Lin, K. Hsieh, Y. Dang, H. Zhang, K. Sui, Y. Xu, J.-G. Lou, C. Li, Y. Wu, R. Yao, et al. Predicting node failure in cloud service systems. In *Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pages 480–490, 2018.

[118] Q. Lin, J.-G. Lou, H. Zhang, and D. Zhang. idice: problem identification for emerging issues. In *Proceedings of the 38th International Conference on Software Engineering*, pages 214–224, 2016.

[119] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 102–111, 2016.

[120] F. Liu, X. Zhou, J. Cao, Z. Wang, T. Wang, H. Wang, and Y. Zhang. Anomaly detection in quasi-periodic time series based on automatic data segmentation and attentional lstm-

cnn. *IEEE Transactions on Knowledge and Data Engineering*, 34(6):2626–2640, 2020.

[121] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining (ICDM)*, pages 413–422. IEEE, 2008.

[122] J. Liu, W. Gu, Z. Chen, Y. Li, Y. Su, and M. R. Lyu. Mtad: Tools and benchmarks for multivariate time series anomaly detection. *arXiv preprint arXiv:2401.06175*, 2024.

[123] J. Liu, S. He, Z. Chen, L. Li, Y. Kang, X. Zhang, P. He, H. Zhang, Q. Lin, Z. Xu, et al. Incident-aware duplicate ticket aggregation for cloud systems. *arXiv preprint arXiv:2302.09520*, 2023.

[124] J. Liu, J. Huang, Y. Huo, Z. Jiang, J. Gu, Z. Chen, C. Feng, M. Yan, and M. R. Lyu. Scalable and adaptive log-based anomaly detection with expert in the loop. *arXiv preprint arXiv:2306.05032*, 2023.

[125] J. Liu, Z. Jiang, J. Gu, J. Huang, Z. Chen, C. Feng, Z. Yang, Y. Yang, and M. R. Lyu. Prism: Revealing hidden functional clusters from massive instances in cloud systems. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 268–280. IEEE, 2023.

[126] J. Liu, S. Wang, A. Zhou, S. A. Kumar, F. Yang, and R. Buyya. Using proactive fault-tolerance approach to enhance cloud service reliability. *IEEE Transactions on Cloud Computing*, 6(4):1191–1202, 2016.

[127] J. Liu, T. Yang, Z. Chen, Y. Su, C. Feng, Z. Yang, and M. R. Lyu. Practical anomaly detection over multivariate monitoring metrics for online services. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pages 36–45. IEEE, 2023.

[128] X. Liu, J. Wen, Z. Chen, D. Li, J. Chen, Y. Liu, H. Wang, and X. Jin. Faaslight: General application-level cold-start latency optimization for function-as-a-service in serverless computing. *ACM Transactions on Software Engineering and Methodology*, 32(5):1–29, 2023.

[129] Y. Liu, X. Zhang, S. He, H. Zhang, L. Li, Y. Kang, Y. Xu, M. Ma, Q. Lin, Y. Dang, et al. Uniparser: A unified log parser for heterogeneous log data. In *Proceedings of the ACM Web Conference 2022 (WWW)*, pages 1893–1901, 2022.

[130] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li. Mining invariants from console logs for system problem detection. In *USENIX annual technical conference*, pages 1–14, 2010.

[131] R. Lu, E. Xu, Y. Zhang, F. Zhu, Z. Zhu, M. Wang, Z. Zhu, G. Xue, J. Shu, M. Li, et al. Perseus: A {Fail-Slow} detection framework for cloud storage systems. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*, pages 49–64, 2023.

[132] S. Lu, X. Wei, Y. Li, and L. Wang. Detecting anomaly in big data system logs using a convolutional neural network. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, pages 151–158. IEEE, 2018.

[133] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu. Characterizing microservice dependency and performance: Alibaba trace analysis. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, pages 412–426, 2021.

[134] M. Ma, W. Lin, D. Pan, and P. Wang. Servicerank: Root cause identification of anomaly in large-scale microservice architectures. *IEEE Transactions on Dependable and Secure Computing*, 19(5):3087–3100, 2021.

