Anomaly Detection and Diagnosis for Microservices-based IoT Applications in Edge Computing Environments

Duneesha Fernando

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

School of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE, AUSTRALIA

March 2026

ORCID: 0000-0003-4156-6695

Anomaly Detection and Diagnosis for Microservices-based IoT Applications in Edge Computing Environments

Duneesha Fernando
Principal Supervisor: Prof. Rajkumar Buyya
Co-Supervisor: Dr. Maria Rodriguez Read

Abstract

The rapid growth of the Internet of Things (IoT) has led to the deployment of large-scale, latency-sensitive applications across distributed edge–cloud computing environments. To meet diverse Quality of Service (QoS) requirements, modern IoT applications are increasingly designed using microservice architectures and dynamically placed across heterogeneous edge and cloud resources. While edge computing reduces communication latency and alleviates network congestion compared to cloud-centric models, it also introduces significant operational complexity due to device heterogeneity, limited resources, dynamic workloads, and multi-tenant execution. These characteristics make microservice-based IoT applications particularly susceptible to performance anomalies caused by resource contention, workload surges, deployment interference, and network disruptions. Moreover, performance anomalies can propagate through service communication and colocation dependencies, leading to cascading effects that obscure the true root cause. As a result, accurate anomaly detection must be complemented by efficient root cause localization and analysis to enable timely and effective mitigation. However, most existing anomaly detection and diagnosis techniques have been developed for centralized cloud environments and struggle to scale to distributed edge settings due to the lack of realistic evaluation data, high training overhead, centralized diagnosis assumptions, and computationally expensive graph-based reasoning.

This thesis adopts a diagnosis-centric perspective on performance anomaly management in microservice-based IoT applications deployed in edge computing environments. Its primary objective is to enable scalable, efficient, and accurate anomaly diagnosis—encompassing root cause localization and fault type analysis—under the heterogeneity, resource constraints, and large-scale distribution inherent to edge systems. Recognizing that effective diagnosis cannot be achieved in isolation, the thesis jointly

addresses realistic dataset generation, reproducible evaluation, and efficient anomaly detection model training as tightly coupled enablers of diagnosis. To overcome the limitations of existing centralized and cloud-oriented approaches, the proposed solutions explicitly exploit deployment structure, communication locality, and system hierarchy through clustering-based learning, decentralized graph analysis, and hierarchical graph neural network architectures. By decomposing global system representations into structured and localized components, the thesis reduces computational overhead, improves diagnostic efficiency, and preserves accuracy, even as system scale increases.

The key contributions of this thesis are summarized as follows:

- A comprehensive review and taxonomy of performance anomaly detection, root cause localization, and root cause analysis techniques for microservice-based edge–cloud systems, identifying key limitations in data availability, centralized assumptions, training efficiency, and scalability.
- The design and implementation of *iAnomaly*, a full-system emulation and automated dataset generation framework for edge computing environments, together with the public release of realistic performance anomaly datasets and a trace-driven simulator that enables scalable and reproducible evaluation.
- Scalable and efficient training strategies for anomaly detection models in heterogeneous edge environments, including intra-cluster parameter transfer learning and cluster-level model training, which balance detection accuracy with training and management overhead.
- A decentralized root cause localization framework that performs diagnosis directly at the edge device level using communication- and colocation-aware clustering and Personalized PageRank, significantly reducing diagnosis latency and communication overhead compared to centralized approaches.
- A cascaded graph neural network architecture for joint root cause localization and fault type analysis, which decomposes large dependency graphs into structured subgraphs via communication-aware clustering and enables hierarchical inference with near-constant latency at scale while preserving diagnostic accuracy.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Duneesha Fernando, March 2026

Preface

Main Contributions

This thesis research has been carried out in the Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne. The main contributions of the thesis are discussed in Chapters 2-6 and are based on the following publications:

- **Duneesha Fernando**, Maria A. Rodriguez, and Rajkumar Buyya, ”Performance Anomaly Detection and Diagnosis for Microservices-based IoT Applications in Edge Computing Environments: A Taxonomy and Future Directions”, *submitted to ACM Computing Surveys (CSUR), March 2026*.

- **Duneesha Fernando**, Maria A. Rodriguez, and Rajkumar Buyya, “iAnomaly: A Toolkit for Generating Performance Anomaly Datasets in Edge-Cloud Integrated Computing Environments”, *Proceedings of the 17th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2024)*, Pages: 236–245, Sharjah, United Arab Emirates, December 16-19, 2024.

- **Duneesha Fernando**, Maria A. Rodriguez, Patricia Arroba, Leila Ismail, and Rajkumar Buyya, “Efficient Training Approaches for Performance Anomaly Detection Models in Edge Computing Environments”, *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, Volume 20, Pages: 1-27, ISSN: 1556-4665, Association for Computing Machinery, New York, NY, USA, June 2025.

- **Duneesha Fernando**, Maria A. Rodriguez, and Rajkumar Buyya, “A Decentralized Root Cause Localization Approach for Edge Computing Environments”, *IEEE Transactions on Services Computing (TSC)*, pp. 1–14, February 2026, doi: 10.1109/TSC.2026.3668228.

- **Duneesha Fernando**, Maria A. Rodriguez, and Rajkumar Buyya, ”A Cascaded Graph Neural Network for Joint Root Cause Localization and Analysis in Edge Computing Environments”, *submitted to IEEE Transactions on Services Computing (TSC), March 2026* (Under Review, available on arXiv:2603.01447).

Acknowledgements

Undertaking a PhD is a long and challenging journey that would not have been possible without the support, encouragement, and guidance of many individuals. I am deeply grateful to everyone who has contributed to this journey.

First and foremost, I would like to express my sincere gratitude to my primary supervisor, Professor Rajkumar Buyya, for giving me the opportunity to pursue my PhD under his supervision. His guidance, encouragement, and unwavering support throughout my doctoral studies have been invaluable. I am particularly grateful for the intellectual freedom he provided while conducting my research and for the inspiration drawn from his vast experience and positive outlook toward both research and life.

I would also like to thank my co-supervisor, Dr. Maria Read, for her constructive feedback, insightful discussions, and generous dedication of time to carefully read and refine my work. Her thoughtful guidance has significantly strengthened my research and helped me grow in confidence as an independent researcher.

My sincere thanks also go to my advisory committee chair, Associate Professor Jorge Goncalves, for his support throughout my candidature and for ensuring the timely progress of my PhD. I greatly appreciate his understanding and empathetic guidance.

I am grateful to the members of the Quantum Cloud Computing and Distributed Systems (qCLOUDS) Laboratory, both past and present, for creating such a collaborative and supportive research environment. In particular, I would like to thank Dr. Mohammad Goudarzi, Dr. Samodha Pallewatta, and Dr. Anupama Mampage, who served as mentors during my early research journey and generously shared their knowledge and experience. I also thank Dr. Amanda Jayanetti, Dr. Tharindu B. Hewage, Dr. Thanh Hoa Nguyen, Dr. Siddharth Agarwal, Dr. Ming Chen, Dr. Zhiyu Wang, Jie Zhao, Thakshila Imiya Mohottige, Qifan Deng, TianYu Qi, Hootan Zhian, Murtaza Rangwala, Prabhjot Singh, Haoyu Bai, Abhishek Sawaika, and Avishka Sandeepa for their support, collaboration, and the many valuable discussions we shared.

I gratefully acknowledge The University of Melbourne for providing scholarships and travel grants that supported my studies and international conference travel. I also thank the ARC-funded Discovery Project that supported aspects of this research, as well as the administrative staff of the School of Computing and Information Systems for their assistance throughout my candidature.

I would also like to thank my friends in Melbourne who made this journey more meaningful and enjoyable. In particular, I am grateful to Hiruni and Charith, Shavindra, and Maneesha and Sameera, whom I have known since Sri Lanka, for their constant encouragement, friendship, and for being a source of support throughout my time in

Melbourne. I am also thankful to my extended CLOUDS lab family, including Tharindu and Nilakshi, Hoa, Ming, and Siddharth and Poorvi, for the friendships that grew beyond research and the lab environment. Their companionship, support, and shared moments made life in Melbourne much more enjoyable. A special mention goes to Siddharth and Poorvi's baby, Harvin, born in 2024, whose bright smiles and joyful presence brought moments of warmth and happiness during this journey, reminding us all that even amidst the demands of research, there is always space for simple and beautiful moments of joy.

I would also like to acknowledge my research mentors in Sri Lanka, Dr. Nalin Ranasinghe, Dr. Srinath Perera, and Dr. Malith Jayasinghe, who encouraged and prepared me for doctoral studies. Their guidance during my early academic journey played an important role in shaping my research aspirations.

My deepest gratitude goes to my parents. My father, Migara Fernando, instilled in me the importance of education, curiosity, and self-learning from a very young age, laying a strong foundation for my academic journey. He always believed in my potential and encouraged me to pursue my dreams, including undertaking a PhD abroad. My mother, Samanmalie Fernando, has always supported me with unconditional love, constant encouragement, and daily support throughout this journey. Even though it was difficult for her to see me move far away from home, she always supported my decision and stood by me every step of the way.

I am also thankful to my in-laws and extended family in Sri Lanka for their encouragement and support. I especially appreciate how they regularly checked in on me throughout this journey and for their deep appreciation of the value of education and research.

Finally, my heartfelt thanks go to my husband, Madhushan Buwaneswaran, for his unwavering love, support, and understanding throughout this journey. Pursuing a PhD required many sacrifices, including maintaining a long-distance relationship for four long years, even through the first year of our marriage. Despite the time differences and the challenges of distance, he was always there for me, listening, encouraging, and giving me strength during both the difficult moments and the celebrations along the way. His patience, belief in me, and constant reassurance helped me navigate the many ups and downs of this journey. This achievement would not have been possible without his love and support.

Looking back on this journey, I am deeply grateful for the guidance, encouragement, and kindness I have received from so many people along the way. Each of you has played a meaningful role in shaping this experience and supporting me through its many challenges and milestones. I carry forward not only the knowledge gained during this PhD, but also the friendships, memories, and lessons that will stay with me for a lifetime.

Duneesha Fernando
March 2026, Melbourne, Australia

To my mentors, friends and family, whose unwavering support and encouragement made this journey possible

Contents

List of Figures

List of Tables

List of Acronyms

AD	Anomaly Detection
AE	Autoencoder
AIOps	Artificial Intelligence for IT Operations
API	Application Programming Interface
BIRCH	Balanced Iterative Reducing and Clustering using Hierarchies
CM	Cluster-level Model Training
CNN	Convolutional Neural Network
DDoS	Distributed Denial-of-Service
DL	Deep Learning
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GM	Generic Model
GNN	Graph Neural Network
GRU	Gated Recurrent Unit
HPC	High Performance Computing
ICPTL	Intra-Cluster Parameter Transfer Learning
IF	Isolation Forest
IoT	Internet of Things
KPI	Key Performance Indicator

LSTM Long Short-Term Memory

ML Machine Learning

MPD Model per Device

P2P Peer-to-Peer

PPR Personalized PageRank

QML Quantum Machine Learning

QoS Quality of Service

QPS Queries Per Second

RCA Root Cause Analysis

RCL Root Cause Localization

SLA Service Level Agreement

SRE Site Reliability Engineering

VM Virtual Machine

Chapter 1
Introduction

The emergence of the Internet of Things (IoT) paradigm, supporting a wide range of smart services across domains such as healthcare, transportation, industrial automation, and agriculture, has led to an exponential growth in the number of connected devices worldwide. According to industry forecasts, the number of connected IoT devices surpassed 20 billion in 2025 and is projected to reach around 50 billion by 2035, reflecting sustained explosive growth in IoT adoption across consumer and industrial domains [1]. Initially, the massive volumes of data generated by IoT devices were transmitted to centralized cloud data centers for processing, giving rise to cloud-centric IoT architectures. However, this centralized model results in increased network congestion and high communication latency, particularly for latency-sensitive applications.

To overcome these limitations, edge computing was introduced, enabling data processing and analytics to be performed closer to data sources. Edge platforms consist of heterogeneous devices, including smart routers, gateways, edge servers, and micro-datacenters, with diverse computing, storage, and networking capabilities. Compared to IoT devices, edge nodes provide substantially higher computational and storage resources, although their capacities remain lower than those of centralized cloud servers [2]. Moreover, edge infrastructures are typically organized hierarchically, such that devices closer to data sources possess fewer computational resources than higher-tier edge nodes and cloud servers [3, 4].

As a result of this hierarchical organization, IoT applications are increasingly deployed across both edge and cloud resources based on their Quality of Service (QoS) requirements and computational demands. In this thesis, such integrated deployments are referred to as edge computing environments. This architectural shift is also reflected in industry adoption trends, with the global edge computing market projected to reach USD 327.79 billion by 2033 [5].

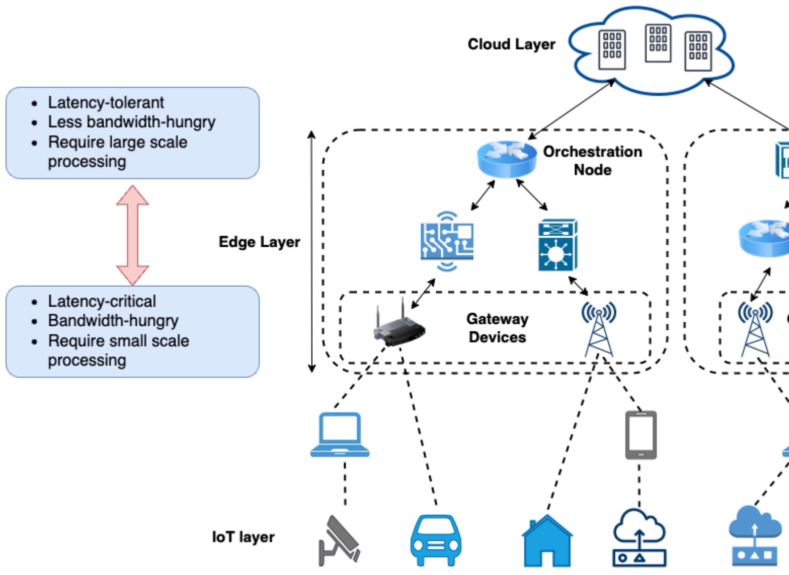


Figure 1.1: Properties of applications placed in edge computing environments

As illustrated in Figure 1.1, latency-critical and bandwidth-intensive components are typically placed close to data sources, while computation-intensive and delay-tolerant components are offloaded to higher-tier edge nodes or cloud data centers [2, 6].

In order to be deployed in distributed and heterogeneous edge computing environments, IoT applications are modeled as interdependent, lightweight, and granular modules. Microservice architectures that build distributed applications based on loosely coupled, lightweight, independently deployable, and scalable modules are increasingly being used for this purpose. Several studies have utilized the abstraction provided by microservice architectures to model IoT applications [7, 8] and have developed algorithms to schedule those containerized microservice-based IoT applications in edge computing environments [9, 10]. This granular placement ensures that the QoS requirements of each module can be satisfied while maximizing resource usage.

However, long-running microservice-based IoT applications, such as smart health monitoring and smart road traffic management, deployed on edge platforms may be prone to a degradation in performance, commonly referred to as performance anoma-

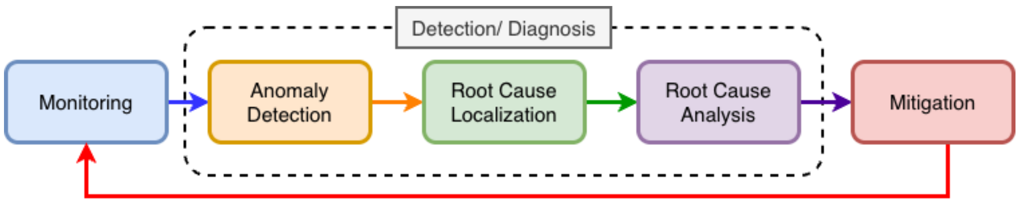


Figure 1.2: Anomaly management lifecycle

lies, mainly due to the environment's highly dynamic and multi-tenant nature [11, 12]. For instance, an upsurge in workload may result in an application hogging a resource (e.g., CPU or memory), or the co-location of microservices may result in resource contention, both cases potentially negatively impacting the performance of a given application. Furthermore, several types of malicious attacks, such as Distributed Denial-of-Service (DDoS) attacks, also indirectly cause performance problems [13]. These performance anomalies occurring in edge computing environments adversely affect the QoS of microservice-based IoT applications and lead to violations of their Service Level Agreement (SLA)s. In this thesis, performance anomalies are defined as deviations from expected behavioral patterns in QoS- and resource-related metrics, including abnormal outliers.

Moreover, anomalies can propagate through communication and colocation dependencies in microservice architectures [14, 15]. In other words, an anomaly originating in one microservice can cascade to others: for example, if a data aggregation microservice running on an edge node suffers from memory saturation, it may delay the delivery of processed sensor data. This slowdown can then cause dependent services, such as real-time anomaly detectors or actuator controllers, to experience increased latency or dropped requests, which in turn may appear anomalous even though they are not the true source. As a result, the system may encounter multiple detections across the architecture, even though only a single source is responsible for triggering the anomaly.

Consequently, continuous monitoring and timely diagnosis of microservice-based IoT applications in edge environments are essential for enabling effective mitigation. From an operational perspective, managing performance anomalies typically follows an anomaly management lifecycle consisting of monitoring, Anomaly Detection (AD), Root Cause Localization (RCL), Root Cause Analysis (RCA), and mitigation, as illustrated in

Figure 1.2. Monitoring continuously collects metrics, traces, and logs from distributed system components, while anomaly detection identifies deviations from normal behavior. Once an anomaly is detected, diagnosis becomes critical: RCL aims to identify the specific microservice responsible for the observed anomalous behavior, whereas RCA seeks to determine why the anomaly occurred, such as whether it was caused by CPU saturation, memory pressure, or network degradation. Accurate and timely diagnosis is a prerequisite for effective mitigation, as remediation actions taken without precise localization and analysis may be ineffective or even harmful in complex, distributed edge environments. Therefore, this thesis focuses on the diagnosis stages of the anomaly management lifecycle.

While substantial progress has been made in developing techniques for performance anomaly detection and diagnosis in cloud-based and microservice-oriented systems, the characteristics of edge computing environments introduce additional complexity. The high degree of heterogeneity, resource constraints, large-scale distribution, and dynamic deployment patterns fundamentally challenge existing centralized and resource-intensive approaches. Moreover, the close coupling between anomaly detection, localization, and analysis stages necessitates holistic and scalable solutions that can operate effectively across the diagnosis phases of the anomaly management lifecycle. These factors give rise to several key challenges that must be addressed to enable accurate, efficient, and automated performance management in edge computing environments.

1.1 Challenges in Edge-based Anomaly Detection and Diagnosis

Despite growing interest in performance anomaly detection and diagnosis for microservice-based systems, most existing solutions have been developed primarily for centralized cloud environments. Directly applying these techniques to edge computing settings is often ineffective due to fundamental differences in system architecture, resource availability, deployment dynamics, and operational constraints. In particular, the distributed and heterogeneous nature of edge environments introduces challenges across all stages of the anomaly management lifecycle, including data collection and evaluation, model training, RCL, and joint diagnosis. This thesis addresses four key challenges that hinder the development of scalable and reliable anomaly detection and diagnosis solutions for edge computing environments.

- *Lack of Realistic and Reproducible Evaluation Data*: The development and evaluation of anomaly detection and diagnosis techniques critically depend on the availability of high-quality datasets that capture realistic system behavior and diverse failure scenarios. However, publicly available datasets for edge–cloud microservice environments remain scarce [16, 17]. Existing benchmarks are often limited to cloud-only deployments, small-scale testbeds, or synthetic workloads that fail to reflect the heterogeneity, deployment hierarchies, and complex dependency structures present in real edge systems [11, 12, 18]. Furthermore, many experimental studies rely on proprietary datasets that cannot be shared, hindering reproducibility and fair comparison among competing approaches. The absence of realistic, reproducible, and scalable datasets makes it difficult to systematically evaluate detection accuracy, localization effectiveness, and efficiency under varying system conditions, thereby impeding scientific progress in this domain.

- *Scalable and Efficient Model Training in Heterogeneous Edge Environments*: Anomaly detection models deployed in edge environments must operate across a large number of devices with diverse hardware capabilities, workload characteristics, and performance profiles. Training separate models for each device or microservice can achieve high detection accuracy but incurs substantial computational, storage, and management overhead [11, 16, 19, 20]. Conversely, training a single generic model for all edge devices reduces operational cost but often fails to capture local performance patterns, leading to degraded accuracy [12, 21, 22]. Striking an appropriate balance between efficiency and effectiveness is particularly challenging in resource-constrained and dynamically evolving edge infrastructures. Moreover, frequent changes in workloads and deployment configurations further complicate model maintenance and retraining. These factors necessitate training strategies that can adapt to heterogeneity while minimizing resource consumption and operational complexity.

- *Efficient Root Cause Localization in Distributed Edge Systems*: Most existing RCL approaches assume centralized access to system-wide monitoring data and perform diagnosis on global dependency graphs [23–25]. While such designs are feasible in cloud environments, they incur excessive communication overhead and inference latency in distributed edge settings. Collecting and aggregating large volumes of metrics, traces, and logs from geographically dispersed devices introduces significant network traffic and delays, which can hinder timely mitigation. In addition, centralized analysis becomes a scalability bottleneck as system size grows [14, 26, 27]. At the same time, purely local diagnosis risks overlooking cross-device anomaly propagation and inter-service dependencies. Therefore, there is a fundamental challenge in designing localization mechanisms that can operate close to edge devices, reduce reliance on centralized processing, and still preserve sufficient global context to accurately identify root causes.