[135] M. Ma, Z. Yin, S. Zhang, S. Wang, C. Zheng, X. Jiang, H. Hu, C. Luo, Y. Li, N. Qiu, et al. Diagnosing root causes of intermittent slow queries in cloud databases. *Proceedings of the VLDB Endowment*, 13(8):1176–1189, 2020.

[136] M. Ma, S. Zhang, J. Chen, J. Xu, H. Li, Y. Lin, X. Nie, B. Zhou, Y. Wang, and D. Pei. {Jump-Starting} multivariate time series anomaly detection for online service systems. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 413–426, 2021.

[137] Z. Ma, A. R. Chen, D. J. Kim, T.-H. Chen, and S. Wang. Llmparser: An exploratory study on using large language models for log parsing. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13, 2024.

[138] Z. Ma, D. J. Kim, and T.-H. P. Chen. Librelog: Accurate and efficient unsupervised log parsing using open-source large language models. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pages 924–936, 2025.

[139] P.-J. Maenhaut, B. Volckaert, V. Ongenae, and F. De Turck. Resource management in a containerized cloud: Status and challenges. *Journal of Network and Systems Management*, 28:197–246, 2020.

[140] A. A. Makanju, A. N. Zincir-Heywood, and E. E. Milios. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD international conference on*

Knowledge discovery and data mining (KDD), pages 1255–1264, 2009.

[141] L. Mariani, C. Monni, M. Pezzé, O. Riganelli, and R. Xin. Localizing faults in cloud systems. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, pages 262–273. IEEE, 2018.

[142] L. Mariani, M. Pezzè, O. Riganelli, and R. Xin. Predicting failures in multi-tier distributed systems. *Journal of Systems and Software*, 161:110464, 2020.

[143] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley. Neural architecture search without training. In *International conference on machine learning*, pages 7588–7598. PMLR, 2021.

[144] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, et al. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *IJCAI*, volume 19, pages 4739–4745, 2019.

[145] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas. A search-based approach for accurate identification of log message formats. In *Proceedings of the 26th Conference on Program Comprehension*, pages 167–177, 2018.

[146] A. S. Milani and N. J. Navimipour. Load balancing mechanisms and techniques in the cloud environments: Systematic literature review and future trends. *Journal of Network and Computer Applications*, 71:86–98, 2016.

[147] D. Minnen, C. Isbell, I. Essa, and T. Starner. Detecting subdimensional motifs: An efficient algorithm for generalized multivariate pattern discovery. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 601–606. IEEE, 2007.

[148] M. Mizutani. Incremental mining of system log format. In *2013 IEEE International Conference on Services Computing*, pages 595–602. IEEE, 2013.

[149] Y. Moon, S. H. Shin, S. Jang, D. Won, and S. Kang. A novel prediction-based two-tiered ecc for mitigating swd errors in hbm. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2024.

[150] M. Nagappan and M. A. Vouk. Abstracting log lines to log event types for mining software system logs. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 114–117. IEEE, 2010.

[151] P. J. Nair, D. A. Roberts, and M. K. Qureshi. Citadel: Efficiently protecting stacked memory from tsv and large granularity failures. *ACM Transactions on Architecture and Code Optimization (TACO)*, 12(4):1–24, 2016.

[152] H. Nguyen, Y. Tan, and X. Gu. Pal: Propagation-aware anomaly localization for cloud hosted distributed applications. In *Managing Large-Scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*, SLAML '11. Association for Computing Machinery, 2011.

[153] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[154] R. P. Padhy. Big data processing with hadoop-mapreduce in cloud systems. *International Journal of Cloud Computing and Services Science*, 2(1):16, 2013.

[155] B. Panda, D. Srinivasan, H. Ke, K. Gupta, V. Khot, and H. S. Gunawi. {IASO}: A {Fail-Slow} detection and mitigation

framework for distributed storage services. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 47–62, 2019.