- *Scalable Joint Root Cause Localization and Analysis Using Graph Neural Network (GNN)s*: Recent advances in supervised GNNs have demonstrated strong performance for joint RCL and fault type identification by modeling complex inter-service dependencies [28–30]. However, these approaches typically rely on centralized processing over full-system graphs and involve computationally expensive message-passing operations. As the number of microservices and devices increases, the size of the resulting graphs grows rapidly, leading to prohibitively high inference latency and memory consumption [14, 24]. This limits the applicability of state-of-the-art GNN-based diagnosis techniques in large-scale edge deployments. Furthermore, naive graph partitioning or model simplification can result in information loss and degraded diagnostic accuracy. Consequently, there is a critical need for architectures that can decompose large dependency graphs into structured components and enable hierarchical reasoning, thereby achieving scalability without sacrificing diagnostic effectiveness.

Collectively, these challenges highlight the need for an integrated research effort that addresses data availability, learning efficiency, diagnostic scalability, and system-level coordination in edge computing environments. Overcoming these limitations requires not only algorithmic advances but also holistic system designs that account for the structural and operational characteristics of distributed edge–cloud infrastructures. Motivated by these challenges, this thesis formulates a set of research objectives and research questions aimed at guiding the systematic development of scalable and effective performance anomaly detection and diagnosis techniques.

1.2 Research Questions and Objectives

The primary objective of this thesis is to develop scalable, efficient, and reliable techniques for performance anomaly detection and diagnosis in microservice-based edge computing environments. In particular, this work focuses on the diagnosis phases of the anomaly management lifecycle, namely anomaly detection, root cause localization (RCL), and root cause analysis (RCA). In addition, the thesis addresses the challenge of realistic data generation and evaluation, which underpins the systematic development and assessment of diagnostic techniques in large-scale edge environments. By exploiting the structural, operational, and deployment characteristics of cloud–edge integrated systems, the thesis aims to improve diagnostic efficiency while preserving high accuracy and enabling reproducible evaluation under realistic and large-scale conditions.

To achieve this objective, the thesis is guided by the following overarching research question:

How can performance anomalies in large-scale, microservice-based edge–cloud environments be detected and diagnosed accurately and efficiently, despite data scarcity, heterogeneity, and scalability constraints?

This overarching question is addressed through four specific research questions, each corresponding to a key aspect of the diagnosis phases of the anomaly management lifecycle.

- *RQ1. How can realistic, reproducible, and large-scale performance anomaly datasets be generated for edge–cloud microservice environments in the absence of publicly available benchmarks?*

  The first research question addresses the fundamental challenge of evaluation and reproducibility in edge-based anomaly management. Existing studies often rely on small-scale testbeds, synthetic workloads, or proprietary datasets, which limit the generalizability and repeatability of reported results [11, 12, 18]. This question investigates how full-system emulation, automated monitoring, and controlled anomaly injection can be combined to generate realistic datasets that capture heterogeneous deployments, complex dependency structures, and diverse failure scenarios. In addition, it explores how trace-driven simulation techniques can be used to extend these datasets to large scales, thereby enabling systematic evaluation of scalability and efficiency.

- *RQ2. How can anomaly detection models be trained efficiently in heterogeneous edge environments without significantly compromising detection accuracy?*

  The second research question focuses on the scalability of model training in resource-constrained and highly diverse edge infrastructures. Training separate models for individual edge devices offers high accuracy but incurs prohibitive computational and management costs [11, 16, 19, 20], whereas generic models often fail to capture local performance characteristics [12, 21, 22]. This question examines how clustering-based learning strategies and knowledge transfer mechanisms can be leveraged to balance training efficiency and detection effectiveness. It further considers how such approaches can adapt to changing workloads and deployment configurations while minimizing retraining overhead.

- *RQ3. How can root causes of performance anomalies be localized efficiently in edge environments without relying on centralized analysis?*

  The third research question addresses the limitations of centralized diagnosis in distributed edge environments. Centralized localization approaches suffer from high communication overhead and increasing latency as system scale grows [14, 26, 27], while purely local methods risk overlooking global anomaly propagation patterns. This question investigates how communication- and colocation-aware clustering, decentralized graph analysis, and lightweight coordination mechanisms can be employed to perform RCL close to edge devices. The objective is to reduce reliance on global data aggregation while preserving sufficient contextual information for accurate diagnosis.

- *RQ4. How can supervised GNN-based joint root cause localization and fault type analysis be scaled to large edge deployments while maintaining diagnostic accuracy?*

  The fourth research question examines the scalability of state-of-the-art GNN–based diagnosis techniques in large edge systems. While supervised GNNs can effectively model inter-service dependencies and anomaly propagation, their reliance on full-system graphs leads to high computational overhead and limited scalability [14, 24]. This question explores how hierarchical and cascaded learning architectures, combined with communication-driven clustering, can decompose large dependency graphs into structured components. By enabling multi-stage and distributed inference, this research seeks to achieve near-constant diagnostic latency at scale without sacrificing localization and classification performance.

1.3 Thesis Contributions

This thesis makes the following contributions to address the research problems mentioned above:

1. Presents a comprehensive review and taxonomy of performance anomaly detection and diagnosis techniques for microservice-based edge–cloud systems, establishing the research foundation for this thesis.

   - A comprehensive review of state-of-the-art approaches to performance anomaly detection, root cause localization (RCL), and root cause analysis (RCA) in cloud–edge environments.
   - A structured taxonomy that classifies existing techniques according to diagnostic scope, observability modalities, learning paradigms, modeling techniques, dependency modeling strategies, reasoning mechanisms, and system architectures (centralized vs. decentralized), while considering scalability and efficiency aspects.

- A critical analysis of key limitations in existing approaches, including data scarcity, centralized assumptions, training overhead, and scalability challenges in edge environments.
- Identification of open research problems for scalable and automated anomaly management in microservice-based edge computing systems.

2. Develops a realistic and reproducible dataset generation framework for evaluating performance anomaly detection and diagnosis techniques in edge computing environments *(addresses RQ1)*.

   - A full-system emulation and automated dataset generation framework *iAnomaly*, that integrates workload generation, system monitoring, and controlled anomaly injection to capture realistic execution behavior in cloud–edge deployments.
   - An automated dataset generation and orchestration mechanism that coordinates emulation, anomaly injection, data collection, and dataset integration, enabling systematic generation of normal and anomalous traces under diverse deployment and fault scenarios.
   - A comprehensive public performance anomaly dataset derived from real executions of diverse microservice-based IoT applications, covering heterogeneous resource, workload, and network failure scenarios in cloud–edge environments.
   - A trace-driven dataset generation simulator, influenced by the *iAnomaly* emulator, that enables scalable synthesis of large service dependency graphs while preserving realistic temporal dynamics and anomaly propagation characteristics.
   - Public release of implementation artifacts, configuration files, and preprocessing scripts to support reproducible research and fair comparison.

3. Proposes scalable and efficient training strategies for anomaly detection models in heterogeneous edge environments *(addresses RQ2)*.

   - A similarity-based device clustering mechanism that groups edge devices based on the similarity of their normal performance data distributions.

- An Intra-Cluster Parameter Transfer Learning (ICPTL) strategy that accelerates model training by reusing knowledge across similar devices within a cluster.
- A Cluster-level Model Training (CM) approach that reduces the number of maintained models while preserving detection accuracy.
- An integrated training framework that balances computational efficiency, management overhead, and detection effectiveness.
- Extensive experimental validation on both public and emulated datasets demonstrating improved scalability compared to per-device and generic training baselines.

4. Introduces a decentralized root cause localization framework for efficient diagnosis in distributed edge environments *(addresses RQ3)*.
    - A communication- and colocation-aware clustering method that partitions large microservice systems into structurally coherent regions.
    - A decentralized Personalized PageRank (PPR)-based localization algorithm that performs diagnosis directly at the edge-device level.
    - A novel anomaly scoring mechanism tailored to the characteristics of edge infrastructures, capturing anomaly triggers across microservice, device, and network layers to improve localization accuracy.
    - A lightweight Peer-to-Peer (P2P) approximation mechanism that enables coordinated reasoning in multi-cluster anomaly propagation scenarios.
    - Comprehensive evaluation on real datasets as well as large-scale iAnomaly-generated datasets demonstrating reduced diagnosis latency while maintaining localization accuracy.

5. Develops a scalable GNN framework for joint root cause localization and fault type analysis in large edge deployments *(addresses RQ4)*.
    - A cascaded GNN architecture comprising a proposal network operating on communication-aware clusters to generate cluster-level outputs, and an out-

put network that models clusters as nodes in an inter-cluster graph to produce graph-level predictions.

- A communication-driven clustering strategy that preserves critical dependency structures while reducing graph complexity.
- A hierarchical inference mechanism that enables efficient message passing over structured subgraphs and compact cluster representations.
- A joint optimization framework that supports simultaneous learning of localization and fault classification tasks.
- Extensive evaluation on MicroCERCL and large-scale iAnomaly-generated datasets demonstrating near-constant inference latency with accuracy comparable to centralized GNN baselines.

1.4 Thesis Organisation

The structure of this thesis is shown in Figure 1.3. The remaining chapters of this thesis are organised as follows:

- Chapter 2 presents a systematic review and taxonomy of performance anomaly detection and diagnosis techniques for microservice-based edge–cloud systems. The chapter surveys state-of-the-art approaches across anomaly detection, RCL, and RCA, and classifies existing work according to diagnostic scope, observability sources, learning paradigms, modeling approaches, and system architectures, while also considering scalability constraints in edge environments. By critically analyzing current limitations and identifying open research challenges, this chapter establishes the conceptual foundation and motivates the technical contributions developed in the subsequent chapters. This chapter is derived from:

  Duneesha Fernando, Maria A. Rodriguez, and Rajkumar Buyya, "Performance Anomaly Detection and Diagnosis for Microservices-based IoT Applications in Edge Computing Environments: A Taxonomy and Future Directions", *submitted to ACM Computing Surveys (CSUR), March 2026.*

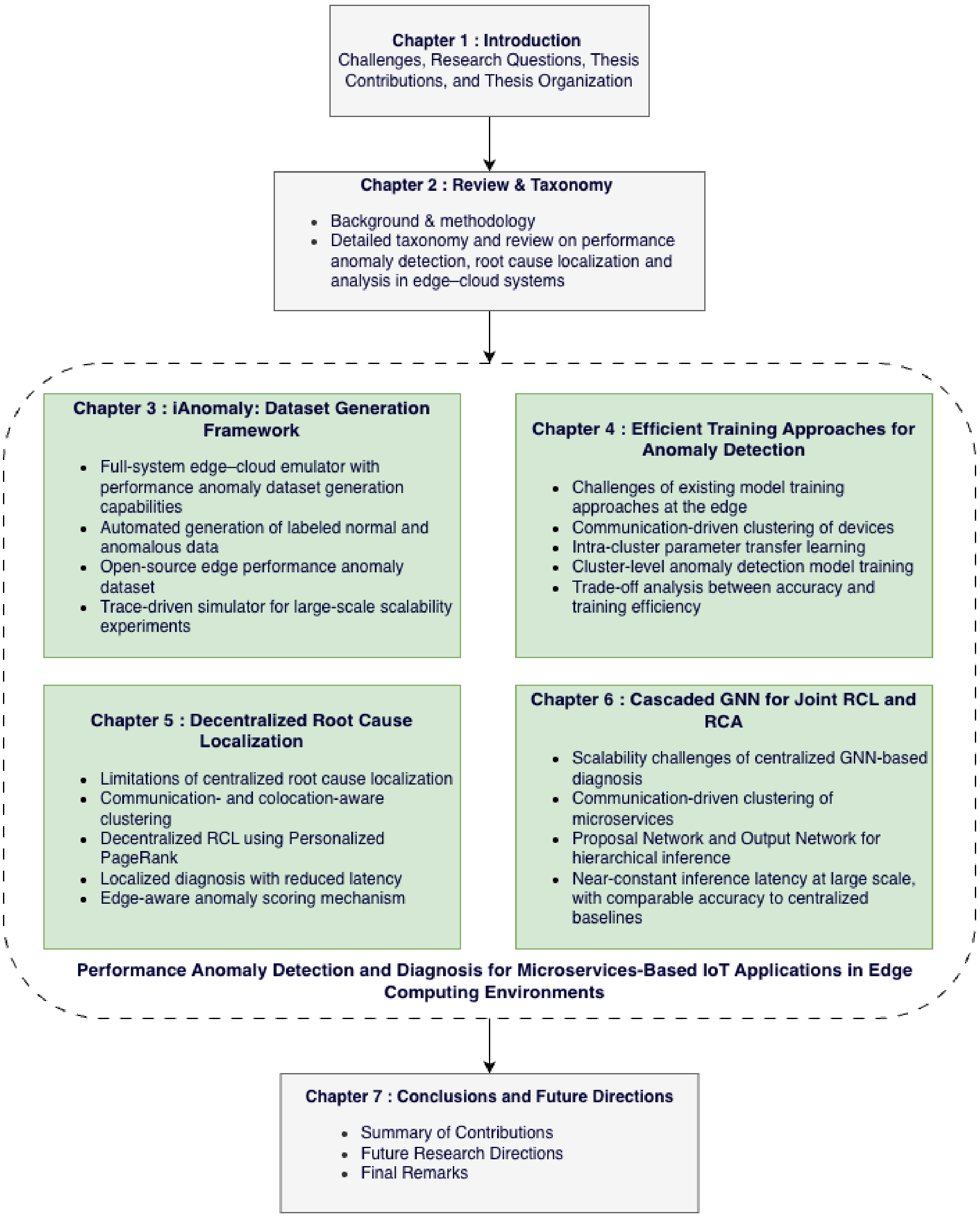

Chapter 1 : Introduction
Challenges, Research Questions, Thesis Contributions, and Thesis Organization
Chapter 2 : Review & Taxonomy
Background & methodology
Detailed taxonomy and review on performance anomaly detection, root cause localization and analysis in edge–cloud systems
Chapter 3 : iAnomaly: Dataset Generation Framework
Full-system edge–cloud emulator with performance anomaly dataset generation capabilities
Automated generation of labeled normal and anomalous data
Open-source edge performance anomaly dataset
Trace-driven simulator for large-scale scalability experiments
Chapter 4 : Efficient Training Approaches for Anomaly Detection
Challenges of existing model training approaches at the edge
Communication-driven clustering of devices
Intra-cluster parameter transfer learning
Cluster-level anomaly detection model training
Trade-off analysis between accuracy and training efficiency
Chapter 5 : Decentralized Root Cause Localization
Limitations of centralized root cause localization
Communication- and colocation-aware clustering
Decentralized RCL using Personalized PageRank
Localized diagnosis with reduced latency
Edge-aware anomaly scoring mechanism
Chapter 6 : Cascaded GNN for Joint RCL and RCA
Scalability challenges of centralized GNN-based diagnosis
Communication-driven clustering of microservices
Proposal Network and Output Network for hierarchical inference
Near-constant inference latency at large scale, with comparable accuracy to centralized baselines
Performance Anomaly Detection and Diagnosis for Microservices-Based IoT Applications in Edge Computing Environments
Chapter 7 : Conclusions and Future Directions
Summary of Contributions
Future Research Directions
Final Remarks

Figure 1.3: Overview of the thesis structure

- Chapter 3 addresses the lack of realistic and reproducible evaluation data for performance anomaly detection and diagnosis in edge computing environments. It introduces *iAnomaly*, a full-system emulation and automated dataset generation framework for microservice-based IoT applications deployed in edge environments. The chapter also presents a publicly available performance anomaly dataset derived from real executions, together with a trace-driven dataset generation simulator that enables scalable evaluation under large system sizes. This chapter is derived from:

  Duneesha Fernando, Maria A. Rodriguez, and Rajkumar Buyya, "iAnomaly: A Toolkit for Generating Performance Anomaly Datasets in Edge-Cloud Integrated Computing Environments", *Proceedings of the 17th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2024)*, Pages: 236–245, Sharjah, United Arab Emirates, December 16-19, 2024.

- Chapter 4 focuses on improving the scalability of anomaly detection model training in heterogeneous edge infrastructures. It proposes two clustering-based training strategies, namely intra-cluster parameter transfer learning (ICPTL) and cluster-level model training (CM), which reduce training time and management overhead while preserving detection accuracy. The chapter evaluates these strategies on both public and emulated datasets, demonstrating their effectiveness compared to traditional per-device and generic training approaches. This chapter is derived from:

  Duneesha Fernando, Maria A. Rodriguez, Patricia Arroba, Leila Ismail, and Rajkumar Buyya, "Efficient Training Approaches for Performance Anomaly Detection Models in Edge Computing Environments", *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, Volume 20, Pages: 1-27, ISSN: 1556-4665, Association for Computing Machinery, New York, NY, USA, June 2025.

- Chapter 5 investigates the limitations of centralized RCL in distributed edge environments. It presents a decentralized localization framework based on communication and colocation-aware clustering and PPR algorithm, enabling efficient diagnosis directly at the edge. The chapter further introduces an edge-specific anomaly scoring mechanism and a lightweight P2P approximation strategy to handle multi-cluster anomaly propagation, and evaluates the approach using both real and large-scale synthetic datasets. This chapter is derived from:

- **Duneesha Fernando**, Maria A. Rodriguez, and Rajkumar Buyya, "A Decentralized Root Cause Localization Approach for Edge Computing Environments", *IEEE Transactions on Services Computing (TSC)*, pp. 1–14, February 2026, doi: 10.1109/TSC.2026.3668228.

- Chapter 6 addresses the scalability challenges of state-of-the-art supervised GNN–based diagnosis techniques. It proposes a cascaded GNN architecture for joint RCL and fault type identification, combining communication-driven clustering with hierarchical inference. By decomposing global dependency graphs into structured subgraphs, the proposed approach achieves near-constant inference latency at scale while maintaining diagnostic accuracy comparable to centralized GNN baselines. This chapter is derived from:

  Duneesha Fernando, Maria A. Rodriguez, and Rajkumar Buyya, "A Cascaded Graph Neural Network for Joint Root Cause Localization and Analysis in Edge Computing Environments", *submitted to IEEE Transactions on Services Computing (TSC), March 2026* (Under Review, available on arXiv:2603.01447).

- Chapter 7 presents the conclusions of this thesis, summarizes the key contributions, discusses the implications of the research findings, and outlines future research directions for advancing anomaly-aware management of microservice-based edge–cloud systems.

Chapter 2

A Review and Taxonomy of Performance Anomaly Detection and Diagnosis in Edge Computing Environments

Microservice-based IoT applications deployed across edge–cloud environments introduce unique operational challenges due to device heterogeneity, resource constraints, geo-distribution, and dynamic workloads. These characteristics increase susceptibility to performance anomalies and complicate both detection and diagnosis. While anomaly detection has been extensively studied in cloud environments, the assumptions underlying many existing approaches do not directly hold in edge settings. In particular, centralized processing models, homogeneous resource assumptions, and large-scale telemetry aggregation strategies are often impractical or inefficient at the edge.

This chapter reviews existing research on performance anomaly detection and diagnosis for microservices-based systems, with a focus on edge computing environments. The review is organized around the performance anomaly management lifecycle, comprising monitoring, anomaly detection (AD), root cause localization (RCL), root cause analysis (RCA), and mitigation. For the detection and diagnosis stages, we introduce taxonomies to systematically analyze existing approaches. In addition, the chapter examines experimental environments and benchmarking platforms used in prior studies. The chapter also identifies key research gaps and opportunities that motivate the anomaly detection and root cause diagnosis techniques developed in the subsequent chapters of this thesis.

This chapter is derived from:

- **Duneesha Fernando**, Maria A. Rodriguez, and Rajkumar Buyya, "Performance Anomaly Detection and Diagnosis for Microservices-based IoT Applications in Edge Computing Environments: A Taxonomy and Future Directions", *submitted to ACM Computing Surveys (CSUR), March 2026.*

2.1 Edge Computing and Microservices-based IoT Applications

The rapid proliferation of IoT has fundamentally transformed the computing landscape by enabling large-scale sensing, actuation, and data-driven intelligence across domains such as healthcare, transportation, smart cities, and industrial automation [31, 32]. Early IoT deployments primarily relied on centralized cloud infrastructures for data processing and storage. In this cloud-centric model, data generated by geographically distributed IoT devices were transmitted to remote data centers for analysis and decision-making. While this approach offers virtually elastic compute and storage capacity, it introduces substantial communication latency, network congestion, and bandwidth overhead, particularly for latency-sensitive and bandwidth-intensive applications [2, 33].

To address these limitations, edge computing has emerged as a complementary paradigm that extends cloud capabilities toward the network edge [33, 34]. In this thesis, an *edge computing environment* is defined as an integrated deployment model spanning both edge and cloud resources, where edge nodes consist of any device with computational and storage capabilities located along the path between IoT devices and centralized cloud data centers. These nodes include gateways, routers, micro-datacenters, and edge servers. Some literature uses the term *fog computing* to describe similar intermediate infrastructures [3, 4, 35]. Regardless of terminology, such environments are characterized by heterogeneous computing, storage, and networking capacities, hierarchical organization, and large-scale geographical distribution [34, 36].

Compared to centralized clouds, edge infrastructures exhibit several distinctive properties. First, edge nodes are resource-constrained relative to cloud servers, yet substantially more capable than end IoT devices [2, 33]. Second, they are highly heterogeneous, encompassing devices with diverse hardware configurations, virtualization capabilities, and connectivity characteristics [36]. Third, they are widely distributed, often deployed across multiple administrative domains and physical locations. Finally, they operate under dynamic conditions, including fluctuating workloads, mobility of devices, and varying network quality [37]. These characteristics collectively form what is often referred to as the *edge–cloud continuum*, in which application components are dynamically distributed across multiple tiers according to QoS requirements and resource availability

[3, 38].

2.1.1 Microservices as the Architectural Enabler for Edge–Cloud IoT Systems

To effectively exploit the edge–cloud continuum, IoT applications must be designed in a manner that supports modularity, elasticity, and fine-grained deployment control. Monolithic architectures are ill-suited for such environments, as they require co-locating all application functionality on a single node or tier, limiting flexibility and scalability. Consequently, microservice architectures have become the de facto design paradigm for modern distributed IoT systems [39, 40].