[156] J. Paparrizos, Y. Kang, P. Boniol, R. S. Tsay, T. Palpanas, and M. J. Franklin. Tsb-uad: an end-to-end benchmark suite for univariate time-series anomaly detection. *Proceedings of the VLDB Endowment*, 15(8):1697–1711, 2022.

[157] D. Park, Y. Hoshi, and C. C. Kemp. A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder. *IEEE Robotics and Automation Letters*, 3(3):1544–1551, 2018.

[158] M. Peiris, J. H. Hill, J. Thelin, S. Bykov, G. Kliot, and C. Konig. Pad: Performance anomaly detection in multi-server distributed systems. In *IEEE 7th International Conference on Cloud Computing*, pages 769–776. IEEE, 2014.

[159] V. Perrone, R. Jenatton, M. W. Seeger, and C. Archambeau. Scalable hyperparameter transfer learning. *Advances in neural information processing systems*, 31, 2018.

[160] A. Putina, M. Bahri, F. Salutari, and M. Sozio. Autoad: an automated framework for unsupervised anomaly detection. In *2022 IEEE 9th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10. IEEE, 2022.

[161] J. Qi, S. Huang, Z. Luan, S. Yang, C. Fung, H. Yang, D. Qian, J. Shang, Z. Xiao, and Z. Wu. Loggpt: Exploring chatgpt for log-based anomaly detection. In *2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pages 273–280. IEEE, 2023.

[162] L. Qian, Z. Luo, Y. Du, and L. Guo. Cloud computing: An overview. In *IEEE international conference on cloud computing*, pages 626–631. Springer, 2009.

[163] J. Qiu, Q. Du, K. Yin, S.-L. Zhang, and C. Qian. A causality mining and knowledge graph based method of root cause diagnosis for performance anomaly in cloud applications. *Applied Sciences*, 10(6):2166, 2020.

[164] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, and Q. Zhang. Time-series anomaly detection service at microsoft. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3009–3017, 2019.

[165] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning (ICML)*, pages 1278–1286. PMLR, 2014.

[166] D. Scheinert, A. Acker, L. Thamsen, M. K. Geldenhuys, and O. Kao. Learning dependencies in distributed cloud applications to identify and localize anomalies. In *2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*, pages 7–12. IEEE, 2021.

[167] S. Schmidl, P. Wenig, and T. Papenbrock. Anomaly detection in time series: a comprehensive evaluation. *Proceedings of the VLDB Endowment*, 15(9):1779–1797, 2022.

[168] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[169] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu. Dbscan revisited, revisited: why and how you should (still)

use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21, 2017.

[170] P. Senin and S. Malinchik. Sax-vsm: Interpretable time series classification using sax and vector space model. In *2013 IEEE 13th international conference on data mining*, pages 1175–1180. IEEE, 2013.

[171] H. Shan, Y. Chen, H. Liu, Y. Zhang, X. Xiao, X. He, M. Li, and W. Ding. ?-diagnosis: Unsupervised and real-time diagnosis of small-window long-tail latency in large-scale microservice platforms. In *The World Wide Web Conference (WWW)*, pages 3215–3222, 2019.

[172] L. K. Shar, A. Goknil, E. J. Husom, S. Sen, Y. N. Tun, and K. Kim. Autoconf: Automated configuration of unsupervised learning systems using metamorphic testing and bayesian optimization. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1326–1338. IEEE, 2023.

[173] Y. Sharma, D. Bhamare, N. Sastry, B. Javadi, and R. Buyya. Sla management in intent-driven service management systems: A taxonomy and future directions. *ACM Computing Surveys*, 2023.

[174] L. Shen, Z. Li, and J. Kwok. Timeseries anomaly detection using temporal hierarchical one-class network. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:13016–13026, 2020.

[175] Y. Shen, Y. Li, J. Zheng, W. Zhang, P. Yao, J. Li, S. Yang, J. Liu, and B. Cui. Proxybo: Accelerating neural architecture search via bayesian optimization with zero-cost proxies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 9792–9801, 2023.

[176] X. Shi, Z. Ke, Y. Zhou, H. Jin, L. Lu, X. Zhang, L. He, Z. Hu, and F. Wang. Deca: A garbage collection optimizer for in-memory data processing. *ACM Transactions on Computer Systems (TOCS)*, 36(1):1–47, 2019.

[177] K. Shima. Length matters: Clustering system log messages using length of words. *arXiv preprint arXiv:1611.03213*, 2016.

[178] A. Shojaie and E. B. Fox. Granger causality: A review and recent advances. *Annual Review of Statistics and Its Application*, 9:289–319, 2022.

[179] M. A. Siddiqui, A. Fern, T. G. Dietterich, R. Wright, A. Theriault, and D. W. Archer. Feedback-guided anomaly discovery via online optimization. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2200–2209, 2018.

[180] S. Singh, R. Batheri, and J. Dias. Predictive analytics: How to improve availability of manufacturing equipment in automotive firms. *IEEE Engineering Management Review*, 2023.

[181] J. Soldani and A. 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.

[182] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2828–2837, 2019.

[183] Y. Su, Y. Zhao, W. Xia, R. Liu, J. Bu, J. Zhu, Y. Cao, H. Li, C. Niu, Y. Zhang, et al. Coflux: robustly correlating kpis by fluctuations for service troubleshooting. In *Proceedings of the*

International Symposium on Quality of Service, pages 1–10, 2019.

[184] S. V. Subramanyam. Cloud-based enterprise systems: Bridging scalability and security in healthcare and finance. *IJSAT-International Journal on Science and Technology*, 16(1), 2025.

[185] M. B. Sullivan, N. Saxena, M. O'Connor, D. Lee, P. Racunas, S. Hukerikar, T. Tsai, S. K. S. Hari, and S. W. Keckler. Characterizing and mitigating soft errors in gpu dram. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 641–653, 2021.

[186] E. Sylligardos, P. Boniol, J. Paparrizos, P. Trahanias, and T. Palpanas. Choose wisely: An extensive evaluation of model selection for anomaly detection in time series. *Proceedings of the VLDB Endowment*, 16(11):3418–3432, 2023.

[187] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in neural information processing systems*, 33:6377–6389, 2020.

[188] L. Tang, T. Li, and C.-S. Perng. Logsig: Generating system events from raw textual logs. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 785–794, 2011.

[189] S. Tuli, G. Casale, and N. R. Jennings. Tranad: Deep transformer networks for anomaly detection in multivariate time series data. *arXiv preprint arXiv:2201.07284*, 2022.

[190] S. Tuli, S. S. Gill, P. Garraghan, R. Buyya, G. Casale, and N. Jennings. Start: Straggler prediction and mitigation for cloud computing environments using encoder lstm networks. *IEEE Transactions on Services Computing*, 2021.

[191] R. Vaarandi. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM)(IEEE Cat. No. 03EX764)*, pages 119–126. Ieee, 2003.

[192] R. Vaarandi and M. Pihelgas. Logcluster-a data clustering and pattern mining algorithm for event logs. In *2015 11th International conference on network and service management (CNSM)*, pages 1–7. IEEE, 2015.

[193] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems (NeurIPS)*, 30, 2017.

[194] A. Visheratin, A. Struckov, S. Yufa, A. Muratov, D. Nasonov, N. Butakov, Y. Kuznetsov, and M. May. Peregreen–modular database for efficient storage of historical time series in cloud environments. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 589–601, 2020.

[195] C. Wang, Z. Chen, and M. Zhou. Automl from software engineering perspective: Landscapes and challenges. In *Proceedings of the 20th International Conference on Mining Software Repositories*. MSR, 2023.

[196] D. Wang, Z. Chen, J. Ni, L. Tong, Z. Wang, Y. Fu, and H. Chen. Hierarchical graph neural networks for causal discovery and root cause localization. *arXiv preprint arXiv:2302.01987*, 2023.

[197] H. Wang, P. Nguyen, J. Li, S. Kopru, G. Zhang, S. Katariya, and S. Ben-Romdhane. Grano: Interactive graph-based root cause analysis for cloud-native distributed data platform. *Proceedings of the VLDB Endowment*, 12(12):1942–1945, 2019.