Microservices decompose an application into loosely coupled, lightweight, independently deployable modules that communicate via well-defined interfaces. Each microservice encapsulates a specific functional capability, such as data acquisition, preprocessing, analytics, aggregation, or actuation. Containerization technologies and lightweight virtualization further facilitate their deployment across heterogeneous edge infrastructures [41, 42]. This architectural style provides several advantages in edge computing environments:

1. **Fine-Grained Placement Control:** Individual microservices can be deployed at different tiers of the edge–cloud hierarchy based on their latency sensitivity, computational intensity, and data locality requirements [9].

2. **QoS-Aware Optimization:** Latency-critical and bandwidth-intensive components can be placed closer to data sources, while computation-intensive but delay-tolerant components can be offloaded to higher-tier edge nodes or cloud data centers [6, 34].

3. **Independent Scaling:** Services experiencing workload surges can be scaled independently without affecting the entire application [40].

4. **Resource Efficiency:** Granular deployment allows better utilization of heterogeneous edge resources by matching service requirements with device capabilities [9].

Several studies have leveraged microservice abstractions to model IoT applications in fog and edge environments [7, 8]. These applications typically consist of interconnected processing pipelines where upstream services ingest sensor data, intermediate services perform filtering and aggregation, and downstream services execute analytics or trigger control actions. Such modular decomposition is particularly suited to edge deployments where application components must be distributed across multiple tiers while preserving service dependencies and QoS guarantees.

For example, in a smart traffic management system, data ingestion and preliminary filtering services may execute on roadside edge gateways to ensure low-latency responses, while more computationally intensive prediction or optimization modules may run on regional edge servers or the cloud [34]. Similarly, in a smart healthcare monitoring application, real-time anomaly detection modules may execute at hospital edge nodes to guarantee timely alerts, while long-term trend analysis and model retraining are performed in the cloud.

2.1.2 Microservice Placement in Edge Computing Environments

Given the heterogeneity and resource constraints of edge infrastructures, determining the optimal placement of microservices across the edge–cloud continuum is a non-trivial optimization problem [3, 43]. Numerous placement and scheduling algorithms have been proposed to address this challenge, with many works focusing primarily on meeting application-level performance requirements.

Existing approaches can be broadly categorized as follows:

1. **QoS-Aware Placement**: These methods primarily aim to satisfy application-level QoS requirements, such as latency bounds, throughput guarantees, or deadline constraints. Resource capacities are treated as constraints that must be respected while ensuring SLA compliance [9, 10, 43]. Due to the latency-sensitive nature of many IoT applications, QoS-aware placement has emerged as a central design objective in edge computing research.

2. **Communication-Aware Placement**: These techniques minimize inter-device communication overhead by placing frequently communicating microservices in close

network proximity. Reducing cross-device traffic lowers end-to-end latency, decreases bandwidth consumption, and mitigates energy usage in resource-constrained environments [44, 45].

3. **Resource-Aware and Multi-Objective Placement**: These approaches focus on optimizing infrastructure-level objectives such as CPU utilization, energy consumption, load balancing, and operational cost. QoS may be included as one objective among several, but the primary goal is efficient resource management across the edge–cloud infrastructure [46].

Collectively, these placement strategies aim to deploy microservice-based IoT applications across heterogeneous edge–cloud resources in a manner that preserves application performance while efficiently utilizing available infrastructure capacity. However, placement decisions are typically made based on estimated workloads and predicted resource demands. Over long execution periods, dynamic factors such as workload surges, user mobility, infrastructure failures, and multi-tenant interference may invalidate initial placement assumptions [33, 37].

2.1.3 Dynamism in Edge Environments and the Emergence of Performance Problems

Edge computing environments are inherently dynamic [34]. Workload characteristics may fluctuate significantly due to user behavior patterns, time-of-day effects, or sudden events. Resources at edge nodes may be shared among multiple tenants, leading to unpredictable contention. Furthermore, emerging paradigms such as osmotic computing promote the dynamic migration of microservices between edge and cloud tiers based on real-time resource availability and performance considerations [47]. While such flexibility improves adaptability, it also introduces additional operational complexity.

Even when initial microservice placement satisfies QoS constraints, prolonged execution under dynamic conditions may lead to performance degradation. Resource contention caused by co-located services, CPU or memory saturation during workload spikes, network congestion, or interference from other applications can trigger deviations from expected performance behavior [11, 12]. These degradations manifest as abnormal variations in latency, throughput, or resource utilization metrics and may propagate across dependent microservices due to communication and colocation relationships [14, 15].

Thus, while microservice-based architectures and intelligent placement algorithms enable efficient deployment across edge–cloud infrastructures, they also introduce complex interdependencies that make long-running IoT applications susceptible to performance anomalies. Understanding this architectural and operational context is essential before examining the nature of such anomalies and their management.

The next section therefore focuses on characterizing performance anomalies in microservice-based edge IoT systems and analyzing how the unique properties of edge environments amplify their occurrence and propagation.

2.2 Performance Anomalies in Microservices-based Edge IoT Systems

Anomalies in edge–cloud environments encompass a broad spectrum of abnormal system behaviors. In general, anomalies observed in edge–cloud infrastructures largely overlap with those studied in traditional cloud environments, with additional categories emerging due to the unique characteristics of edge computing. These characteristics include resource constraints, device heterogeneity, limited network bandwidth, geographical distribution, and reduced fault tolerance compared to centralized cloud data centers [24, 34].

In the context of edge computing, anomalies can broadly be categorized into three types:

- **Performance anomalies:** Abnormal behaviors that degrade the expected performance of applications, typically reflected through deviations in QoS- and resource-related metrics such as latency, throughput, CPU utilization, and memory usage [11, 12, 23, 24]. These anomalies have been extensively studied in cloud and microservice environments, particularly in the context of performance anomaly detection, RCL, and reliability engineering.

- **Security anomalies:** Malicious or unauthorized activities such as DDoS attacks, intrusion attempts, or abnormal traffic patterns [13, 48]. Security anomaly detection has received significant attention in both cloud and edge computing research due to the increasing attack surface introduced by distributed deployments.

- **Application anomalies:** Irregularities in application-level data streams, such as anomalous sensor readings, corrupted measurements, or abnormal user behavior patterns [49, 50]. These anomalies are typically studied in the context of data-driven anomaly detection for IoT analytics and Machine Learning (ML) applications.

While anomaly detection research in both cloud and edge environments has predominantly focused on security-related threats, comparatively fewer works address performance anomaly detection and diagnosis in distributed edge computing settings [11, 24]. This thesis focuses specifically on performance anomalies, as they directly impact the QoS of microservice-based IoT applications and may lead to SLA violations. Although security incidents can indirectly induce performance degradation (e.g., DDoS-induced CPU saturation or abnormal traffic congestion), detecting malicious intent itself falls outside the scope of this thesis. Application-level data anomalies, on the other hand, primarily concern irregularities in IoT-generated sensor data rather than infrastructure or system performance. While methodologies developed for detecting data anomalies may inspire certain modeling strategies, such anomalies are not directly related to performance management in edge–cloud infrastructures.

2.2.1 Definition of Performance Anomalies

Performance anomalies are defined in this thesis as deviations from expected behavioral patterns in QoS- and resource-related metrics that degrade application performance. Such anomalies may manifest as abnormal latency spikes, throughput drops, excessive resource utilization, or unexpected error rates. They can originate from faults, resource contention, workload surges, or network disruptions and may propagate across dependent microservices due to communication and colocation relationships [14, 15].

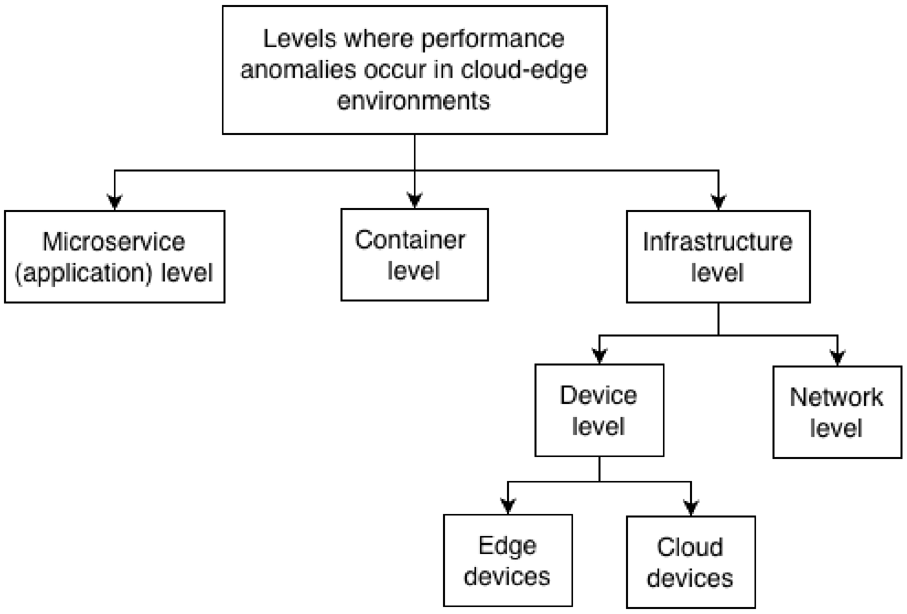

Levels where performance anomalies occur in cloud-edge environments
Microservice (application) level
Container level
Infrastructure level
Device level
Network level
Edge devices
Cloud devices

Figure 2.1: Categorization of performance anomalies by infrastructure level

Performance anomalies in edge–cloud environments can be categorized along multiple dimensions. Two complementary perspectives are particularly relevant: (i) the level at which the anomaly occurs, and (ii) the type of anomalous behavior.

2.2.2 Categorization by Infrastructure Level

Performance anomalies may occur at different levels of the edge–cloud computing stack, as illustrated in Figure 2.1.

- **Microservice (Application) Level:** Anomalies affecting individual microservices, such as abnormal response times, request failures, or degraded throughput. These may arise from software bugs, inefficient algorithms, or cascading effects from lower-level issues.

- **Container Level:** Anomalies related to container runtime behavior, including resource throttling, misconfigured limits, or container crashes. Since microservices are typically deployed in containers, container-level disruptions can directly impact service performance.

- **Infrastructure Level:**
  - *Device Level (Edge and Cloud Nodes):* Hardware or operating system–level issues such as CPU saturation, memory exhaustion, disk I/O bottlenecks, or hardware failures.
  - *Network Level:* Packet loss, bandwidth congestion, routing failures, or kernel-level network anomalies affecting communication between services.

Compared to traditional cloud infrastructures, edge–cloud environments introduce an additional layer of vulnerability at the edge device level. Edge devices are typically more resource-constrained and less fault-tolerant than cloud servers [11, 12]. Furthermore, network links at the edge are often less stable and more exposed to failures, leading to a higher frequency of network-related anomalies [51]. Recent studies, including MicroCERCL [52], identify network-level (kernel-level) anomalies as particularly prominent in edge environments. Survey by Fu et al. [24] further emphasizes that limited bandwidth, distributed topologies, and dynamic deployment patterns increase the number of potential failure points across infrastructure, instances, and services in edge computing environments.

Failures may therefore occur in the cloud, at edge nodes, or along the communication paths between them. The distributed nature of microservices amplifies the impact of such failures, as dependencies between services can cause localized issues to cascade across multiple components.

2.2.3 Categorization by Anomaly Type

From a behavioral perspective, performance anomalies can also be classified according to their underlying characteristics, as demonstrated in Figure 2.2.

- **Local Anomalies:**
  - *Failures:* Sudden and often binary disruptions, such as service crashes, device outages, or network disconnections.
  - *Resource Saturation:* Excessive utilization of CPU, memory, disk, or network bandwidth beyond normal operating thresholds.

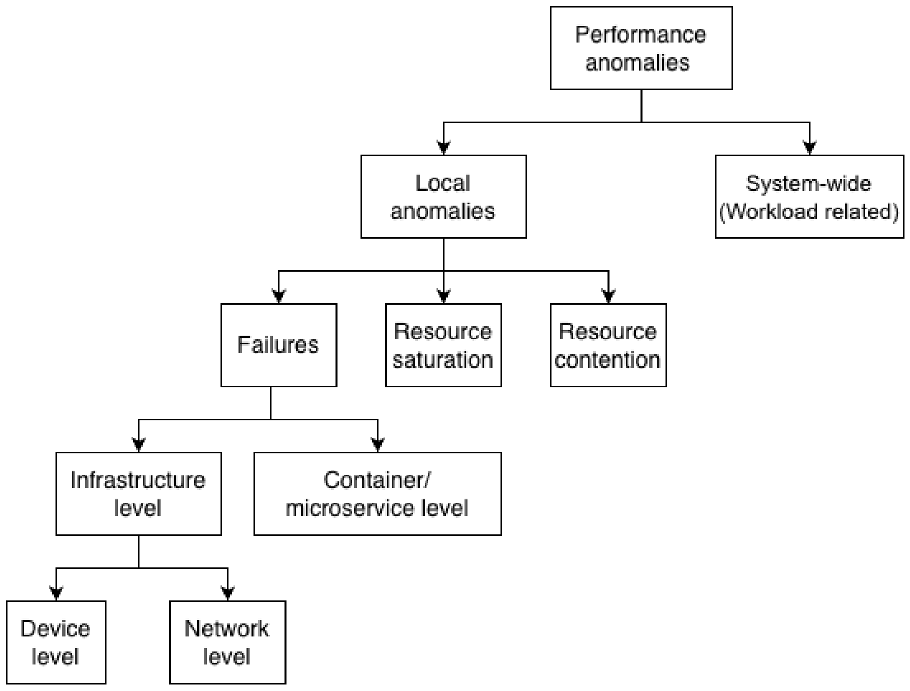


Figure 2.2: Categorization of performance anomalies by anomaly type

  - *Resource Contention:* Performance degradation caused by interference among co-located microservices competing for shared resources.

- **System-Wide Anomalies:**

  - *Workload-Related Anomalies:* Abnormal traffic surges, flash crowds, or unexpected workload spikes affecting multiple services simultaneously.

This categorization aligns with fault-type classifications adopted in prior anomaly detection and RCA studies, where anomalies are distinguished based on performance degradation, reliability failures, or traffic irregularities. For example, works such as MicroHECL [53] distinguish between performance-related, reliability-related, and traffic-related fault types in microservice systems.

Importantly, not all anomalous components represent true root causes. Some anomalies are *source anomalies*, directly responsible for performance degradation, while others are *propagated anomalies* resulting from cascading effects across communication or colocation dependencies. For instance, a network bottleneck at an edge node may delay upstream data aggregation services, which in turn causes downstream analytics services to exhibit increased latency. Although multiple services appear anomalous, only the network-level disruption constitutes the true source anomaly.

Such propagation effects are particularly pronounced in microservice-based edge environments due to their distributed architecture and tight inter-service dependencies. This complexity underscores the importance of distinguishing between anomaly detection and RCL when managing performance degradation in edge–cloud systems.

Overall, performance anomalies in microservice-based edge IoT applications can emerge at multiple infrastructure levels, manifest in diverse behavioral forms, and propagate across service dependencies. These characteristics significantly complicate detection and diagnosis, motivating the need for structured anomaly management mechanisms tailored to edge computing environments.

2.3 Mitigation Strategies for Performance Anomalies in Edge Environments

Following the categorization of performance anomalies presented in the previous section, this section reviews existing mitigation strategies in microservice-based edge environments. Mitigation mechanisms can broadly be categorized into *reactive* and *proactive* approaches. Reactive measures are triggered after anomaly detection or failure occurrence, whereas proactive measures aim to reduce the likelihood or impact of anomalies before they occur. In practice, reactive measures become necessary when protection mechanisms are insufficient to prevent performance degradation.

Across most anomaly types, mitigation ultimately reduces to two dominant mechanisms: *resource reallocation* (e.g., scaling) and *workload redistribution* (e.g., migration). However, mitigation actions operate at different system layers, including the application layer, container orchestration layer, and infrastructure layer.

Table 2.1 provides a high-level mapping from common performance anomaly types to representative reactive and proactive mitigation strategies. The remainder of this section discusses these mechanisms in more detail.

Table 2.1: Mapping of performance anomaly types to reactive and proactive mitigation strategies in edge environments

Anomaly Type	**Typical Cause**	**Reactive Mitigation**	**Proactive Mitigation**
CPU / memory hogging	Resource saturation by microservices	Horizontal scaling; vertical scaling; container restart	Autoscaling policies; resource-aware placement
User surge (flash crowd)	Sudden workload spike	Horizontal scaling; vertical scaling	Rate limiting; load shedding; autoscaling
Resource contention	Multiple services competing on same device	Migration of containers	Capacity-aware placement
Device-level failure	Hardware crash; edge node failure	Migration to healthy node; checkpoint & restore	Redundant placement; replication
Network failure / delay	Link disruption; congestion	Migration; traffic rerouting	Network-aware placement; service colocation
Process-level failure	Container crash; application fault	Restart (self-healing); checkpoint restore	Health monitoring; fail-fast design
Third-party service slowness	Slow backend; quota throttling	Invocation redistribution; migration	Circuit breaker; caching; retry with backoff

2.3.1 Reactive Mitigation Mechanisms

Reactive mitigation actions are triggered after an anomaly has been detected. Their objective is to restore acceptable performance levels or maintain service continuity under degraded conditions.

Scaling-Based Mitigation

For resource saturation anomalies such as CPU hogging, memory exhaustion, or sudden workload surges (e.g., flash crowds), scaling is the most widely adopted mitigation strategy.

- **Horizontal scaling** increases the number of microservice instances to distribute workload across replicas.
- **Vertical scaling** increases the resource limits (CPU, memory) allocated to an existing container or virtual instance.

Container orchestration platforms such as Kubernetes support autoscaling mechanisms (e.g., Horizontal Pod Autoscaling), which dynamically adjust replica counts based on resource utilization or custom metrics [54]. When memory limits are exceeded, containers may be terminated and restarted, whereas CPU limit violations typically lead to throttling rather than termination. Consequently, scaling remains the primary reactive mechanism for load-induced performance anomalies.

Migration-Based Mitigation

When anomalies arise due to resource contention, device-level failures, or network degradation, container migration is often employed. Migration redistributes workload away from overloaded or failed nodes to healthier ones while preserving microservice dependencies and scheduling constraints.

In edge environments, migration decisions must account for:

- Communication dependencies between microservices,

- Network latency and bandwidth limitations,
- Device heterogeneity,
- Limited resource availability at alternative nodes.

For infrastructure-level failures, checkpoint-and-restore mechanisms enable containers to resume execution on alternative nodes, reducing recovery time.

Restart and Self-Healing Mechanisms

Modern container orchestration platforms provide self-healing capabilities that automatically restart failed containers. At the microservice level, liveness and readiness probes detect faulty states and trigger restart procedures. Such mechanisms are particularly relevant for process-level failures and transient faults.

Third-Party Dependency Mitigation

Performance degradation may also originate from external service dependencies, such as slow backend services or quota throttling. Mitigation strategies in such cases include:

- Invocation redistribution across multiple endpoints,
- Caching frequently accessed responses,
- Asynchronous execution,
- Circuit breaker mechanisms to prevent cascading failures.

While these measures can alleviate indirect performance degradation, mitigation options are limited when third-party services exhibit sustained slowness.

2.3.2 Proactive Mitigation Mechanisms

Proactive mitigation mechanisms (also referred to as protection mechanisms) aim to reduce the probability of anomalies or limit their impact before they manifest. The extent to which such mechanisms are deployed depends on organizational design choices and system requirements. Notably, the degree of anomaly propagation within an environment is strongly influenced by the availability of proactive protection mechanisms.

Redundant Placement and Replication

Deploying redundant microservice instances across multiple devices or regions enhances resilience against infrastructure failures. In the event of a node failure, replicas can immediately serve requests without requiring migration, thereby reducing downtime and performance disruption.

Autoscaling Policies

Predictive or threshold-based autoscaling policies can be configured proactively based on historical workload patterns or anticipated demand. By allocating additional resources before saturation occurs, these mechanisms help absorb workload surges and prevent resource exhaustion.

Rate Limiting and Load Shedding

At the application layer, rate limiting (Application Programming Interface (API) throttling) restricts incoming traffic to prevent overload, while load shedding selectively drops low-priority requests during high-load conditions. These mechanisms are particularly effective in mitigating flash-crowd scenarios and preventing cascading failures.

Failure-Aware Microservice Design Patterns

Microservice architectures commonly incorporate resilience patterns that proactively limit failure propagation:

- **Retry with exponential backoff:** Failed requests are retried after progressively increasing delays to allow dependent services time to recover.

- **Circuit breaker:** Repeated failures trigger the temporary suspension of requests to a failing service, preventing cascading degradation.
- **Fail-over caching:** Cached responses are served when backend services become unavailable.
- **Fail-fast design:** Services terminate quickly under unrecoverable conditions, avoiding resource waste and request blocking.

These design patterns are particularly important in distributed edge environments where network instability and partial failures are common.

2.3.3 Layered View of Mitigation

The previously introduced mitigation mechanisms can also be examined from a layered system perspective. In addition to the reactive–proactive classification, mitigation actions may operate at different layers of the edge–cloud stack, including the application layer, container orchestration layer, and infrastructure layer.

At the **application layer**, mitigation is typically implemented through resilience patterns such as retry logic, circuit breakers, caching, and rate limiting. These mechanisms are lightweight and localized, but their effectiveness depends on accurate identification of failure conditions.

At the **container orchestration layer**, mitigation involves autoscaling, self-healing, and container restarts. These actions are coordinated by orchestration platforms and directly influence resource allocation and service availability.

At the **infrastructure layer**, mitigation includes migration, redundant placement, replication, and traffic rerouting. Such actions may incur significant communication overhead and coordination cost, particularly in geographically distributed and resource-constrained edge environments.