[198] H. Wang, Z. Wu, H. Jiang, Y. Huang, J. Wang, S. Kopru, and T. Xie. Groot: An event-graph-based approach for root cause analysis in industrial settings. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 419–429. IEEE, 2021.

[199] L. Wang, S. Chen, and Q. He. Concept drift-based runtime reliability anomaly detection for edge services adaptation. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[200] L. Wang, C. Zhang, R. Ding, Y. Xu, Q. Chen, W. Zou, Q. Chen, M. Zhang, X. Gao, H. Fan, et al. Root cause analysis for microservice systems via hierarchical reinforcement learning from human feedback. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5116–5125, 2023.

[201] W. Wang, P. Chen, Y. Xu, and Z. He. Active-mtsad: multivariate time series anomaly detection with active learning. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 263–274. IEEE, 2022.

[202] X. Wang, Y. Jin, S. Schmitt, and M. Olhofer. Recent advances in bayesian optimization. *ACM Computing Surveys*, 55(13s):1–36, 2023.

[203] X. Wang, X. Zhang, L. Li, S. He, H. Zhang, Y. Liu, L. Zheng, Y. Kang, Q. Lin, Y. Dang, et al. Spine: a scalable log parser with feedback guidance. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1198–1208, 2022.

[204] Y. Wang, G. Li, Z. Wang, Y. Kang, Y. Zhou, H. Zhang, F. Gao, J. Sun, L. Yang, P. Lee, et al. Fast outage analysis of large-

scale production clouds with service correlation mining. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 885–896. IEEE, 2021.

[205] Z. Wang and S. Jegelka. Max-value entropy search for efficient bayesian optimization. In *International Conference on Machine Learning*, pages 3627–3635. PMLR, 2017.

[206] J. S. Ward and A. Barker. Semantic based data collection for large scale cloud systems. In *Proceedings of the fifth international workshop on Data-Intensive Distributed Computing Date*, pages 13–22, 2012.

[207] C. Wen, H. Wang, Y. Li, S. Qin, Y. Liu, Z. Xu, H. Chen, X. Xie, G. Pu, and T. Liu. Memlock: Memory usage guided fuzzing. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 765–777, 2020.

[208] J. Weng, J. H. Wang, J. Yang, and Y. Yang. Root cause analysis of anomalies of multitier services in public clouds. *IEEE/ACM Transactions on Networking (TON)*, 26(4):1646–1659, 2018.

[209] B. Wickremasinghe, R. N. Calheiros, and R. Buyya. Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In *2010 24th IEEE international conference on advanced information networking and applications*, pages 446–452. IEEE, 2010.

[210] J. Wilson, F. Hutter, and M. Deisenroth. Maximizing acquisition functions for bayesian optimization. *Advances in neural information processing systems*, 31, 2018.

[211] R. Wu, S. Zhou, J. Lu, Z. Shen, Z. Xu, J. Shu, K. Yang, F. Lin, and Y. Zhang. Removing obstacles before breaking through the memory wall: A close look at hbm errors in the field. In

2024 USENIX Annual Technical Conference (USENIX ATC 24), pages 851–867, 2024.

[212] Y. Wu, Z. Yu, M. Wen, Q. Li, D. Zou, and H. Jin. Understanding the threats of upstream vulnerabilities to downstream projects in the maven ecosystem. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1046–1058. IEEE, 2023.

[213] Y. Xiong, Y. Jiang, Z. Yang, L. Qu, G. Zhao, S. Liu, D. Zhong, B. Pinzur, J. Zhang, Y. Wang, et al. {SuperBench}: Improving cloud {AI} infrastructure reliability with proactive validation. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 835–850, 2024.

[214] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[215] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng, et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference*, pages 187–196, 2018.

[216] J. Xu, R. Yang, Y. Huo, C. Zhang, and P. He. Divlog: Log parsing with prompt-enhanced in-context learning. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–12, 2024.

[217] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan. Large-scale system problem detection by mining console logs. In *Proceedings of SOSP*, volume 9, pages 1–17. Citeseer, 2009.