This layered perspective highlights an important observation: mitigation decisions often span multiple layers simultaneously. For instance, resolving latency degradation may require scaling at the orchestration layer, migration at the infrastructure layer, or application-level throttling. In distributed edge–cloud environments, uninformed cross-layer mitigation can introduce additional overhead or instability, particularly under resource constraints.

This cross-layer nature of mitigation amplifies the importance of accurate diagnosis, since the appropriate response depends on both where the anomaly originates and why it occurs.

2.3.4 Diagnosis as a Prerequisite for Effective Mitigation

Although a wide range of mitigation mechanisms exist across application, orchestration, and infrastructure layers, their effectiveness fundamentally depends on accurate identification of both the anomaly source and its underlying cause. Scaling, migration, restart, or rate limiting actions may alleviate performance degradation; however, without precise diagnosis, such actions risk addressing symptoms rather than root causes.

For example, scaling a microservice experiencing latency may temporarily mask the symptom if the true cause is network congestion or resource contention at the device level. Similarly, migrating containers without understanding communication dependencies may increase cross-device traffic and further degrade performance. In distributed edge–cloud environments, where resources are constrained and inter-service dependencies are complex, uninformed mitigation decisions can introduce additional overhead and instability.

These observations indicate that mitigation cannot be treated as an isolated operational step. Instead, it must be embedded within a structured anomaly management process that begins with monitoring and progresses through detection and diagnosis before remediation decisions are made.

2.4 Performance Anomaly Management Lifecycle

Managing performance anomalies in microservice-based edge–cloud environments requires a coordinated sequence of monitoring, detection, diagnosis, and mitigation activities. Rather than isolated actions, these stages form an interconnected anomaly management lifecycle that governs how performance degradation is identified and resolved in distributed systems.

From an operational perspective, the performance anomaly management lifecycle consists of five primary stages:

1. **Monitoring (Observability)**
2. **Anomaly Detection**
3. **Root Cause Localization (RCL)**
4. **Root Cause Analysis (RCA)**
5. **Mitigation**

Figure 1.2 illustrates this lifecycle.

Monitoring (Observability) Monitoring refers to the continuous collection of system-level data, including metrics, traces, and logs, from microservices, containers, edge devices, and network components. These observability signals provide visibility into system behavior and form the foundation for anomaly detection.

Anomaly Detection Anomaly detection determines whether system behavior deviates from expected patterns in monitored metrics. Detection mechanisms may rely on statistical thresholds, ML models, or Deep Learning (DL) techniques. The goal at this stage is to identify the presence of abnormal behavior.

Root Cause Localization (RCL) RCL aims to identify which component is responsible for the observed anomaly. In distributed microservice architectures, this typically involves analyzing service dependency graphs, communication patterns, and resource metrics to pinpoint the anomalous microservice, container, or infrastructure component.

Root Cause Analysis (RCA) While RCL determines the location of the anomaly, RCA seeks to understand why the anomaly occurred. For example, a microservice may exhibit high latency due to CPU saturation, memory pressure, network delay, or resource contention. Accurate RCA enables selection of the most appropriate mitigation strategy.

Mitigation and Feedback Loop Mitigation constitutes the operational response stage of the lifecycle. Based on the outputs of RCL and RCA, mitigation mechanisms such as scaling, migration, restart, replication, or rate limiting are applied to restore acceptable performance levels.

Importantly, mitigation actions modify system state, potentially introducing new performance dynamics. Consequently, the lifecycle operates as a feedback loop: post-mitigation monitoring evaluates whether performance has stabilized or whether further intervention is required.

The remainder of this chapter systematically examines the performance anomaly management lifecycle in the context of microservice-based IoT applications deployed across edge–cloud environments. While all lifecycle stages are reviewed to provide a coherent end-to-end perspective, particular emphasis is placed on the diagnosis phases, namely anomaly detection (AD) and root cause localization/analysis (RCL/RCA), which constitute the primary technical focus of this thesis.

Section 2.5 discusses edge monitoring infrastructures and observability mechanisms that provide the foundational telemetry signals required for anomaly management. Section 2.6 presents a taxonomy of performance anomaly detection techniques for edge environments. Section 2.7 introduces a taxonomy for RCL and analysis approaches at the edge. Section 2.8 reviews how mitigation strategies are operationalized in edge environments following anomaly identification. Finally, Section 2.9 analyzes the experimental environments and evaluation platforms adopted in prior work.

Table 2.3 presents a classification of existing edge-focused studies analyzed along the dimensions discussed under each stage of the performance anomaly management lifecycle.

2.5 Edge Monitoring Infrastructure and Observability

Although comparatively fewer studies address advanced anomaly detection and diagnosis at the edge, monitoring and telemetry collection for edge–cloud environments is relatively mature. This maturity is largely driven by operational needs such as resource accounting, autoscaling, orchestration, and service management across the cloud–edge continuum [55–57]. Consequently, this thesis does not propose a new monitoring architecture; instead, the anomaly detection and diagnosis techniques developed in subsequent chapters assume the availability of existing monitoring infrastructure and build upon its collected observability signals.

2.5.1 Observability Signals in Edge–Cloud Systems

The continuous collection of monitoring data is commonly framed through the concept of observability, which characterizes how well internal system states can be inferred from externally collected signals [58, 59]. In modern cloud-native systems, observability is typically operationalized through three complementary signal types: metrics, logs, and traces. This three-signal view is widely adopted in practice and is reflected in contemporary observability standards and tooling (e.g., OpenTelemetry) [60].

Metrics provide time-series measurements of system and application behavior (e.g., latency, throughput, CPU, memory). In microservice-based edge systems, metrics are often collected at multiple granularities: (i) application/microservice-level metrics such as request latency distributions, error rates, and throughput; (ii) container-level metrics such as cgroup CPU throttling, memory usage, and restart counts; and (iii) infrastructure-level metrics including device resource utilization and network QoS (e.g., packet loss, link delay, bandwidth) [57, 61]. This multi-level metric collection is particularly important in edge environments because performance anomalies may originate at any layer of the stack, including edge devices and edge networks.

Logs capture discrete events and messages emitted by services, runtimes, and operating systems. They are useful for post-hoc diagnosis, error analysis, and interpreting rare failure conditions, especially when symptoms are not fully reflected in metrics [62].

Traces record end-to-end request execution paths across microservices and are essential for understanding dependency and propagation effects. In particular, distributed traces can be used to derive service call graphs or dependency graphs, which are widely leveraged by diagnosis techniques (including graph-based localization approaches) to reason about anomaly propagation across service interactions [23, 24].

Together, metrics, logs, and traces provide complementary visibility: metrics offer coarse-grained continuous signals, logs provide rich contextual events, and traces reveal cross-service causal paths.

2.5.2 Monitoring Architectures in Edge Environments

A key design axis in edge monitoring is where monitoring agents run and where analysis occurs. Due to geographical distribution, device heterogeneity, and limited bandwidth, edge monitoring commonly adopts decentralized data collection at the device level, often combined with centralized or hierarchical aggregation for global visibility [61, 63]. In such designs, lightweight agents on each edge node collect local metrics, logs, and traces and either: (i) forward summarized or raw telemetry to a centralized location for storage and analysis [63, 64], or (ii) perform partial analysis locally and transmit only derived signals or alerts when necessary [61].

Hierarchical aggregation models are particularly common in multi-tier deployments, where regional edge servers act as intermediate collectors before forwarding data to cloud-level control planes. This structure balances bandwidth efficiency with system-wide observability.

More recent studies investigate fully decentralized monitoring architectures that eliminate centralized aggregation points altogether [65, 66]. These approaches leverage P2P communication mechanisms (e.g., gossip-based monitoring) to disseminate monitoring information across nodes. These approaches are motivated by highly volatile edge settings where centralized collection can become unreliable or too costly.

The prevalence of decentralized monitoring agents is particularly important in the context of anomaly management. Since each edge node already maintains localized observability signals, it becomes feasible to deploy anomaly detection and diagnosis capabilities directly at the device level, thereby avoiding continuous transmission of high-frequency telemetry to centralized analysis engines.

Building upon this architectural principle, this thesis assumes a monitoring design in which each edge device hosts: (i) a monitoring agent responsible for collecting multi-level observability signals (microservice-, container-, device-, and network-level met-

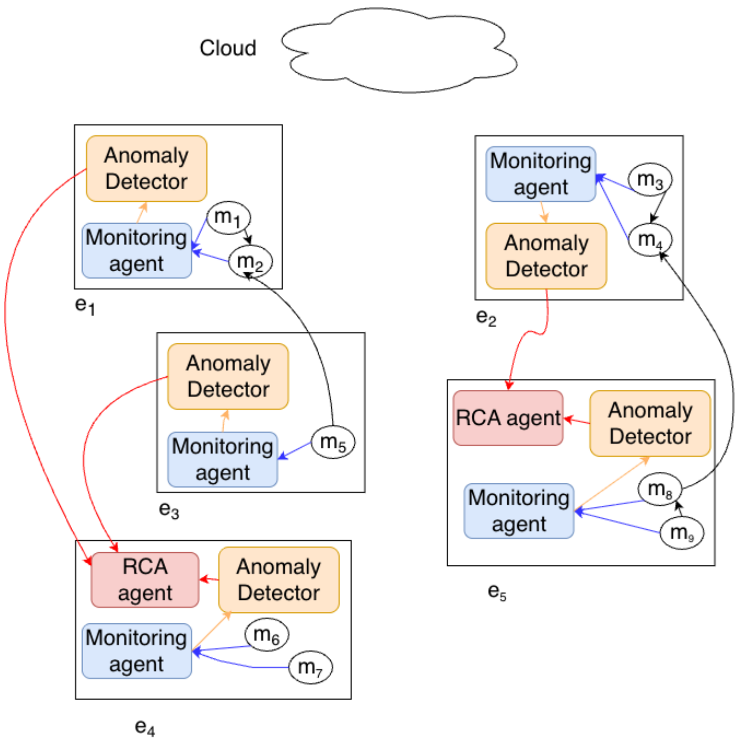


Figure 2.3: Device-level monitoring and anomaly detection/diagnosis architecture in edge–cloud environments (e_i refers to an edge device. m_i refers to a microservice.)

rics), and (ii) a local anomaly detection module capable of analyzing multivariate time-series data associated with the microservices deployed on that device. Figure 2.3 illustrates this architecture.

This design aligns with prior edge-focused anomaly detection frameworks [11], where anomaly detection models are deployed per edge device to enable localized inference under bandwidth constraints. Extending this rationale beyond detection, RCL and RCA can also benefit from device-proximal execution. Since anomalies often originate from device-level resource contention, container-level throttling, or localized network degradation, performing diagnosis closer to the source reduces detection latency and limits unnecessary cross-device data exchange.

Overall, existing edge monitoring architectures already provide the necessary decentralized observability foundation required for device-level anomaly detection and diagnosis. Rather than introducing new monitoring mechanisms, this thesis leverages existing decentralized telemetry collection to develop scalable and efficiency-aware detection and localization techniques for distributed edge–cloud systems. The subsequent sections, therefore, focus on anomaly detection and RCA approaches that operate atop this monitoring foundation.

2.6 Performance Anomaly Detection in Edge Environments

This section reviews performance anomaly detection (AD) techniques applicable to microservices based IoT applications deployed across the edge–cloud continuum and introduces a taxonomy to structure the literature. In contrast to cloud-centric AD, where large-scale centralized data collection and powerful servers often make accuracy the primary design concern, edge settings impose additional constraints including limited compute and memory, intermittent and bandwidth-limited connectivity, and heterogeneous device capabilities. These properties push AD solutions toward decentralized inference, lightweight models, and efficient training and update strategies, while still maintaining acceptable detection effectiveness.

2.6.1 Taxonomy Overview

Figure 2.4 presents the taxonomy used in this thesis to organize existing performance AD work for microservices-based IoT applications in edge environments. The taxonomy spans seven dimensions: *(i) observability modality*, *(ii) learning paradigm*, *(iii) modeling technique*, *(iv) evaluation criteria*, *(v) design objective*, *(vi) model training strategy*, and *(vii) lightweight mechanisms*. The taxonomy aligns with established categorizations in cloud-native microservice monitoring and diagnosis surveys, while emphasizing edge-specific concerns such as deployability on resource-constrained devices and reduced reliance on centralized telemetry aggregation [23]. The remainder of this section discusses each dimension and positions representative edge-focused studies accordingly.

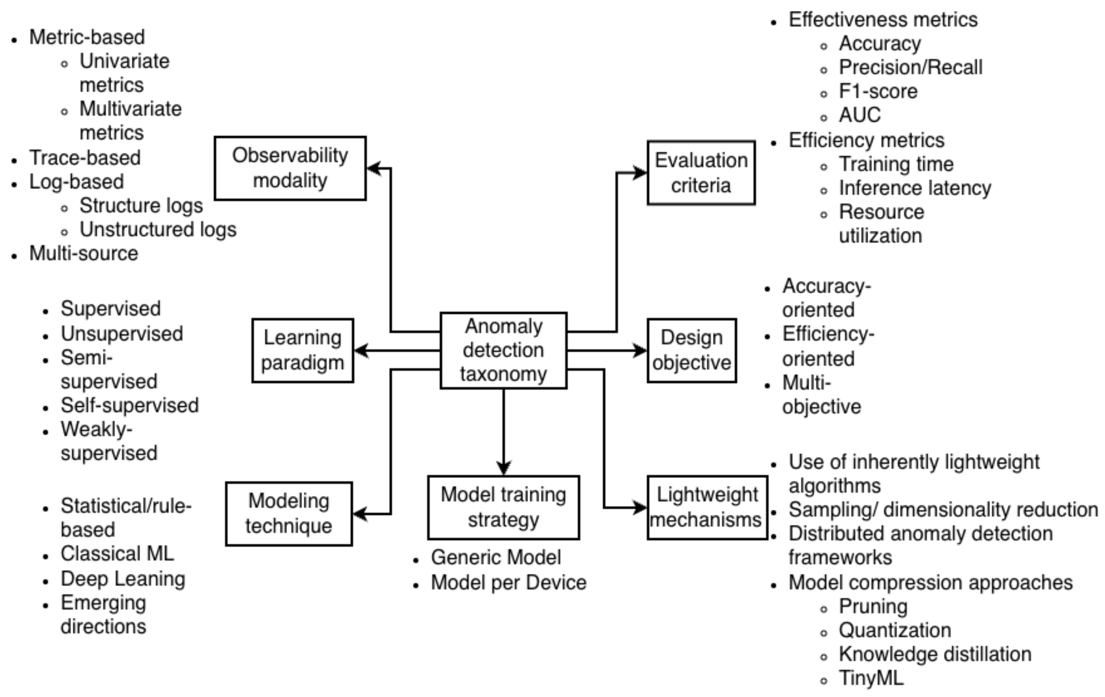


Figure 2.4: Anomaly detection taxonomy

2.6.2 Observability Modality

Performance anomalies can be detected from different observability modalities, including metrics (KPIs), traces, logs, and multi-source combinations. Survey literature on microservice systems highlights that each modality offers different trade-offs in terms of granularity, semantic richness, collection overhead, and scalability [23].

Trace-based AD exploits distributed traces to capture end-to-end request paths and latency contributions across microservices. Trace analysis enables reasoning about propagation effects and critical-path delays, making it particularly useful for distributed performance diagnosis [58, 67]. However, trace collection incurs non-trivial instrumentation and storage overhead, and incomplete tracing coverage may limit effectiveness in resource-constrained edge deployments.

Log-based AD relies on events emitted by applications, containers, and operating systems. Log-based detection can capture semantically rich contextual information that may not be visible in metrics alone. Structured logs (e.g., JSON logs) follow consistent schemas that allow direct feature extraction with minimal preprocessing, whereas unstructured logs require template mining and NLP-based processing (e.g., Drain-style parsing or embedding-based representations) prior to anomaly detection [68, 69]. While log-based AD has shown strong results in cloud systems, its computational overhead may challenge direct deployment on low-power edge devices.

Metric/Key Performance Indicator (KPI)-based AD is the dominant modality in operational monitoring because metrics are continuously collected, compact, and suitable for real-time analysis [12, 16, 70]. Within metric-based AD, it is important to distinguish between **univariate** and **multivariate** time-series approaches. Many production monitoring stacks rely on threshold-based alerting applied to univariate streams [55, 71]. Such thresholding requires manual per-metric configuration and does not scale well in large, heterogeneous edge deployments, particularly under workload drift.

Univariate methods treat each metric independently and may fail to detect anomalies that arise from cross-metric interactions. In contrast, multivariate approaches explicitly model correlations among metrics (e.g., CPU–latency or memory–throughput relationships), enabling detection of more complex anomaly patterns in distributed microservice systems [72, 73]. Reflecting this advantage, existing edge performance AD studies predominantly adopt multivariate telemetry as input [11, 12, 16, 70].

2.6.3 Learning Paradigm

Edge performance AD studies predominantly adopt **unsupervised** learning. The primary motivation is practical: labeled anomaly data is rarely available during initial deployments, and manual labeling across distributed edge sites is costly. Unsupervised models are typically trained on unlabeled (or mostly normal) data to learn baseline behavior and detect deviations as anomalies.

Becker et al. evaluate several unsupervised approaches, including ARIMA, BIRCH, LSTM, and EMA, under edge constraints [11]. Skaperas et al. focus on unsupervised change-point detection techniques such as Bayesian methods and CUSUM, emphasizing lightweight deployability [18]. Tuli et al. evaluate one-class and reconstruction-based methods while discussing deployment feasibility in edge environments [16]. Similarly, Wang et al. propose an unsupervised anomaly detection method based on a Gated Recurrent Unit (GRU)-autoencoder to identify reliability anomalies in vehicular fog computing services, where the model learns normal temporal patterns of service metrics and detects anomalies through reconstruction errors [70]. Even when **supervised** methods are proposed, their reliance on labeled anomaly data is often identified as a practical limitation in edge settings [12]. A similar reliance on unsupervised anomaly detection can also be observed in the majority of edge RCA approaches, where anomaly detection modules are used to trigger root cause diagnosis [52, 74, 75].

Beyond supervised/unsupervised, **semi-supervised**, **self-supervised**, and **weakly supervised** paradigms are increasingly explored in cloud AD and streaming settings to cope with limited labels and drift [76, 77]; however, their systematic adoption at the edge remains comparatively limited, largely due to resource constraints and deployment complexity.

2.6.4 Modeling Technique

Following established AD taxonomies, anomaly detection techniques can be broadly grouped into: *(i) conventional/statistical methods*, *(ii) classical machine learning (ML) methods*, and *(iii) deep learning (DL) methods* [78].

Conventional/statistical methods include thresholding, statistical process control techniques (e.g., EWMA, CUSUM) [79], and classical time-series forecasting models such as ARIMA [80]. These approaches are typically lightweight, interpretable, and computationally inexpensive, making them attractive for resource-constrained environments. However, they typically assume stationarity or low-dimensional inputs and are limited in their ability to model nonlinear dependencies and complex interactions within high-dimensional microservice telemetry.

In practice, many existing monitoring solutions for edge–cloud systems integrate alerting mechanisms that are primarily threshold- or rule-based [55, 71]. They allow operators to define static alerting rules (e.g., latency exceeding a predefined bound or CPU utilization surpassing a threshold) and trigger notifications when such conditions are violated [81]. While effective for well-understood and deterministic failure modes, these mechanisms typically do not constitute advanced anomaly detection in the sense of adaptive baselining, multivariate modeling, or learning-based inference.

As edge–cloud microservice systems grow in scale and complexity, static thresholding becomes increasingly insufficient. Performance degradation may emerge gradually, propagate across services, or manifest only through correlated deviations across multiple metrics. In such scenarios, rule-based alerting can generate excessive false positives or fail to capture subtle anomalies altogether. Consequently, fault handling frequently devolves into manual inspection and reactive troubleshooting, underscoring the limitations of conventional detection techniques in distributed edge environments.

Classical machine learning (ML) methods include one-class classification (e.g., One-Class SVM [82]), isolation-based methods (e.g., Isolation Forest (IF) [83]), and clustering-based techniques (e.g., *k*-means, DBSCAN). These approaches operate on engineered features extracted from telemetry streams and often provide a favorable effectiveness–efficiency trade-off. As a result, they are frequently used as baselines in both cloud and edge anomaly detection studies.

Deep learning (DL) methods leverage representation learning to model complex temporal and cross-metric dependencies. Reconstruction-based approaches (e.g., autoencoders and adversarial autoencoders [84, 85]) detect anomalies via reconstruction error. Sequence models such as Long Short-Term Memory (LSTM) and GRU networks capture temporal dependencies in multivariate time series [73, 86]. Convolutional architectures have also been adapted for temporal pattern learning in structured telemetry data. Deep models generally improve detection performance under nonlinear and non-stationary dynamics, but their higher training and inference cost can challenge deployment on resource-constrained edge devices.

Edge performance AD studies often adopt this three-group evaluation methodology when comparing candidate algorithms under resource constraints. Many edge works adapt and evaluate techniques originally proposed for cloud AD, benchmarking statistical, classical ML, and DL methods under edge-specific constraints [11, 12, 16, 18, 70].

Within edge RCA pipelines, the anomaly detection component commonly relies on classical ML techniques, particularly clustering-based methods [27, 52, 75]. The prevalence of clustering-based detectors largely stems from their unsupervised nature, computational efficiency, and scalability, making them well suited for resource-constrained edge deployments where lightweight anomaly triggers are required before more sophisticated diagnosis mechanisms are invoked.

An exception is CIRCA [87], which employs a more expressive anomaly detection pipeline incorporating Graph Attention Network (GAT)s, GRU, self-attention mechanisms, and Graph Fourier Transform (GFT). While such DL architectures can better capture complex temporal and structural dependencies, their higher computational overhead highlights an important accuracy–efficiency trade-off that must be considered when deploying anomaly detection at the edge.