[218] S. Yan, C. Shan, W. Yang, B. Xu, D. Li, L. Qiu, J. Tong, and Q. Zhang. Cmmd: Cross-metric multi-dimensional root

cause analysis. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4310–4320, 2022.

[219] S. Yan, B. Tang, J. Luo, X. Fu, and X. Zhang. Unsupervised anomaly detection with variational auto-encoder and local outliers factor for kpis. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 476–483. IEEE, 2021.

[220] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang. Semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1448–1460. IEEE, 2021.

[221] T. Yang, J. Shen, Y. Su, X. Ling, Y. Yang, and M. R. Lyu. Aid: efficient prediction of aggregated intensity of dependency in large-scale cloud systems. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 653–665. IEEE, 2021.

[222] J. Yin, X. Zhao, Y. Tang, C. Zhi, Z. Chen, and Z. Wu. Cloudscout: A non-intrusive approach to service dependency discovery. *IEEE Transactions on Parallel and Distributed Systems*, 28(5):1271–1284, 2016.

[223] B. Yu, J. Yao, Q. Fu, Z. Zhong, H. Xie, Y. Wu, Y. Ma, and P. He. Deep learning or classical machine learning? an empirical study on log-based anomaly detection. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pages 1–13, 2024.

[224] G. Yu, P. Chen, Z. He, Q. Yan, Y. Luo, F. Li, and Z. Zheng. Changerca: Finding root causes from software changes in large online systems. *Proceedings of the ACM on Software Engineering*, 1(FSE):24–46, 2024.

[225] G. Yu, P. Chen, P. Li, T. Weng, H. Zheng, Y. Deng, and Z. Zheng. Logreducer: Identify and reduce log hotspots in kernel on the fly. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1763–1775. IEEE, 2023.

[226] G. Yu, P. Chen, Y. Li, H. Chen, X. Li, and Z. Zheng. Nezha: Interpretable fine-grained root causes analysis for microservices on multi-modal observability data. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 553–565, 2023.

[227] G. Yu, P. Chen, and Z. Zheng. Microscaler: Cost-effective scaling for microservice applications in the cloud with an online learning approach. *IEEE Transactions on Cloud Computing*, 10(2):1100–1116, 2020.

[228] Q. Yu, W. Zhang, J. Cardoso, and O. Kao. Exploring error bits for memory failure prediction: An in-depth correlative study. In *2023 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 01–09. IEEE, 2023.

[229] Q. Yu, W. Zhang, M. Zhou, J. Yu, Z. Sheng, J. Bogatinovski, J. Cardoso, and O. Kao. Investigating memory failure prediction across cpu architectures. In *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, pages 88–95. IEEE, 2024.

[230] Z. Yu, C. Pei, S. Zhang, X. Wen, J. Li, G. Xie, and D. Pei. Autokad: Empowering kpi anomaly detection with label-free deployment. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pages 13–23. IEEE, 2023.

[231] C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, and N. V. Chawla. A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 1409–1416, 2019.

[232] C. Zhang, H. Sun, S. Li, Y. Wang, H. Chen, and H. Liu. A survey of memory-centric energy-efficient computer architecture. *IEEE Transactions on Parallel and Distributed Systems*, 2023.

[233] J. Zhang, W. Gu, Y. Huang, Z. Jiang, W. Wu, and M. R. Lyu. Curvature-invariant adversarial attacks for 3d point clouds. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 7142–7150, 2024.

[234] J. Zhang, Y. Huang, W. Wu, and M. R. Lyu. Transferable adversarial attacks on vision transformers with token gradient regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16415–16424, 2023.

[235] J. Zhang, W. Wu, J.-t. Huang, Y. Huang, W. Wang, Y. Su, and M. R. Lyu. Improving adversarial transferability via neuron attribution-based attacks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14993–15002, 2022.

[236] P. Zhang, Y. Wang, X. Ma, Y. Xu, B. Yao, X. Zheng, and L. Jiang. Predicting dram-caused node unavailability in hyper-scale clouds. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 275–286. IEEE, 2022.