Emerging directions. Beyond conventional, classical ML, and DL approaches, recent anomaly detection research in cloud environments increasingly explores more expressive model families such as **Transformers** and **graph neural networks (GNNs)**. Transformer-based models leverage attention mechanisms to capture long-range temporal dependencies in multivariate telemetry streams [77, 88]. Similarly, graph-based models exploit service dependency structures or metric correlation graphs to reason about anomaly propagation in distributed systems [28, 89]. While these approaches demonstrate strong detection capabilities in cloud-scale environments, they typically assume centralized telemetry collection and substantial computational resources. As a result, their practical deployment in edge environments often requires additional mechanisms such as model compression, distillation, or hierarchical inference architectures.

Another emerging research direction involves **Quantum Machine Learning (QML)**-based approaches for anomaly detection. Recent surveys and exploratory studies discuss the potential use of quantum anomaly detection and quantum autoencoder-based time-series analysis [90, 91]. These works are primarily motivated by potential computational advantages offered by future quantum hardware. However, QML techniques have not yet been incorporated into edge performance anomaly detection pipelines in a meaningful way. This limitation is consistent with the practical constraints of current edge environments, including limited compute and memory resources and the absence of accessible quantum hardware at deployment sites.

2.6.5 Evaluation Criteria

Cloud and microservice AD studies commonly prioritize **effectiveness** metrics such as precision, recall, F1-score, and AUC, reflecting an accuracy-oriented evaluation culture in centralized infrastructures [23, 78]. In edge environments, however, some studies additionally evaluate both effectiveness and **efficiency**, since resource overhead directly constrains deployability. In particular, several edge AD works report inference-time CPU and memory usage as well as detection latency, and some additionally report training time and memory footprint when training is executed on edge nodes [11, 12, 16, 18]. Nevertheless, not all edge-oriented studies evaluate efficiency explicitly. For example, Wang et al.'s evaluation primarily focuses on detection effectiveness metrics such as precision, recall, and F1-score, reflecting the continued emphasis on accuracy even in some edge-oriented studies [70]. This variation motivates treating evaluation metrics as distinct from the *design objective*: some works measure efficiency without explicitly optimizing for it, whereas others explicitly design algorithms or systems to meet latency/throughput/resource targets.

2.6.6 Design Objective

In much of the anomaly detection literature for cloud and microservice systems, the primary design objective is **improving detection accuracy**. Many studies develop increasingly expressive models to capture complex temporal patterns and cross-metric dependencies in large-scale telemetry data. Consequently, evaluation typically emphasizes effectiveness metrics such as precision, recall, and F1-score, while efficiency considerations such as computational cost or runtime overhead receive less attention [23, 78]. In some cases efficiency metrics are reported, but the underlying methods are not explicitly designed to optimize resource consumption.

Edge environments introduce different priorities. Since anomaly detection models may operate on heterogeneous and resource-constrained devices, **both detection effectiveness and operational efficiency** become critical design objectives. Edge-focused studies therefore often evaluate multiple algorithms and select models that provide an acceptable accuracy–efficiency trade-off. For example, Becker et al. compare statistical, classical ML, and DL approaches under edge constraints to identify feasible detection techniques [11]. Similarly, Skaperas et al. investigate lightweight change-point detection algorithms suitable for deployment on edge devices [18], while Tuli et al. study generative models for anomaly detection and discuss their feasibility in distributed edge environments [16]. Although the work targets edge environments, the model design primarily focuses on improving detection effectiveness rather than optimizing computational efficiency.

Although efficiency has historically received less attention in cloud-based AD research, a few studies explicitly target performance optimization goals such as reducing training time, lowering inference latency, or improving throughput. Examples include streaming anomaly detection frameworks designed for large-scale monitoring pipelines (e.g., the Microsoft KDD'19 service anomaly detection system [92] and Skyline [93]) as well as distributed streaming analytics platforms built on systems such as Apache Flink [94]. Some recent work also considers energy-aware model design; for instance, USAD proposes an adversarial autoencoder architecture that improves computational efficiency while maintaining competitive detection accuracy [85]. These studies provide useful insights for designing more efficient anomaly detection pipelines that may also benefit edge deployments.

Despite these efforts, cloud AD research generally assumes access to powerful centralized infrastructure and rarely targets deployability on resource-constrained devices. Objectives such as minimizing model memory footprint, reducing energy consumption, or enabling inference on low-power edge hardware are seldom addressed explicitly. Consequently, cloud-based approaches provide limited guidance for designing models suitable for highly resource-constrained environments. In this regard, research on lightweight ML for IoT and embedded systems offers more relevant design insights, as these works explicitly focus on developing models that operate within strict compute, memory, and energy constraints [95–97].

2.6.7 Model Training Strategy

Two dominant training strategies appear in edge–cloud AD deployments.

Generic Model (GM). In the GM strategy, a single model is trained using aggregated data collected across devices and then deployed broadly. This is common in cloud-centric pipelines and in some edge–cloud MLOps settings because it reduces model management overhead and can be efficient in centralized training [21, 22, 70]. However, GM often suffers from reduced detection accuracy in heterogeneous edge environments, where devices exhibit diverse QoS characteristics and normal data distributions. Furthermore, centralized aggregation of high-frequency telemetry can increase network utilization and introduce additional overhead, particularly when bandwidth is limited [16].

Model per Device (MPD). In the MPD strategy, each edge device trains and maintains its own model using local data. This approach can improve detection accuracy by specializing to device-specific baselines and workload patterns, especially when services co-located on a device share QoS characteristics [9, 11, 16, 98]. The trade-off is higher aggregate training time across the fleet, greater total resource consumption, and increased model management complexity. These costs can be prohibitive for large-scale deployments unless training is made substantially more efficient or partially offloaded.

Overall, GM and MPD represent opposing trade-offs: GM improves training efficiency and operational simplicity but risks poor generalization under heterogeneity, whereas MPD improves specialization and accuracy but increases training overhead and infrastructure-wide resource demand.

2.6.8 Lightweight Mechanisms

To meet the efficiency requirements of edge environments, anomaly detection systems often incorporate lightweight mechanisms that reduce computational cost, memory usage, or telemetry processing overhead while maintaining detection effectiveness.

One common strategy is the use of **inherently lightweight algorithms**. Statistical process control methods and change-point detection techniques are frequently favored because they require minimal computation and memory. For example, Skaperas et al. explore Bayesian and CUSUM-based change-point detection approaches that can operate efficiently on edge devices without complex model training [18]. Classical ML techniques such as isolation-based and clustering-based detectors are also attractive due to their favorable effectiveness–efficiency trade-offs [83].

Another approach focuses on improving the efficiency of training and data processing pipelines. Tuli et al. propose a generative anomaly detection framework that **reduces the amount of training data required** while maintaining detection performance, thereby lowering memory and storage overhead during training [16]. Such techniques are particularly useful in edge environments where storing large volumes of telemetry locally may be impractical.

Techniques developed for large-scale cloud monitoring systems also offer useful insights. In high-dimensional monitoring environments, some studies employ **sampling or dimensionality reduction** to reduce the volume of telemetry processed by anomaly detection models [99, 100]. These techniques help limit computational overhead while preserving detection capability in systems with large numbers of metrics. Similarly, **distributed anomaly detection frameworks** such as the Microsoft KDD'19 detection system [92] and Skyline [93] distribute detection tasks across multiple nodes, reducing centralized processing bottlenecks. Although designed primarily for cloud infrastructures, their architectural principles align well with decentralized edge deployments.

Finally, research on lightweight ML for IoT and embedded systems provides additional techniques that can be adapted for edge anomaly detection [95–97]. These include **model compression approaches** such as pruning, quantization, and knowledge distillation, as well as TinyML techniques that enable inference on low-power microcontrollers. Such methods aim to reduce model size, memory footprint, and energy consumption, thereby enabling deployment on resource-constrained devices. Despite their potential relevance, these techniques remain largely underexplored in existing edge anomaly detection studies.

2.6.9 Research Gaps and Opportunities

The review above highlights several important research gaps in performance anomaly detection for microservices-based IoT applications deployed across the edge–cloud continuum.

- **Training efficiency as a first-class design objective.** While many studies evaluate inference-time efficiency, comparatively fewer explicitly optimize the efficiency of model training. This limitation is particularly important in edge environments where models may require frequent retraining to adapt to workload drift, device heterogeneity, and evolving application behavior. Developing training strategies that jointly optimize detection accuracy, training time, and resource consumption remains an open challenge.

- **Limited holistic multi-source anomaly detection.** Most existing edge AD approaches rely primarily on metrics, while logs and traces are either ignored or processed separately. However, performance anomalies in microservice systems often manifest across multiple observability modalities. Efficient techniques for jointly analyzing metrics, logs, and traces under edge resource constraints remain largely unexplored.

- **Underexplored learning paradigms under limited labels.** Although semi-supervised, self-supervised, and weakly supervised learning paradigms have gained traction in cloud anomaly detection research, their adoption in edge environments remains limited. Investigating lightweight variants of these paradigms that can operate under limited labels and constrained resources represents an important research opportunity.

- **Lack of edge-aware algorithm design.** Many current approaches adapt anomaly detection techniques originally developed for cloud environments. However, algorithms designed explicitly for edge settings, such as resource-adaptive models, compression-first architectures, or decentralized learning strategies, remain relatively scarce.

- **Limited adaptation of foundation-model techniques to edge environments.** Transformer-based models and large foundation models are increasingly being explored for cloud anomaly detection. However, practical pathways for deploying such models in edge environments, such as model distillation, hierarchical inference, or edge–cloud collaborative architectures, remain largely unexplored.

- **Opportunities for composable efficiency mechanisms.** Existing efficiency techniques such as dimensionality reduction, lightweight models, online learning, and distributed inference are often studied in isolation. Designing anomaly detection pipelines that systematically combine multiple efficiency mechanisms to meet strict edge constraints represents a promising research direction.

Overall, addressing these challenges requires rethinking anomaly detection algorithms and system architectures with edge constraints as a primary design consideration rather than an afterthought.

While anomaly detection plays a crucial role in identifying abnormal system behavior, it does not explain why such anomalies occur. In complex microservices-based IoT applications, performance degradation often propagates across multiple services and infrastructure components, making manual diagnosis difficult. Consequently, anomaly detection must be complemented by techniques that identify the root causes of observed anomalies. The next section therefore reviews existing research on RCL and analysis in edge environments and examines how diagnostic techniques can operate under similar scalability and resource constraints.

2.7 Root Cause Localization and Analysis in Edge Environments

While anomaly detection identifies deviations from normal system behavior, it does not explain the underlying cause of those deviations. In distributed microservice-based systems, anomalies often propagate across multiple components due to service dependencies and shared infrastructure resources. Consequently, determining the true source of performance degradation requires additional diagnostic techniques capable of identifying both the location and the cause of anomalies.

Root cause diagnosis in microservice environments is commonly discussed under the umbrella of Root Cause Analysis (RCA). Within this process, Root Cause Localization (RCL) identifies the component responsible for the anomaly, while deeper RCA techniques attempt to explain why the anomaly occurred. These diagnostic tasks are typically triggered after anomalies are detected in monitored telemetry streams. In many cloud environments where web applications dominate, anomaly detection is commonly applied to response time metrics of user-facing services [15, 101–103]. When abnormal latency is detected in such metrics, the diagnosis process is initiated to identify the faulty microservice.

However, this triggering mechanism does not translate directly to edge computing environments. Many edge applications operate through streaming pipelines or publish–subscribe architectures that do not expose explicit user-facing response time metrics. Instead, anomalies may manifest across multiple infrastructure layers, including microservices, containers, edge devices, and network links. Therefore, effective RCL in edge environments requires anomaly triggers that span multiple monitoring layers rather than relying solely on frontend latency metrics.

To systematically analyze existing approaches, this section introduces a taxonomy for RCL and RCA techniques applicable to microservice-based IoT applications deployed across the edge–cloud continuum. However, the body of work that explicitly targets RCL and analysis in edge computing environments remains limited. To date, only a small number of studies explicitly consider edge-specific constraints during diagnosis [27, 52, 87, 104]. Consequently, this review draws not only on the limited edge-focused literature but also on the broader body of cloud and microservice RCA research. This is appropriate because many diagnostic principles developed for cloud microservice systems, such as dependency modeling, anomaly propagation analysis, and graph-based localization, remain relevant to edge environments, even though they must be revisited in light of edge-specific constraints including resource limitations, network instability, and decentralized deployment.

2.7.1 Taxonomy Overview

Figure 2.5 presents the taxonomy used in this thesis to organize existing RCL and analysis work for microservices-based IoT applications in edge environments. The taxonomy spans several dimensions that capture how diagnosis techniques operate in distributed microservice environments: *(i) RCA granularity*, *(ii) observability modality*, *(iii) dependency modeling strategy*, *(iv) RCA reasoning strategy*, *(v) analysis architecture*, *(vi) evaluation cri-*

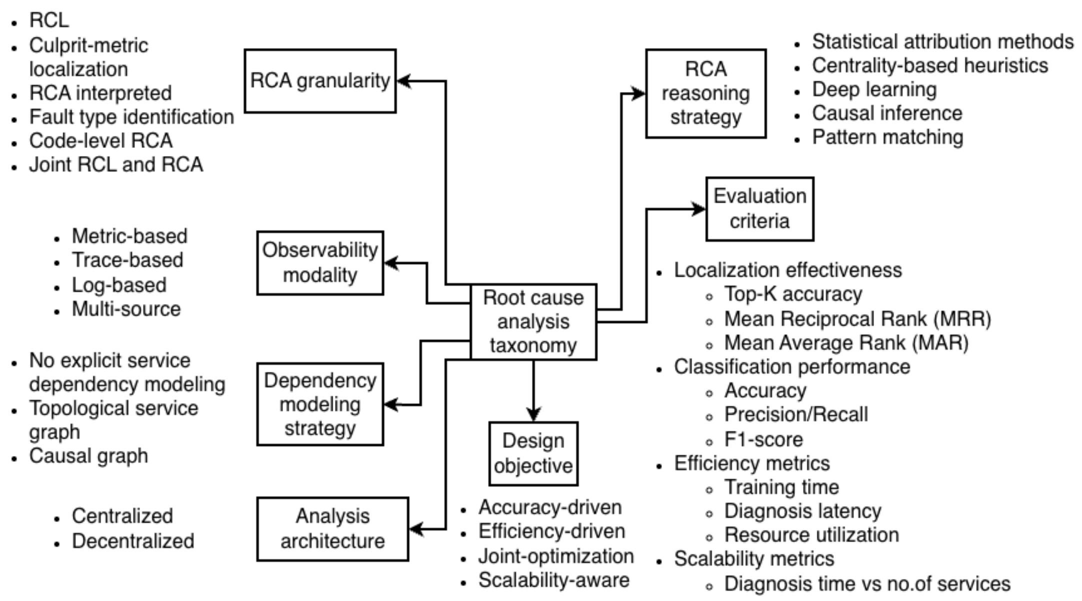


Figure 2.5: Root cause analysis taxonomy

teria, and *(vii) design objective*. These dimensions reflect the key design choices that influence the effectiveness and practicality of RCA solutions in distributed edge environments. The following subsections examine each dimension and position representative studies accordingly.

2.7.2 RCA Granularity

RCA in microservices-based systems spans a spectrum of diagnostic granularity, ranging from coarse-grained localization to fine-grained fault identification.

At the coarsest level, **root cause localization (RCL)** aims to identify the microservice responsible for an observed anomaly. RCL is generally considered the first and most fundamental step of RCA and is performed, either explicitly or implicitly, by nearly all diagnostic approaches [25, 105]. Several studies focus exclusively on this task, including MicroRCA [101], MicroGBPM [15], MicroCERCL [52], and MicroEGRCL [106]. While microservice-level localization can often be achieved quickly, identifying only the faulty service provides limited diagnostic insight because the underlying cause of the anomaly remains unknown. Consequently, Site Reliability Engineering (SRE)s must manually investigate potential issues such as CPU contention, memory exhaustion, or abnormal request patterns [107, 108].

Beyond microservice-level localization, many RCA techniques incorporate finer-grained analysis. One intermediate diagnostic level is **culprit metric localization**, which identifies the specific performance metrics responsible for the anomaly, such as abnormal CPU usage, memory consumption, or request latency. Representative approaches include MicroDiag [103], AAMR [102], HeMIRCA [108], CausalRCA [105], and PDiagnose [109]. Identifying the culprit metric enables more targeted mitigation actions, such as resource scaling or scheduling adjustments, which can often restore performance more efficiently than restarting entire services.

Some approaches further enhance diagnosis by incorporating **interpretability**. For example, Grace [110] performs RCL while highlighting influential service dependencies and relevant time-series metrics that contributed to the anomaly. Such contextual explanations support human-in-the-loop diagnosis and facilitate deeper fault investigation. The edge-specific CIRCA framework by Jiang et al. [87] also falls into this category.

At a more advanced level, RCA techniques perform **fault type identification**. These approaches classify anomalies into predefined categories such as CPU hogging, memory leaks, network delay, or I/O contention. Methods such as DiagFusion [30], MicroIRC [29], DejaVu [28], Brandon et al. [111], Kalinagac et al.[27], and MicroHECL [53] adopt this strategy. Fault-type identification provides semantically meaningful explanations of anomalies and enables more automated mitigation actions.

Finally, the most fine-grained level of RCA operates at the **source-code level**. Approaches such as Nezha [112] attempt to localize performance issues to specific code segments. While valuable for long-term fixes, such methods typically require extensive instrumentation and introduce nontrivial overhead, making them less suitable for real-time diagnosis in production environments.

Although RCL and RCA are often discussed as separate tasks, they are inherently interconnected: both are triggered by the same anomalous event and rely on similar signals, including temporal performance patterns and service dependencies. Consequently, recent studies increasingly perform **RCL and RCA jointly** within a unified model rather than through sequential pipelines. Representative examples include MicroDiag [103], DiagFusion [30], MicroIRC [29], and DejaVu [28]. By sharing feature extraction over temporal and dependency information, such joint approaches can improve both diagnostic efficiency and overall accuracy.

Joint RCL and fault-type identification is particularly valuable for autonomous Artificial Intelligence for IT Operations (AIOps) systems because it produces *actionable diagnosis* [28]. Effective remediation requires knowing both where the failure occurs and what type of failure has occurred. Together, this pair of information elements forms a *failure unit* [28]. By identifying both the faulty component and the fault type, diagnosis results can directly guide mitigation actions such as scaling resources, migrating services, or adjusting scheduling policies.

2.7.3 Observability Modality

RCA techniques leverage various telemetry sources to diagnose performance anomalies. Based on the primary data modality analyzed, existing approaches can be categorized into metric-based, trace-based, log-based, and multi-source approaches [23–25].

Metric-based RCA analyzes quantifiable system indicators such as CPU utilization, memory usage, latency, and request throughput [27]. Because metrics provide continuous time-series signals and structured data representations, they can be processed efficiently and are well suited for automated analysis.

Log-based RCA relies on textual records generated by applications, operating systems, and network components. Logs provide rich contextual information, including error messages and warnings, but they are often voluminous and unstructured, making real-time analysis challenging.

Trace-based approaches analyze distributed request traces that capture service invocation paths. While traces are valuable for understanding service dependencies and request flows, they can be expensive to collect and process, particularly in large-scale distributed systems [108]. Additionally, traces are often too coarse-grained to capture internal resource bottlenecks within services.

Multi-source approaches combine metrics, logs, and traces to obtain a more comprehensive view of system behavior [52]. However, integrating heterogeneous data sources introduces challenges related to data alignment, synchronization, and feature extraction [25, 112].

Overall, metric-based approaches offer a practical balance between diagnostic capability and computational efficiency, making them particularly suitable for edge environments.

2.7.4 Dependency Modeling Strategy

The effectiveness of root cause diagnosis largely depends on how service dependencies and anomaly propagation are modeled.

Early RCA methods **do not explicitly model service dependencies** and instead rely on statistical relationships between performance metrics. Such approaches identify root causes by computing correlations between anomalous response time metrics and other service metrics [53, 113]. Although lightweight, these correlation-based methods often fail to capture indirect dependencies and propagation paths across microservices.

More recent approaches construct **topological service graphs** representing structural relationships among services [52, 75, 87]. These graphs are derived from deployment configurations, communication patterns, or distributed traces. Topological graphs enable anomaly propagation analysis by allowing diagnostic algorithms to propagate anomaly scores through service dependency networks.

A more advanced strategy involves constructing **causal graphs**, where edges represent cause–effect relationships between services or metrics [27, 104]. Causal graphs provide stronger interpretability but are difficult to maintain in dynamic microservice environments because causal structures may change with workload fluctuations and deployment updates.

2.7.5 RCA Reasoning Strategy

Once dependencies are modeled, RCA techniques differ in how they reason over available telemetry and graph structures.

Statistical attribution methods identify root causes by computing statistical associations between anomalous metrics and other system signals [102, 108]. These methods are computationally lightweight but often struggle with complex propagation paths.

Centrality-based heuristics analyze graph structures to identify influential nodes in anomaly propagation networks. Techniques such as degree centrality, betweenness centrality, eigenvector centrality, and PPR are commonly used. Among these, PPR has demonstrated strong performance in microservice RCL [15, 75, 101]. These methods are lightweight, interpretable, and do not require labeled training data. This makes them highly suitable for resource-constrained edge environments. However, because they do not leverage historical failure patterns extensively, they may not capture complex temporal patterns.

Deep learning (DL) approaches, particularly GNNs, represent the state of the art for joint localization and fault classification due to their ability to model complex inter-service dependencies [110]. Methods such as DiagFusion [30], MicroIRC [29], Micro-CERCL [52], and DejaVu [28] use GNNs to learn representations over service dependency graphs. However, it is important to note that GNNs are computationally expensive [52].

Causal inference approaches explicitly model cause–effect relationships to identify root causes [27, 103, 105]. These methods attempt to distinguish correlation from causation, often using probabilistic graphical models or structural causal models. CI methods improve interpretability and theoretical soundness but require strong assumptions and substantial historical data. Their computational overhead can limit applicability in latency-sensitive edge environments. LLMs-DCGRCA [104] is an edge-specific framework that integrates LLMs with causal inference to generate knowledge-assisted causal hypotheses.