[237] S. Zhang, D. Li, Z. Zhong, J. Zhu, M. Liang, J. Luo, Y. Sun, Y. Su, S. Xia, Z. Hu, et al. Robust system instance clustering for large-scale web services. In *Proceedings of the ACM Web Conference 2022*, pages 1785–1796, 2022.

[238] X. Zhang, C. Du, Y. Li, Y. Xu, H. Zhang, S. Qin, Z. Li, Q. Lin, Y. Dang, A. Zhou, et al. Halo: Hierarchy-aware fault localization for cloud systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 3948–3958, 2021.

[239] X. Zhang, J. Kim, Q. Lin, K. Lim, S. O. Kanaujia, Y. Xu, K. Jamieson, A. Albarghouthi, S. Qin, M. J. Freedman, et al. Cross-dataset time series anomaly detection for cloud systems. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 1063–1076, 2019.

[240] X. Zhang, S. Shi, H. Sun, D. Chen, G. Wang, and K. Wu. Acvae: A novel self-adversarial variational auto-encoder combined with contrast learning for time series anomaly detection. *Neural Networks*, 171:383–395, 2024.

[241] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, et al. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pages 807–817, 2019.

[242] G. Zhao, S. Hassan, Y. Zou, D. Truong, and T. Corbin. Predicting performance anomalies in software systems at run-time. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(3):1–33, 2021.

[243] H. Zhao, Y. Wang, J. Duan, C. Huang, D. Cao, Y. Tong, B. Xu, J. Bai, J. Tong, and Q. Zhang. Multivariate time-series anomaly detection via graph attention network. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 841–850. IEEE, 2020.

[244] N. Zhao, J. Chen, X. Peng, H. Wang, X. Wu, Y. Zhang, Z. Chen, X. Zheng, X. Nie, G. Wang, et al. Understanding and handling alert storm for online service systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 162–171, 2020.

[245] N. Zhao, J. Chen, Z. Wang, X. Peng, G. Wang, Y. Wu, F. Zhou, Z. Feng, X. Nie, W. Zhang, et al. Real-time incident prediction for online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 315–326, 2020.

[246] N. Zhao, P. Jin, L. Wang, X. Yang, R. Liu, W. Zhang, K. Sui, and D. Pei. Automatically and adaptively identifying severe alerts for online service systems. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 2420–2429. IEEE, 2020.

[247] N. Zhao, H. Wang, Z. Li, X. Peng, G. Wang, Z. Pan, Y. Wu, Z. Feng, X. Wen, W. Zhang, et al. An empirical investigation of practical log anomaly detection for online service systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on*

the Foundations of Software Engineering, pages 1404–1415, 2021.

[248] N. Zhao, J. Zhu, R. Liu, D. Liu, M. Zhang, and D. Pei. Label-less: A semi-automatic labelling tool for kpi anomalies. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1882–1890. IEEE, 2019.

[249] Y. Zhao, Z. Nasrullah, and Z. Li. Pyod: A python toolbox for scalable outlier detection. *arXiv preprint arXiv:1901.01588*, 2019.

[250] Y. Zhao, R. Rossi, and L. Akoglu. Automatic unsupervised outlier model selection. *Advances in Neural Information Processing Systems*, 34:4489–4502, 2021.

[251] R. Zhong, Y. Li, J. Kuang, W. Gu, Y. Huo, and M. R. Lyu. Automated defects detection and fix in logging statement. *arXiv preprint arXiv:2408.03101*, 2024.

[252] F. Zhou, M. Goel, P. Desnoyers, and R. Sundaram. Scheduler vulnerabilities and coordinated attacks in cloud computing. *Journal of Computer Security*, 21(4):533–559, 2013.

[253] T. Zhou, C. Zhang, X. Peng, Z. Yan, P. Li, J. Liang, H. Zheng, W. Zheng, and Y. Deng. Tracestream: Anomalous service localization based on trace stream clustering with online feedback. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pages 601–611. IEEE, 2023.

[254] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 121–130. IEEE, 2019.

[255] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International conference on learning representations (ICLR)*, 2018.