Pattern-matching approaches diagnose failures by matching observed anomalous subgraphs against predefined fault templates. For example, Brandon et al. [111] detect faults by comparing anomaly propagation structures to a library of known failure patterns. Similarly, ST-GraphRCA [74] integrates pattern matching with causal inference to enable diagnosis in resource-constrained IoT environments. Pattern matching is interpretable and efficient when fault libraries are well-defined. However, it struggles to generalize to unseen fault types and requires continuous maintenance of fault pattern repositories.

2.7.6 Analysis Architecture

Most existing RCA approaches assume **centralized** analysis architectures in which telemetry collected from distributed monitoring agents is transmitted to a central location for diagnosis [27, 52]. While suitable for cloud environments, centralized analysis introduces significant communication overhead in edge computing environments where bandwidth may be limited and network connectivity is unstable.

To address these challenges, diagnosis mechanisms should ideally operate closer to the edge devices generating telemetry. Deploying **lightweight RCL models at edge nodes** can significantly reduce data transfer latency and enable faster anomaly diagnosis. However, this design requires diagnostic models to be computationally efficient and resource-aware.

2.7.7 Evaluation Criteria

Evaluation of RCA techniques typically considers four main dimensions: localization effectiveness, classification performance, efficiency, and scalability. While cloud-oriented studies have traditionally emphasized accuracy, edge environments require a more holistic evaluation framework due to resource constraints and system scale.

For RCL tasks, evaluation primarily focuses on **localization effectiveness**. This is commonly formulated as a ranking problem using metrics such as Top-K accuracy, Mean Reciprocal Rank (MRR), and Mean Average Rank (MAR) [27, 52, 87, 104]. These metrics measure how effectively the true root cause is prioritized within a ranked list of candidate services.

For approaches that perform fault type identification, evaluation focuses on **classification performance**. In this case, the problem is typically framed as a classification task, using metrics such as accuracy, precision, recall, and F1-score [27]. Some works also introduce specialized metrics to better capture diagnostic performance across diverse fault categories. For example, ST-GraphRCA [74] evaluates model robustness using Mean Average Recall (MAR) across different failure types, such as CPU exhaustion and network latency. Similarly, the GRALAF framework proposed by Kalinagac et al. [27] introduces a Fault Type Recall metric to measure how effectively anomalies are categorized into traffic, performance, or reliability faults. While accuracy is commonly reported, precision and recall are particularly important in operational systems, where false positives may trigger unnecessary mitigation actions and false negatives may delay system recovery.

Efficiency is increasingly critical, especially in edge environments where computational and energy resources are limited [25]. Key efficiency metrics include training time, inference time, and resource utilization (CPU, memory, and network overhead). Most existing edge-oriented RCA studies explicitly evaluate these efficiency aspects to demonstrate the feasibility of their approaches under resource-constrained conditions [52, 74, 87]. In edge environments, minimizing diagnosis latency is particularly important because delayed diagnosis directly delays mitigation actions.

Scalability evaluates how diagnostic performance changes as system size increases. A common scalability metric is Diagnosis time versus number of services (or graph size). Graph-based approaches must process increasingly large graphs in distributed edge environments. This complexity can significantly increase localization/diagnosis times [14, 26, 27]. In latency-sensitive edge systems, increased diagnosis time directly delays mitigation actions. Therefore, scalability evaluation is essential for determining whether an RCA technique remains practical in large-scale, distributed environments.

2.7.8 Design Objective

Beyond how RCA techniques are evaluated, it is equally important to distinguish their primary design objectives. Existing works can be categorized into four design orientations: accuracy-driven, efficiency-driven, joint optimization, and scalability-aware approaches.

The majority of cloud RCL /RCA studies follow an **accuracy-driven** design orientation, prioritizing improvements in localization or classification accuracy [14, 15, 29, 89, 103, 105, 107, 108]. These works often employ sophisticated graph modeling, DL, or causal reasoning to maximize diagnostic precision, with evaluation focusing primarily on this accuracy dimension.

In contrast, relatively few studies adopt an explicitly **efficiency-driven** design orientation. Although some cloud RCA approaches report faster inference times (e.g., AAMR, MicroEGRCL, Grace), efficiency is rarely treated as a primary design objective. Even edge-oriented solutions such as MicroCERCL and the framework by Kalinagac et al. do not explicitly prioritize efficiency, and evaluations reveal higher inference latency compared to lightweight heuristic approaches due to graph complexity [52].

MicroHECL [53] and PDiagnose [109] are cloud RCL /RCA techniques that specifically aim to improve efficiency, particularly by providing faster localization speeds. Both approaches eliminate the need for training. MicroHECL achieves efficiency by efficiently traversing the service dependency graph and using pruning techniques to eliminate irrelevant service calls during anomaly propagation chain analysis, which further enhances efficiency. On the other hand, PDiagnose tries to reach efficiency by removing the computationally heavy dependency graph-building phase and utilizing a vote-based localization process on an anomaly queue.

Joint optimization approaches attempt to balance accuracy and efficiency simultaneously. Rather than maximizing one dimension, these methods seek an acceptable trade-off between diagnostic performance and computational overhead. This design objective is particularly relevant in edge computing, where neither pure accuracy nor pure efficiency alone is sufficient.

Scalability-aware designs explicitly consider system growth as a primary constraint. These methods aim to maintain near-constant or sub-linear diagnosis latency as the number of services increases. In large-scale edge environments, scalability becomes as critical as accuracy, since delayed diagnosis can negate the benefits of precise localization. Kalinagac et al. [27] attempt to address scalability through metric filtering; however, this approach exhibits limited effectiveness beyond moderate system sizes (approximately 35 nodes), highlighting the need for more robust scalability-oriented solutions.

2.7.9 Research Gaps and Opportunities

The review above highlights several important research challenges in RCL and analysis for edge–cloud environments.

- **Multi-level anomaly triggering mechanisms.** Many existing RCA approaches rely on anomalies in user-facing response times as triggers. Edge environments require multi-layer anomaly triggers spanning microservices, devices, and network infrastructure.
- **Lack of edge-native and decentralized RCA approaches.** Most existing methods assume centralized analysis over full-system graphs. Designing decentralized diagnosis mechanisms that operate closer to edge devices remains an open challenge.
- **Need for lightweight diagnostic architectures.** Resource constraints at edge nodes require computationally efficient RCA models capable of operating with limited compute and memory resources.
- **Balancing diagnostic effectiveness and efficiency.** Most existing research prioritizes accuracy over efficiency. Edge environments require diagnostic techniques that explicitly balance these objectives.
- **Scalability challenges in large service graphs.** Graph-based RCA approaches may suffer from increased latency as system size grows. Developing search-space reduction techniques and scalable graph processing methods is essential.
- **Integration of complementary efficiency techniques.** Efficiency techniques such as graph pruning and lightweight inference are often studied independently. There is limited work on integrating these complementary strategies into unified, efficiency-aware RCA frameworks.

Addressing these challenges requires rethinking RCL and analysis techniques with edge constraints as a primary design consideration.

2.8 Operationalizing Mitigation in Edge Environments

While section 2.3 reviewed mitigation mechanisms available for handling performance anomalies, this section discusses how such mechanisms are operationalized in practice within microservice-based edge–cloud environments.

In most existing anomaly-aware edge studies, monitoring and diagnosis components operate independently from the mitigation control plane [11, 64, 114]. Once an anomaly is detected and diagnosed, mitigation actions are typically executed by external orchestration mechanisms rather than by the anomaly management system itself. For example, anomaly detection systems may trigger alerts that subsequently activate autoscaling policies, container restarts, or workload migration through orchestration controllers. Container orchestration platforms such as Kubernetes implement such control mechanisms through controllers responsible for autoscaling, self-healing, and workload scheduling [54, 59].

This separation between anomaly management and mitigation is particularly evident in edge computing environments. In many edge deployments, mitigation decisions are executed through **centralized orchestration control planes**, which maintain a global view of system state and coordinate resource allocation across edge nodes and cloud infrastructure [51, 114, 115]. While centralized control simplifies coordination and policy enforcement, it introduces challenges in geographically distributed environments, including increased communication latency, and potential single points of failure.

To address these limitations, several studies have explored the potential for **decentralized mitigation mechanisms**, which move mitigation decisions closer to the edge devices where anomalies occur. By enabling local control loops, such approaches can reduce mitigation latency and limit cross-device communication. For instance, Park et al. [116] proposes a decentralized resource management strategy that allows edge nodes to locally adjust resource allocations in response to burst-load scenarios. Despite these conceptual proposals, decentralized mitigation mechanisms remain relatively underexplored in practical large-scale deployments.

Nevertheless, **partial decentralization** can be achieved through multi-cluster or federated edge deployments. In such architectures, mitigation decisions can be scoped to individual clusters or regional edge domains rather than being coordinated across the entire edge–cloud continuum. In such hierarchical management structures, local control planes perform mitigation actions within their respective clusters while higher-level controllers manage global policies [117].

Overall, mitigation in current edge environments is typically operationalized through orchestration control loops, with anomaly detection and diagnosis systems providing input signals rather than directly executing remediation actions. While decentralized mitigation offers potential advantages in terms of latency and scalability, its practical deployment remains limited, highlighting opportunities for tighter integration between anomaly diagnosis and resource management mechanisms in distributed edge infrastructures.

Table 2.3: Classification of representative monitoring and anomaly-aware studies for edge–cloud environments under the proposed taxonomies.

Work	**Mon. Data**	**Mon. Arch**	**AD Mod.**	**Learn.**	**AD Model**	**Lightwt.**	**RCA Gran.**	**Dep. Model**	**RCA Reason.**	**Mitig.**	**Eval.**
[63]	M+L	Cent	Uni-M	N/A	Rule-based	LMon	–	Topo	–	–	Real
[64]	M	Hyb	Uni-M	N/A	Rule-based	–	–	Topo	–	Ext-Orch	Real
[61]	M	Dec	Uni-M	N/A	Rule-based	LMon	–	Topo	–	–	Real
[118]	M	Dec	Uni-M	N/A	Rule-based	LMon	–	–	–	–	Real
[65]	M	Dec	Uni-M	N/A	Rule-based	LMon	–	–	–	–	Real
[66]	M	Dec	Uni-M	N/A	Rule-based	LMon	–	–	–	–	Real
[12]	M	Cent	Multi-M	Sup	CML	–	–	–	–	–	Real
[16]	M	Cent	Multi-M	Unsup	DL	Mem-Eff	–	–	–	–	Real + Cloud Bench
[18]	M	Cent	Uni-M	Unsup	Stat	ILAlg	–	–	–	–	Real
[70]	M	Cent	Multi-M	Unsup	DL	–	–	–	–	–	Sim
[27]	M	Cent	Uni-M	Unsup	CML	–	FTI	Causal	CI	–	Emu
[75]	MS	Cent	Multi-M	Unsup	CML	ILAlg	RCL	Topo	Stat/Heur	–	Emu
[52]	M+L	Cent	MS	Unsup	CML	–	RCL	Topo	DL	–	Bench
[119]	MS	Cent	MS	Varies	Varies	–	RCL + CMLoc	Varies	Varies	–	Real

Work	Mon. Data	Mon. Arch	AD Mod.	Learn.	AD Model	Lightwt.	RCA Gran.	Dep. Model	RCA Reason.	Mitig.	Eval.
[104]	MS	Cent	MS	–	–	–	CMLoc	Causal	CI + LLM	–	Real + Cloud Bench
[87]	MS	Cent	MS	Sup	DL	ILAlg	RCL inter-preted	Topo	DL	–	Sim + Cloud Bench
[74]	MS	Cent	Multi-M	Unsup	CML	–	Joint	ST-Graph	CI + Pattern	–	Real
[11]	M	Hyb	Multi-M	Unsup	CML	ILAlg	RCL + FTI	Topo	Stat/Heur	Ext-Orch	Real
[120]	M	Cent	Multi-M	Mixed	DL	Dist	RCL + FTI	None	Pattern	Ext-Orch	–
[115]	M	Cent	Multi-M	N/A	Stat	Res-Aware	–	–	–	Ext-Orch	Real + Emu
[114]	M	Hyb	Uni-M	Unsup	CML/DL	Dist	–	–	–	Ext-Orch	Emu
[116]	M	Dec	Uni-M	N/A	Rule-based	Dist	–	–	–	Dec-Mit	Sim
[51]	M	Cent	Uni-M	N/A	Rule-based	–	RCL + FTI	None	Pattern	Ext-Orch	Real
[121]	M	Cent	Uni-M	N/A	Rule-based	–	Joint	Topo	LLM	Ext-Orch	Emu

Column Headings: Mon. Data = Monitoring Data Type; Mon. Arch = Monitoring Architecture; AD Mod. = Anomaly Detection Modality; Learn. = Learning Paradigm; AD Model = Anomaly Detection Model Type; Lightwt. = Lightweight Mechanism; RCA Gran. = RCA Granularity; Dep. Model = Dependency Modeling Strategy; RCA Reason. = RCA Reasoning Approach; Mitig. = Mitigation Mechanism; Eval. = Evaluation Environment. **Abbreviations (Cell Values):** M = metrics; L = logs; MS = multi-source; Cent = centralized; Dec = decentralized; Hyb = hybrid; Uni-M = univariate metric-based; Multi-M = multivariate metric-based; Sup = supervised; Unsup = unsupervised; Mixed = mixed learning settings; Stat = statistical/conventional methods; CML = classical machine learning; DL = deep learning; LMon = lightweight monitoring; Mem-Eff = memory-efficient; ILAlg = inherently lightweight algorithm; Dist = distributed framework/mechanism;

Res-Aware = resource-aware; RCL = root cause localization only; RCA = root cause analysis; CMLoc = culprit metric localization; FTI = fault type identification; Joint = joint RCL and RCA; None = no explicit dependency modeling; Topo = topological service/dependency graph; Causal = causal graph; ST-Graph = spatio-temporal graph; Stat/Heur = statistical or centrality/heuristic reasoning; Pattern = pattern matching; CI = causal inference; LLM = LLM-assisted reasoning/modeling; Ext-Orch = external orchestration mechanism; Dec-Mit = decentralized mitigation; Real = real deployment/testbed; Sim = simulator; Emu = emulator; Bench = benchmark/dataset-based evaluation; N/A = Not applicable

2.9 Edge Experimental Environments and Evaluation Platforms

Evaluating anomaly detection and diagnosis techniques for edge computing environments remains challenging due to the limited availability of suitable experimental platforms and datasets. In many existing anomaly-aware edge studies, evaluation is performed using either cloud-based datasets or privately constructed edge testbeds. Progress in this area is constrained by two key factors: the scarcity of publicly available edge-specific performance anomaly datasets and the limited accessibility of real-world edge infrastructures for controlled experimentation.

Several studies evaluate their methods using **datasets originally collected from cloud environments**. For example, Tuli et al. [16] evaluate their approach using the Server Machine Dataset (SMD) [72] and the Multi-Source Distributed System (MSDS) dataset [122]. While these datasets are valuable for benchmarking anomaly detection algorithms, they do not capture several defining characteristics of edge systems, including heterogeneous hardware resources, dynamic microservice placement across devices, fluctuating network conditions, and diverse QoS requirements of IoT applications. Additionally, recent research evaluates RCA frameworks using public cloud-native benchmarks such as the Generic AIOps Atlas (GAIA) [123] for metric-based RCA [104], while models such as CIRCA [87] frequently rely on public microservice and testbed datasets, including MicroSS [124], Swat [125], and WADI [126], to validate the effectiveness of their identification and path inference capabilities. Consequently, there is currently no widely adopted publicly available dataset specifically designed to represent performance anomalies in edge computing environments. Furthermore, publicly available normal edge traces are also scarce, limiting the ability of researchers to construct realistic edge datasets through synthetic anomaly injection.

A few recent studies have begun to move toward more representative edge-oriented datasets. For example, the MicroCERCL benchmark introduced by Zhu et al. [52] deploys a hybrid microservice system in a cloud–edge collaborative environment and evaluates RCL using telemetry collected from this deployment. Nevertheless, the availability of such edge-oriented datasets and benchmarks remains limited, indicating substantial room for developing more diverse and scalable datasets tailored for anomaly detection and diagnosis in edge computing environments.

Another common evaluation approach is experimentation on **privately constructed edge testbeds**. Studies such as Becker et al. [11], Soualhia et al. [12], and Skaperas et al. [18] evaluate their techniques using private edge deployments. Similarly, Tuli et al. [16] perform additional experiments using datasets collected from their own edge testbed. Wang et al. [70] evaluate their GRU-autoencoder-based anomaly detection method using a simulated vehicular fog computing environment with synthetically generated reliability data, further illustrating the reliance on privately constructed datasets in edge AD research. In the context of RCA, several recent studies, including Nguyen et al. [119], Su et al. [74], and Tang et al. [104], conduct evaluations on real edge computing setups to validate their diagnostic methods under realistic deployment conditions. Although such environments provide more realistic experimental conditions than cloud datasets, they introduce significant reproducibility challenges. Most private testbeds are not publicly accessible and often lack detailed documentation regarding hardware configurations, network topology, workload characteristics, deployment strategies, and anomaly injection procedures. As a result, it becomes difficult for other researchers to replicate the experimental setup, regenerate comparable datasets, or independently validate reported results.

Access to real edge infrastructures is also inherently limited. Building and maintaining an edge–cloud deployment requires substantial investment in heterogeneous hardware platforms, networking infrastructure, and management frameworks. Consequently, only a small number of research groups are able to conduct experiments under realistic edge conditions. Moreover, controlled anomaly injection in production-like environments can be risky. Introducing resource contention, workload spikes, or network disruptions may compromise system stability or interfere with shared services, making systematic and repeatable anomaly experimentation difficult.

Some researchers attempt to address accessibility challenges by constructing smaller-scale edge testbeds using devices such as Raspberry Pi or NVIDIA Jetson boards. These platforms better reflect the resource constraints and heterogeneity of edge environments compared to cloud servers. However, such setups typically suffer from limited scalability and availability. Reproducing large service graphs, complex microservice dependencies, and realistic cross-device interactions remains difficult in small-scale deployments.

Given the challenges of conducting experiments on real edge infrastructures, researchers often rely on two alternative approaches: simulation and emulation.

Simulators are widely used in edge computing research to model large-scale distributed environments under controlled and configurable conditions [87, 127, 128]. They enable systematic experimentation with different system sizes, placement strategies, workload intensities, and network configurations. As a result, simulation is particularly useful for evaluating scalability-oriented solutions such as resource management, service placement, and communication overhead in distributed architectures. However, simulation inevitably introduces abstraction layers that limit realism. Many edge simulators do not support real IoT protocols and services [129], and often rely on simplified assumptions regarding device heterogeneity, workload behavior, and network dynamics. Prior studies highlight that such simplifications may diverge significantly from real-world edge deployments [130]. Consequently, simulation-based results may suffer from limited external validity. Additionally, most existing simulators emphasize specific aspects of edge systems—such as placement or scheduling—rather than accurately modeling infrastructure-level dynamics such as packet-level networking, resource contention, or realistic anomaly behavior [131]. Nevertheless, simulation remains an important tool for evaluating scalability, particularly for techniques such as RCL that must operate on large service graphs.

Emulators, in contrast, execute real applications on testbed hardware while emulating large-scale network and infrastructure conditions [27, 75, 131]. Compared to simulators, emulators provide a higher degree of realism because they incorporate actual application behavior and runtime systems. At the same time, they remain more accessible and cost-effective than full-scale edge deployments. Several edge emulation platforms have been proposed, including EmuFog [130], FogBed [129], MockFog [132], EmuEdge [133], Fogify [134], and iContinuum [131]. These platforms primarily aim to reproduce realistic compute and network conditions to enable testing of IoT applications before production deployment.

Although some modern emulators support fault injection mechanisms, these capabilities are typically designed to evaluate system reliability or fault tolerance rather than to systematically generate labeled performance anomaly datasets. Consequently, current edge emulation platforms do not directly support the generation of large-scale, reproducible performance anomaly datasets suitable for training and evaluating anomaly detection and diagnosis models.

2.9.1 Research Gaps and Opportunities

The discussion above highlights several important research gaps in the experimental evaluation of anomaly management techniques for edge environments.

- **Lack of publicly available edge performance anomaly datasets.** There is currently no publicly available dataset specifically designed to represent performance anomalies in microservices-based edge computing environments. An ideal dataset should include both normal and anomalous performance data collected from heterogeneous edge infrastructures, incorporate diverse IoT applications composed of interacting microservices, provide multi-source observability data (metrics, logs, and traces), and support realistic and reproducible anomaly injection mechanisms.
- **Limited reproducibility of experimental environments.** Many existing studies rely on private edge testbeds, which restricts reproducibility and slows cumulative research progress. Releasing both datasets and the associated dataset-generation frameworks would significantly improve transparency and enable independent validation of results.
- **Need for scalable evaluation platforms.** Real edge deployments and small-scale

testbeds make it difficult to evaluate scalability properties of anomaly detection and RCA techniques. High-fidelity simulators capable of realistically modeling heterogeneous edge infrastructures and large service graphs would facilitate systematic scalability evaluation.

- **Need for emulator-based dataset generation frameworks.** Although existing emulators provide realistic edge experimentation environments, they are not designed for systematic performance anomaly dataset generation. Extending edge emulators with capabilities such as comprehensive monitoring, protocol-diverse workload generation, and controlled chaos-engineering-based anomaly injection represents a promising direction for future research.

2.10 Summary

This chapter reviewed existing research on anomaly detection and diagnosis in microservice-based edge environments through structured taxonomies covering modeling approaches, system architectures, observability data sources, and evaluation environments. While prior work has made significant progress in improving diagnostic accuracy, important challenges remain, particularly in addressing scalability and the unique constraints of edge computing environments. The following chapters address these limitations by proposing new approaches for dataset generation, efficient anomaly detection model training, decentralized RCL, and scalable architectures for joint diagnosis tailored to edge–cloud systems.

Chapter 3

iAnomaly: A Toolkit for Generating Performance Anomaly Datasets in Edge-Cloud Integrated Computing Environments

Progress in performance anomaly detection and diagnosis for edge computing environments is limited primarily due to the absence of publicly available edge performance anomaly datasets or due to the lack of accessibility of real edge setups to generate necessary data. To address this gap, this chapter proposes iAnomaly: a full-system emulator equipped with open-source tools and fully automated dataset generation capabilities to generate labeled normal and anomaly data based on user-defined configurations. We also release a performance anomaly dataset generated using iAnomaly, which captures performance data for several microservice-based IoT applications with heterogeneous QoS and resource requirements while introducing a variety of anomalies. Detailed analysis demonstrates that this dataset effectively represents the characteristics found in real edge environments, and the anomalous data in the dataset adheres to the required standards of a high-quality performance anomaly dataset. Additionally, to support scalability experiments that require modeling thousands of computational entities, we extend iAnomaly with a trace-driven dataset generation simulator that produces large-scale service dependency graphs while preserving realism by populating them with normal and anomalous traces derived from the iAnomaly dataset.

This chapter is derived from:

- **Duneesha Fernando**, Maria A. Rodriguez, and Rajkumar Buyya, "iAnomaly: A Toolkit for Generating Performance Anomaly Datasets in Edge-Cloud Integrated Computing Environments", *Proceedings of the 17th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2024)*, Pages: 236–245, Sharjah, United Arab Emirates, December 16-19, 2024.

3.1 Introduction

Currently, there is limited research on performance anomaly detection in edge computing environments. One of the main reasons for this is the absence of publicly available edge performance anomaly datasets, which are crucial for training and evaluating algorithms proposed in such research. The few existing studies rely on cloud datasets [16, 17] or data collected from private edge setups [11, 12, 18] to evaluate their proposed approaches. The cloud datasets have been collected from applications (mostly web applications) deployed on cloud servers. However, these cloud servers lack the heterogeneity found in edge devices in terms of computing, storage, and networking capabilities. Additionally, the microservices in cloud applications do not demonstrate the same diversity in terms of QoS and resource requirements as those in an IoT application. As a result, cloud datasets fail to capture characteristics inherent to real edge environments. On the other hand, private edge setups have not been publicly released and lack detailed information, which makes it difficult to replicate their environments, generate the necessary data, and reproduce the results of the anomaly detection experiments. It also hinders research in the field because not everyone has access to a real edge-cloud deployment for data collection purposes. Hence, relying on cloud datasets and private edge setups does not facilitate performance anomaly detection research in edge computing environments, thus posing a challenge to the progression of the field. Therefore, there is an opportunity to create a performance anomaly dataset that reflects the characteristics of edge computing environments and release the setup used for dataset generation.

Edge computing emulators are a suitable platform to generate performance anomaly datasets. They are more representative of real edge environments when compared to simulators, and are more easily accessible and cost-effective when compared to real edge deployments. The main aim of existing edge computing emulators is to create a staging environment that achieves compute and network realism similar to a real edge environment and facilitate testing of IoT applications before deploying them into production [129–134]. However, these general-purpose emulators do not incorporate in their design, tools and mechanisms required to autonomously and transparently generate large-scale performance anomaly datasets useful for model training and evaluation. For example, they lack adequate monitoring tools to collect performance and system-level metrics, workload generation tools to generate and capture normal performance data, and chaos engineering mechanisms to inject performance anomalies into applications.

This work addresses this gap by presenting the iAnomaly framework, a performance anomaly-enabled full-system emulator that accurately models an edge computing environment hosting microservice-based IoT applications. iAnomaly is designed with open-source tools and provides fully automated dataset generation capabilities to generate labeled normal (data collected under normal conditions without anomalies) and anomaly (data collected under anomalous conditions) data based on user-defined configurations. In addition, we present a performance anomaly dataset generated using the proposed framework. The dataset captures performance data for several microservice-based IoT applications with heterogeneous QoS and resource requirements across a wide range of domains, software architectures, service composition patterns, and communication protocols by introducing a variety of client/sensor-side as well as server-side anomalies. To the best of our knowledge, this multivariate dataset is the first open-source edge performance anomaly dataset.

The analysis of the dataset showed that the microservices within it vary in terms of their QoS and resource usage during regular operation, thus successfully capturing the characteristics of a real edge dataset. Further analysis confirmed that the anomalous data in the dataset meets the necessary standards for a high-quality performance anomaly dataset. This includes having an anomaly ratio comparable to other standard anomaly datasets and the dataset's non-triviality.

Finally, while full-system emulation provides high realism, generating datasets that model thousands of computational entities for scalability experiments can be resource-intensive. To enable large-scale evaluations while maintaining realism, we extend iAnomaly with a dataset generation simulator that produces service dependency graphs ranging from 50 to 10,000 nodes. Instead of relying on purely synthetic signals, the simulator populates generated graphs with normal and anomalous traces derived from real execution data collected using the iAnomaly emulator, with randomized anomaly injection guided by predefined policies. This provides a practical mechanism to evaluate scalability without sacrificing the realism of anomaly and workload characteristics.

The rest of this chapter is organized as follows: Section 3.2 reviews the existing related works. Section 3.3 presents the architecture of the iAnomaly toolkit, while Section 3.4 discusses the implementation aspects of iAnomaly. Section 3.5 provides details of the generated performance anomaly dataset followed by an analysis of the dataset. Section 3.6 introduces the iAnomaly dataset generation simulator for producing large-scale service dependency graphs for scalability experiments. Section 3.7 concludes the chapter.

3.2 Related Work

Out of the existing research studies conducted around performance anomaly detection in edge computing environments, Becker et al. [11], Soualhia et al. [12], and Skaperas et al. [18] evaluated their proposed approaches using data collected from private edge setups. However, the lack of detailed information about these private edge setups makes it challenging to reproduce their environments and generate the necessary data to replicate the results of their anomaly detection experiments. Tuli et al. [16] evaluated their proposed approaches using two publicly available cloud datasets: the Server Machine Dataset (SMD) collected from a large Internet company [72], and the Multi-source Distributed System (MSDS) dataset generated from microservices deployed on a cluster of bare metal nodes with homogeneous computing, storage, and network capabilities [122]. As a result, these cloud datasets are unable to accurately represent the properties inherent to real edge environments. Tuli et al. also conducted a further evaluation on three self-created datasets collected from a private edge setup. However, similar to the rest of the literature, they also do not provide sufficient details required for reproducing the data generation environments. Therefore, the reliance on cloud datasets and private edge setups presents a challenge to the progression of the field of performance anomaly detection research in edge computing environments. Additionally, there is a lack of publicly available normal traces from real edge environments into which anomalies can be injected to generate synthetic datasets. We further explore this gap in current research by creating and releasing a performance anomaly dataset that reflects the characteristics of edge computing environments along with the setup used for dataset generation.

There are three options of platforms for generating a performance anomaly dataset.

They are simulators, emulators, and real edge environments. Out of these, emulators employ real applications deployed on testbed hardware to emulate real-world infrastructure configurations [131], while simulators do not support real-world IoT protocols and services [129]. Simulations make a number of simplifications that may not always hold true, especially with an infrastructure as dynamic as edge computing [130]. Most simulators lack detailed network simulation capabilities and focus on specific aspects of edge modeling, such as service scheduling [131]. Therefore, we identify emulators as the most suitable platform for generating data as they provide a higher Degree of Realism (DoR) than simulators and because they enable the generation of large-scale data in a cost-effective manner as opposed to real edge environments.

Edge emulator / toolkit	**Main objective of Work**	**Fault injection capabilities**	**Performance anomaly dataset generation capabilities**	**Emulation capabilities / architecture**	**Microservice support**	**Applications**
EmuFog [130]	Testing IoT apps in a staging environment before deployment; automatic placement of fog nodes	×	×	Focused on network emulation	×	Not reported
FogBed [129]	Creating an environment for resource management and service orchestration experiments	×	×	Focused on network emulation; container-based emulator	×	Healthcare prevention and monitoring system
MockFog [132]	Testing IoT apps in a staging environment before deployment	×	×	Full-system emulator	×	Ambulance cars communicating vital measures to hospitals
EmuEdge [133]	Achieving compute and network realism of a real edge environment	×	×	Full-system emulator	×	Not reported
Fogify [134]	Testing IoT apps in a staging environment before deployment	✓	×	Container-based emulator	✓	Smart transport applications
iContinuum [131]	Achieving compute and network realism; intent-based emulation	✓	×	Full-system emulator	✓	Image processing application
iAnomaly (Proposed)	Creating a toolkit to generate performance anomaly datasets	✓	✓	Full-system emulator	✓	Face detection/recognition application, Industrial machinery predictive maintenance application, Location retrieval application

Table 3.1: Comparison of performance anomaly dataset generation capabilities in edge emulators

Existing edge computing emulators can be organized under two main categories based on the level of virtualization and abstraction used to model the edge devices. They are 1) full-system emulators and 2) container-based emulators. In container-based emulators, edge devices are represented as docker containers, while full-system emulators provide a higher granularity of emulation by allowing the deployment of multiple containerized microservices within edge devices modeled as Virtual Machine (VM)s. Early emulators such as EmuFog [130], FogBed [129], MockFog [132], and EmuEdge [133] do not support the microservice-level granularity of IoT applications. Fogify [134] is the first edge emulator to support the microservice-level granularity of IoT applications. However, it is limited to deploying only a single microservice per edge device due to being a container-based emulator. Extending such emulators with dataset generation capabilities restricts data collection to device-level anomalies only. In contrast, iContinuum [131] is a full-system emulator with support for microservices deployment. Unlike container-based emulators, full-system emulators provide a higher level of realism and also allow the injection of both device-level and microservice-level anomalies. Consequently, our research aims to bridge the identified gap by developing a full-system emulator with performance anomaly dataset generation capabilities. A comparison of performance anomaly dataset generation capabilities in existing edge emulators [129–134] along with our proposed iAnomaly toolkit is shown in Table 3.1.

The main intention of existing emulators is to test IoT applications in a staging environment before deploying them into production. They are designed to achieve the compute and network realism of an edge environment, and the evaluation of these studies is also focused on those aspects. Modern emulators such as Fogify [134] and iContinuum [131] also have the capability to perform fault injections. However, the main intention of such fault injection capabilities is not to collect data for performance anomaly detection model training but to test the fault tolerance and availability aspects of IoT applications in the face of faults.

Although both Fogify and iContinuum have implemented and evaluated fault injection capabilities, neither of them has specified the tools used to inject anomalies. Since these emulators only conducted injections of a limited number of anomaly types, it can be inferred that they likely utilized basic tools, such as stress-ng, for this purpose. However, such mechanisms do not allow for the introduction of failures or disruptions in a controlled manner.

Both Fogify and iContinuum are capable of collecting both system and application metrics during monitoring. However, Fogify utilizes an in-house developed monitoring tool for this purpose, whereas open-source tools are preferred in emulators to support interoperability and transparency of code. iContinuum employs sFlow-RT, an open-source tool, to capture network and host-level metrics such as CPU and memory usage. It also integrates sFlow agents with Prometheus, another open-source tool, to collect application metrics. While Fogify requires explicit instrumentation, i.e. manually embedding monitoring code within the source code, in order to capture performance metrics, sFlowRT can only capture application metrics without explicit instrumentation from IoT applications that communicate via HTTP protocol. As most IoT applications deployed in edge computing environments are not limited to HTTP protocol and use a variety of protocols such as MQTT, RTSP, etc., it is important to be able to collect metrics from applications communicating via such non-HTTP protocols as well.

Fogify has not provided details of its workload generation tool, while iContinuum uses Locust for workload generation. However, Locust primarily focuses on generating HTTP/HTTPS workloads, while it is important to incorporate a workload generation tool supporting a wide range of protocols, not just HTTP.

Consequently, it is evident that the current emulators lack the necessary tools for generating performance anomaly data. These tools include a monitoring tool for collecting metrics, a workload generation tool for creating normal performance data, and a chaos engineering tool for injecting performance anomalies. Identifying this research gap, this chapter aims to address it by developing a toolkit with an emulator that incorporates a set of open-source tools for generating performance anomaly datasets.

In addition to finding the best open-source tools for generating performance anomaly datasets and creating a full-system emulator with performance anomaly dataset generation capabilities, we also integrate automated dataset generation features into our proposed iAnomaly toolkit. Moreover, as shown in Table 3.1, most emulators have released only one IoT application which they used in their experiments. However, we generate (and release) an open-source performance anomaly dataset using iAnomaly by deploying three IoT applications consisting of microservices with varying QoS and resource

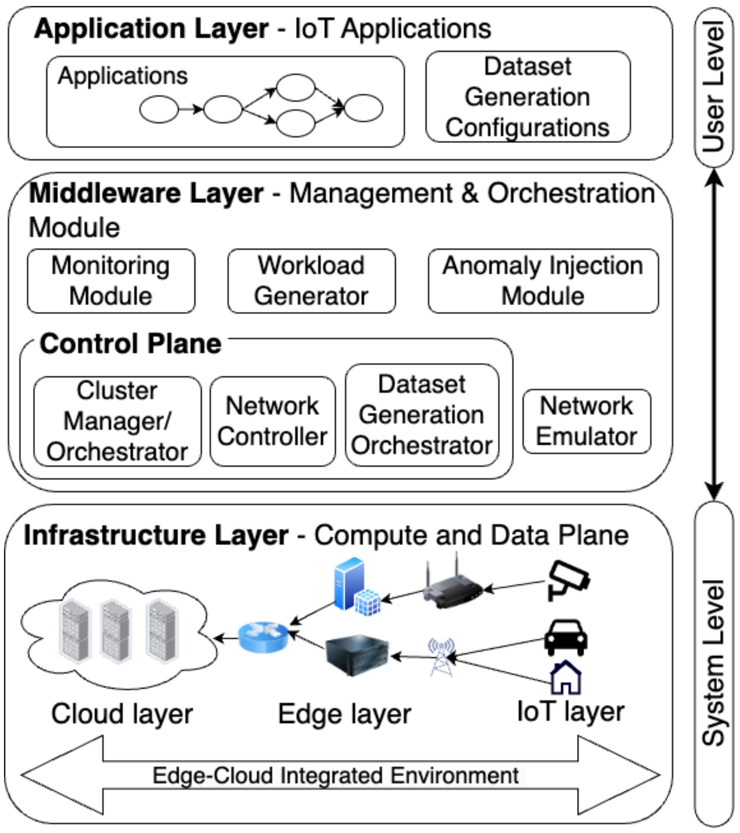

Application Layer - IoT Applications
Applications
Dataset Generation Configurations
User Level
Middleware Layer - Management & Orchestration Module
Monitoring Module
Workload Generator
Anomaly Injection Module
Control Plane
Cluster Manager/ Orchestrator
Network Controller
Dataset Generation Orchestrator
Network Emulator
Infrastructure Layer - Compute and Data Plane
Cloud layer
Edge layer
IoT layer
Edge-Cloud Integrated Environment
System Level

Figure 3.1: System architecture of iAnomaly

requirements. These applications span a wide range of domains, software architectures, service composition patterns, and communication protocols.

3.3 iAnomaly Architecture

Figure 3.1 depicts the architecture of iAnomaly. At the core of the framework is a full system emulator that comprises multiple distinct layers with a set of components to build all the layers from infrastructure to applications. The infrastructure layer hosts a diverse array of computing and networking resources. While the heterogeneity of compute nodes can be emulated using VMs with different resource capacities in the cloud, a network emulator is used to construct the network topology (by creating virtual switches and network elements) and simulate the network within the infrastructure layer. Users can define the application structure of the microservice-based IoT applications through the application layer. The middleware layer of a full-system emulator manages the deployment and operation of applications across the emulated infrastructure. Its control plane consists of a cluster manager/orchestrator that utilizes containerization and orchestration technologies to manage the computing cluster and its resources, and a network controller that uses Software-Defined Networking (SDN) technologies to manage the network (such as effectively regulating the network flow while considering resource usage conditions).

In addition to a full system emulator, iAnomaly includes components to facilitate the data generation and collection process within its middleware layer. These components consist of a monitoring module, a workload generation tool, and a tool for injecting anomalies.

The monitoring module is responsible for gathering system and application metrics from the deployed microservices. The collected data will be stored in a database and retrieved back through queries when creating the dataset. An important requirement of the monitoring tool is to be able to collect data from IoT applications communicating not only via HTTP-based protocols but also via non-HTTP protocols such as Kafka, MQTT, and RTSP. In addition, it is preferable for the tool to be capable of collecting application metrics from programs without the need for explicit instrumentation.

The workload generator is in charge of sending request/sensor data to the microservices. This tool is used during normal data generation as well as for introducing client/sensor-side anomalies such as user surges and spikes. Details of request workloads, including concurrency and duration, are specified using test plans. It is important that the workload generation tool also supports a wide range of protocols, not just HTTP. When generating data from a specific microservice, we need at least two instances of workload generators, one for generating normal workloads, and another for introducing client/sensor-side anomalies.

The anomaly injection tool is responsible for injecting server-side anomalies, such as resource hogging and service failures. MockFog [132], which is one of the early edge emulators, suggested using chaos engineering tools for injecting anomalies and conducting performance testing. Chaos engineering tools are designed to test the fault tolerance of systems by deliberately introducing failures or disruptions in a controlled manner, making them suitable for inclusion in the proposed toolkit to inject performance anomalies. Moreover, by incorporating chaos engineering tools, we can inject a diverse range of anomalies, unlike with basic tools such as stress-ng.

Generating a significant amount of normal and anomaly data by using these data collection tools is a time-consuming and repetitive task necessitating human intervention. Additionally, there is a learning curve associated with using the tools, notably in terms of creating necessary test plans (incorporating varying parameters representing normal and anomalous workloads) using the workload generator, designing chaos engineering configurations, and scripting data collection for retrieving information from the monitoring tool.

To overcome these challenges, we further extend the iAnomaly framework with automated dataset generation capabilities, where users can provide the configurations of the expected dataset, and the resulting labeled normal and anomaly data will be stored in a predefined location. As depicted in Figure 3.1, users can define the dataset generation configurations through the application layer. We also introduce a dataset generation orchestrator to the middleware layer, which interprets the content from the dataset generation configuration and coordinates with the data generation tools in the toolkit to generate the normal and anomaly data required for the performance anomaly dataset. This process will be explained in detail in the next section.

3.4 iAnomaly Implementation

This section describes the implementation details of the architecture presented in section 3.3. The deployment diagram of the iAnomaly toolkit is shown in Figure 3.2. iAnomaly relies on iContinuum [131] as a full-system emulator to accurately model edge computing environments hosting microservice-based IoT applications. iContinuum utilizes VMs with varying resource specifications to demonstrate the heterogeneity of edge devices in terms of computing and storage. Additionally, a VM with higher resource capacities acts as the master node, hosting only the tools required for compute orchestration. Acknowledging the critical role of the master node as the system's control plane, we ensure that microservices are deployed only on VMs other than the master node.

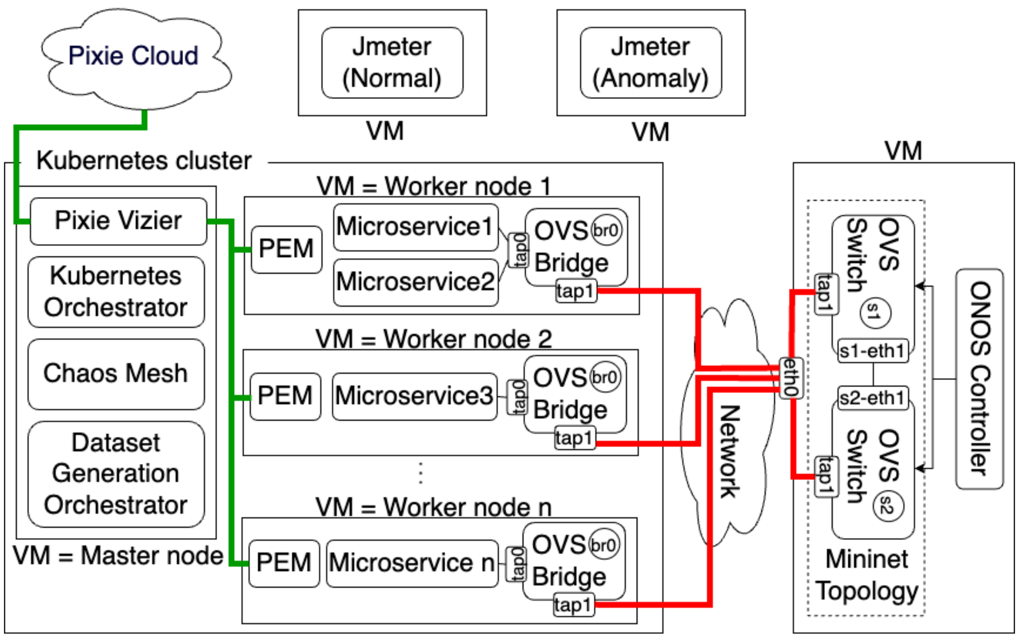


Figure 3.2: Deployment diagram of the iAnomaly toolkit

Following the implementation of iContinuum, iAnomaly uses Mininet[1] as the network emulator to construct the network topology and simulate the bandwidth between edge devices. The Open vSwitch (OVS) switches that form the Mininet topology are deployed in a separate VM, which is also nondeployable for microservices.

In line with iContinuum's control plane implementation, iAnomaly also uses Kubernetes, specifically K3s[2], which is a lightweight Kubernetes distribution designed for resource-constrained environments such as edge computing or IoT devices, as the cluster manager/orchestrator. Additionally, iAnomaly utilizes Open Network Operating System (ONOS)[3] as the network controller, and it manages the OVS switches in the Mininet topology. The ONOS controller is also deployed in the VM where the OVS switches are deployed. Figure 3.2 shows how the Kubernetes orchestrator is deployed in the master node and forms a multi-node Kubernetes cluster with the worker nodes. Each worker node is configured with an OVS bridge featuring two virtual interfaces, tap0, and tap1, out of which tap1, which is configured as a Generic Routing Encapsu-

[1]https://mininet.org/
[2]https://k3s.io/
[3]https://opennetworking.org/onos/

lation (GRE) interface, is linked to the tap1 port of the corresponding OVS switch. This ensures that the worker nodes have a bi-directional connection with the Mininet-created network topology through GRE tunnelling.

The Monitoring Module is realized by using Pixie[4], a lightweight and open-source eBPF(extended Berkeley Packet Filter)-based monitoring tool specifically designed for Kubernetes applications. eBPF-based monitoring tools, which gained popularity recently, allow sandboxed programs to execute directly inside the Linux kernel and automatically capture telemetry data in a non-intrusive manner, i.e., without requiring modifications to user-space applications [135]. It also supports monitoring a wide variety of protocols, including HTTP, Kafka, AMQP, and MySQL, making it well-suited for monitoring edge computing environments. When Pixie is deployed via the toolkit, a Pixie Edge Module (PEM) is deployed for each worker node. These modules capture monitoring data from microservices deployments on the worker nodes and send those to the Pixie Vizier deployed on the Kubernetes master node, from where they are transferred to the Pixie cloud. Pixie Vizier acts as Pixie's central collector and is also responsible for managing PEMs. When retrieving back the collected data, data retrieval queries written in Pixie language (PxL) are executed against the Pixie cloud via a Pixie API client.

The Workload Generator is implemented by leveraging Jmeter[5] as it supports a wide range of protocols, not just HTTP. We are using two separate instances of JMeter: one to generate normal workloads and the other to create client/sensor-side anomalies. Both instances are deployed outside the Kubernetes cluster to run the JMeter loads as needed and to avoid interfering with the master node's operations.

Chaos Mesh[6], an open-source chaos engineering platform for Kubernetes, is used as the anomaly injection tool and deployed in the Kubernetes master node. From there, CRD (Custom Resource Definition) YAMLs are applied to introduce server-side anomalies into the target deployments. Chaos Mesh is capable of injecting a multitude of server-side anomalies such as CPU stress, memory stress, network delay, etc.

The dataset generation orchestrator is deployed in the Kubernetes master node. As

[4]https://docs.px.dev/
[5]https://jmeter.apache.org/
[6]https://chaos-mesh.org/

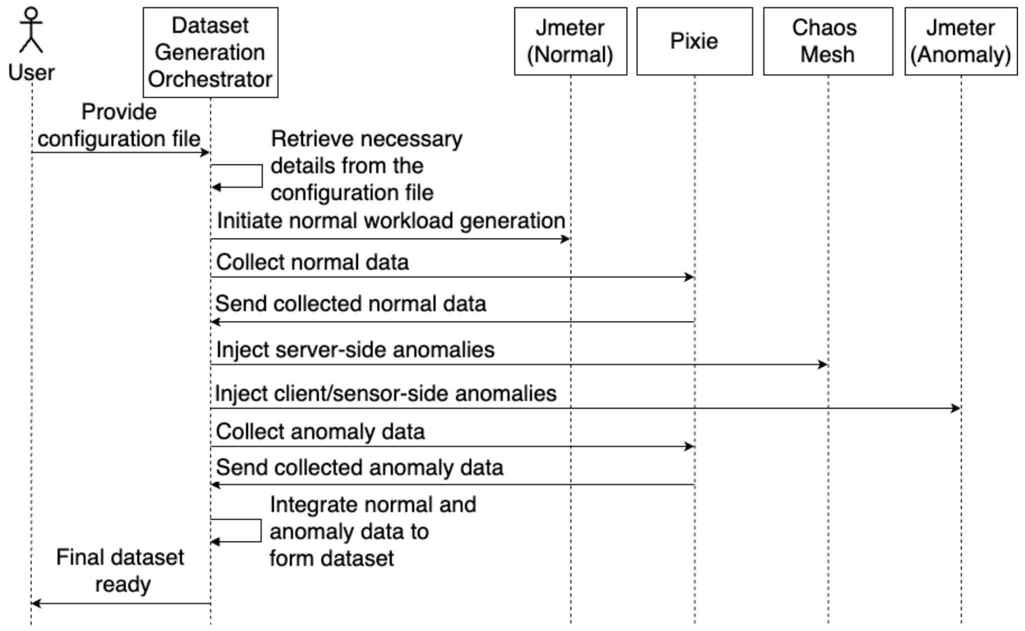

User
Dataset Generation Orchestrator
Jmeter (Normal)
Pixie
Chaos Mesh
Jmeter (Anomaly)
Provide configuration file
Retrieve necessary details from the configuration file
Initiate normal workload generation
Collect normal data
Send collected normal data
Inject server-side anomalies
Inject client/sensor-side anomalies
Collect anomaly data
Send collected anomaly data
Integrate normal and anomaly data to form dataset
Final dataset ready

Figure 3.3: Interactions between the dataset generation orchestrator and other components

illustrated in Figure 3.3, it acts as the central component that interprets the content from the dataset generation configuration/s and coordinates with the data generation tools in the toolkit to produce the labeled normal and anomaly data needed for the performance anomaly dataset. To start the data generation process, the dataset generation orchestrator reads the configuration file/s to retrieve the deployment details, normal data collection parameters, and anomaly injection settings. It then remotely executes the test plans on Jmeter's normal data generation instance using Paramiko's[7] SSH client to initiate normal workload generation. Once the normal data has been generated for the required duration, the orchestrator executes PxL queries to collect the generated normal data. Simultaneously, while the normal workload generation process is ongoing, the orchestrator proceeds to inject anomalies. Anomalies are injected either through server-side disruptions using Chaos Mesh or via client/sensor-side anomalies through Jmeter's anomalous data generation instance. For server-side disruptions, the dataset generation orchestrator applies the corresponding chaos YAMLs through the installation of helm

[7]https://docs.paramiko.org/

charts[8]. After the entire anomaly injection period, the orchestrator executes PxL queries to collect the corresponding anomaly data. Finally, both normal and anomaly data are funnelled back to the orchestrator, which integrates and processes the data to create a comprehensive dataset suitable for training and evaluating performance anomaly detection models.

Therefore, the realization of the architecture proposed in section 3.3 using the open-source tools discussed earlier, has made the process of dataset generation easily accessible for researchers. We have released the source code of the iAnomaly toolkit, which includes iContinuum as its full-system emulator together with the chosen open-source data collection tools and the code for automated dataset generation, in a public repository[9].

3.5 Case Study: Dataset Generation

This section showcases how iAnomaly was used to create an open-source labeled dataset consisting of normal and anomalous data collected for three different IoT applications, followed by an analysis of the generated performance anomaly dataset.

3.5.1 IoT Applications

The generated dataset records data for three IoT applications typically deployed in edge environments:

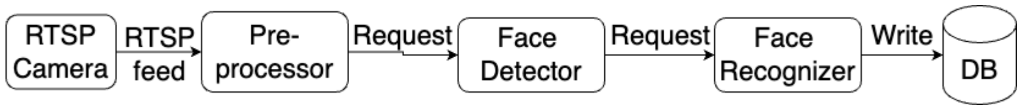


Figure 3.4: Face detection/recognition application

Face detection/recognition This application showcases the scenario of using computer vision for secure access control at a corporate office building. As illustrated in Figure

[8]https://helm.sh/
[9]https://github.com/Cloudslab/iAnomaly

Application	Microservice	Type of task performed	Software architecture	Service composition pattern	QoS properties
Face detection/ recognition application	Preprocessor	Computer vision-based	Stream processing	Chained	LC, HTp, HCI, BI
	Face detector	Computer vision-based	Request–response		LC, MTp, HCI
	Face recognizer	Computer vision-based	Request–response		LC, LTp, HCI
	Database	General purpose	Request–response		LT, LTp
Industrial machinery predictive maintenance application	Orchestrator	Time-series processing	Publish–subscribe	Aggregator	LC, LTp
	Emergency event detection	Time-series processing	Request–response		LC, LTp, MCI
	Missing data imputation	General purpose	Request–response		LC, LTp, MCI
Location retrieval application	Location service with timed cache	Simple non-resource-intensive	Request–response	Passthrough	LC, HTp
LC: Latency Critical, LT: Latency Tolerant, HTp: High Throughput, MTp: Moderate Throughput, LTp: Low Throughput, HCI: High Compute Intensive, MCI: Moderate Compute Intensive, BI: Bandwidth Intensive					

Table 3.2: Properties of IoT applications used to generate performance anomaly dataset

3.4, it comprises four microservices: 1) preprocessor, 2) face detector, 3) face recognizer, and 4) database. Cameras at entry points (e.g., doors, gates) produce an RTSP stream. The preprocessor microservice reads from this video stream, performing resizing and grayscale conversion on the images and carrying out motion detection by thresholding the difference between consecutive frames. Upon detection of motion (i.e., when an employee approaches), the frame is sent to the face detector microservice. This microservice utilizes a Multi-Task Cascaded Convolutional Neural Network (MTCNN) to detect bounding boxes and landmark points of faces in the frame. Additionally, it conducts face alignment using affine transformation and supports multi-face alignment within a single frame. Aligned faces are then input to the face recognizer microservice. The face recognizer microservice employs a ResNet-50 model to extract features from the detected faces and calculates the cosine distance between these features and the features of the authorized personnel's faces to calculate the similarity between detected faces and faces of authorized personnel. Upon successful recognition, the timestamp is logged in the database - which is implemented as another microservice - marking the employee's entry or exit from the building.

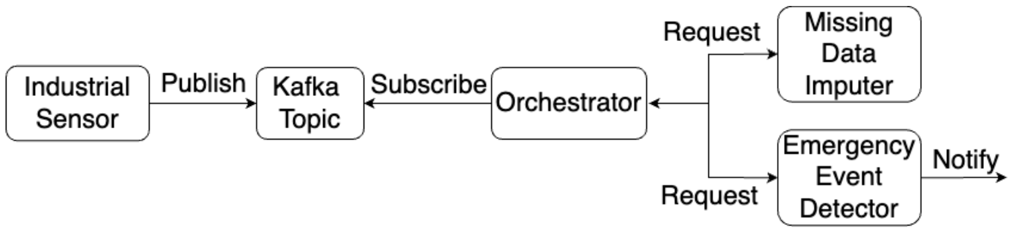


Figure 3.5: Industrial machinery predictive maintenance application

Predictive maintenance for industrial machinery In this application, IoT sensors that measure temperature, vibration, and pressure continuously generate multivariate time series data, which is then written to a Kafka topic. As shown in Figure 3.5, an orchestrator microservice subscribes to this Kafka topic, reads the raw sensor data, and forms time-series windows, which are sent to the emergency event detector microservice. Before the windowed data is sent for emergency event detection, the orchestrator imputes any missing values by calling the missing data imputer microservice and also standardizes the data using standard score normalization. The emergency event detector uses an Isolation Forest (IF) model to detect whether a window of standardized time series data is anomalous by detecting patterns that deviate from the norm, such as overheating, excessive vibration, or mechanical stress. When an anomaly is detected, alerts are triggered to the maintenance teams.

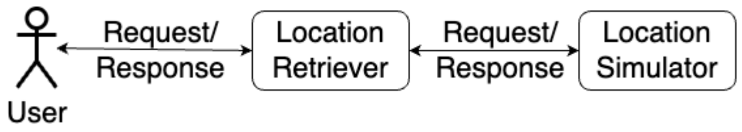


Figure 3.6: Location retrieval application

Location retrieval This application depicts the scenario of fleet management for logistics or delivery services. It facilitates tracking the real-time location of delivery vehicles in a fleet. As shown in Figure 3.6, when a dispatcher or a customer requests the location of a specific vehicle, the request is directed to the location retriever microservice. This microservice checks its Least Recently Used (LRU) cache, and if the vehicle's location was recently queried, it provides the cached location for quick access. If the location is not in the cache, the microservice queries the vehicle's current location, updates the cache, and returns the most up-to-date information. In our implementation, the location simulator microservice is used to mock the location of the delivery vehicle by generating GPS locations in a trajectory.

As shown in Table 3.2, the aforementioned applications span the properties of a wide range of IoT applications. For instance, the face detection/recognition application falls under Computer Vision (CV), the industrial machinery predictive maintenance application focuses on time-series processing, and the location retrieval application is a simple, non-resource-intensive application. Each application relies on different software architectures, including stream processing, request-response, and publish-subscribe, which are implemented using different communication protocols, including HTTP, Kafka, RTSP, and MySQL. The applications also employ different service composition patterns, such as chained, aggregator, and passthrough. Most importantly, the microservices in these applications have unique QoS and resource requirements.

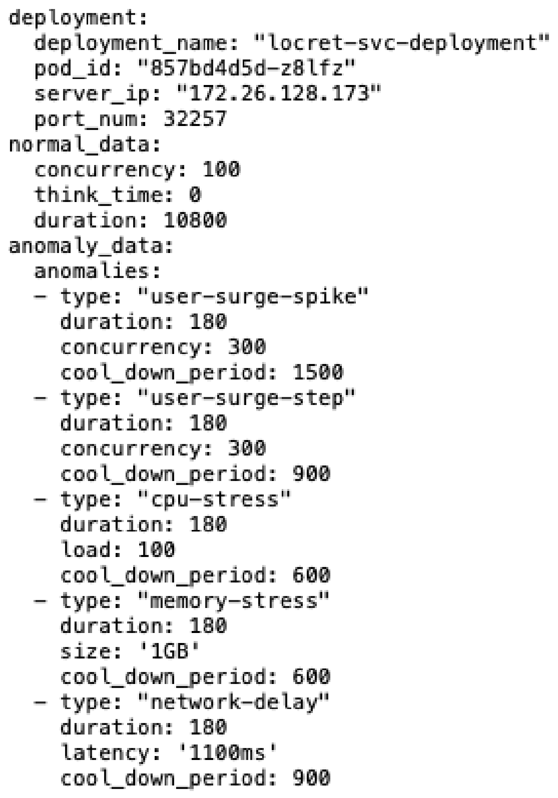
The iAnomaly repository also contains the Python implementation for all three IoT applications. In addition, we have made the Docker images of the microservices accessible to the public[10].

3.5.2 Dataset Generation

The applications presented in Section 3.5.1 were deployed in a Kubernetes cluster, where iAnomaly was responsible for the orchestration and automation of the data generation and collection process. Specifically, the physical environment consisted of ten VMs with heterogeneous computing and storage specifications created in the Melbourne Research Cloud[11] to emulate the worker nodes. In particular, two 2vCPU/8G VMs represented the IoT layer, four 2vCPU/8G VMs represented Fog level 1, three 4vCPU/16GB VMs represented Fog level 2, and one 8vCPU/32GB VM represented Fog level 3. The configuration of the emulated network was determined based on existing related research [136], with the following specifications: IoT layer $\rightarrow$ Fog level 1: 5ms/100Gbps, Fog level 1 $\rightarrow$ Fog level 2: 20ms/10Gbps, Fog level 2 $\rightarrow$ Fog level 3: 50ms/0.15Gbps, and 2ms bandwidth among nodes at the same level. The applications were assigned to the worker

[10]https://hub.docker.com/repository/docker/dtfernando/ianomaly

[11]https://dashboard.cloud.unimelb.edu.au/



```yaml
deployment:
  deployment_name: "locret-svc-deployment"
  pod_id: "857bd4d5d-z8lfz"
  server_ip: "172.26.128.173"
  port_num: 32257
normal_data:
  concurrency: 100
  think_time: 0
  duration: 10800
anomaly_data:
  anomalies:
  - type: "user-surge-spike"
    duration: 180
    concurrency: 300
    cool_down_period: 1500
  - type: "user-surge-step"
    duration: 180
    concurrency: 300
    cool_down_period: 900
  - type: "cpu-stress"
    duration: 180
    load: 100
    cool_down_period: 600
  - type: "memory-stress"
    duration: 180
    size: '1GB'
    cool_down_period: 600
  - type: "network-delay"
    duration: 180
    latency: '1100ms'
    cool_down_period: 900
```

Figure 3.7: Dataset generation configuration YAML file of the location retrieval application

nodes by following the QoS-aware scheduling algorithm proposed by Pallewatta et al. in their research study [9].

Thereafter, normal and anomaly data were generated from each application by providing dataset generation configurations specified in the form of YAML files. For example, the configuration shown in Figure 3.7 was used to generate data from the location retrieval application. This configuration instructs iAnomaly to generate and collect normal performance data from the location retriever microservice over a duration of three hours. Additionally, it specifies the injection of five types of anomalies—CPU hog, memory stress, user surge spike, user surge step, and network delay—over a total duration of two hours. Similar configurations were used to collect performance data from all applications. While five types of anomalies (two client-side and three server-side) were injected into the location retrieval application, only a subset of these anomalies was introduced into the other applications. This selection was based on the likelihood of each

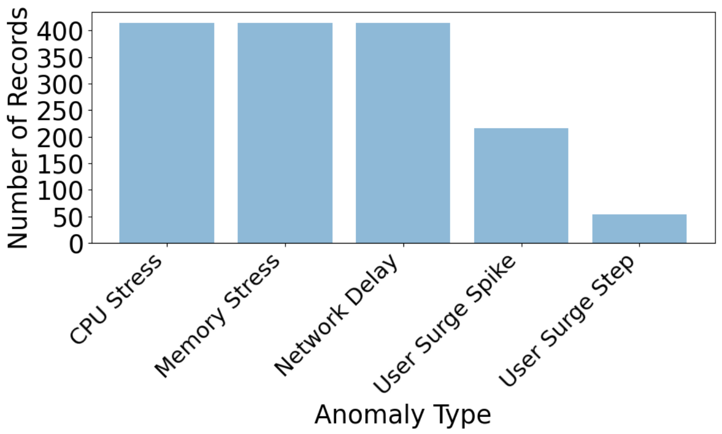
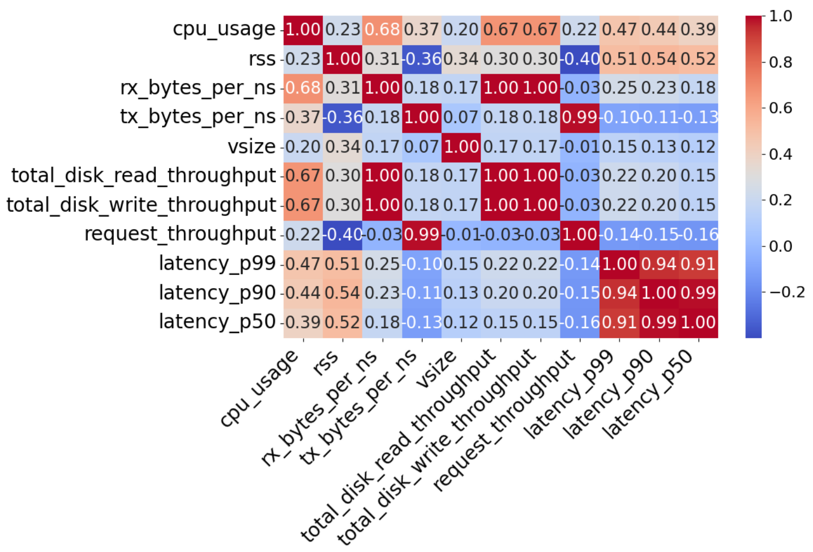


Figure 3.8: Distribution of records by anomaly type

anomaly type occurring in real-world conditions for the respective applications. Figure 3.8 depicts the distribution of different types of anomalies across the dataset. It is also important to note that data for each application was collected independently to avoid anomalies caused by colocation.

Pixie's default granularity of 10 seconds was used when collecting data. For each application, data was collected across 12 metrics, covering key aspects of system and application performance: disk read and write throughput (total_disk_read_throughput, total_disk_write_throughput), memory usage (rss, vsize), CPU utilization (cpu_usage), network activity (rx_bytes_per_ns, tx_bytes_per_ns), latency percentiles (latency_p50, latency_p90, latency_p99), request throughput (request_throughput), and error rate (errors_per_ns), collectively providing a comprehensive view of resource consumption, network efficiency, and service reliability.

The final dataset comprises a total of 30240 records. Of these, 19260 records attribute to 54 hours of normal data, and 10980 records account for 31 hours of anomalous data. Within this dataset, there are 1512 records labeled as anomalous data points, resulting in an anomaly ratio of 5%. This ratio is consistent with the anomaly ratio of other standard anomaly datasets, such as SMD (5.84%) and ASD (4.61%) [137], indicating that our dataset maintains a realistic anomaly density—an important characteristic of a high-quality anomaly dataset [138]. The collected dataset is also made available at the iAnomaly repository.



	cpu_usage	rss	rx_bytes_per_ns	tx_bytes_per_ns	vsize	total_disk_read_throughput	total_disk_write_throughput	request_throughput	latency_p99	latency_p90	latency_p50
cpu_usage	1.00	0.23	0.68	0.37	0.20	0.67	0.67	0.22	0.47	0.44	0.39
rss	0.23	1.00	0.31	-0.36	0.34	0.30	0.30	-0.40	0.51	0.54	0.52
rx_bytes_per_ns	0.68	0.31	1.00	0.18	0.17	1.00	1.00	-0.03	0.25	0.23	0.18
tx_bytes_per_ns	0.37	-0.36	0.18	1.00	0.07	0.18	0.18	0.99	-0.10	-0.11	-0.13
vsize	0.20	0.34	0.17	0.07	1.00	0.17	0.17	-0.01	0.15	0.13	0.12
total_disk_read_throughput	0.67	0.30	1.00	0.18	0.17	1.00	1.00	-0.03	0.22	0.20	0.15
total_disk_write_throughput	0.67	0.30	1.00	0.18	0.17	1.00	1.00	-0.03	0.22	0.20	0.15
request_throughput	0.22	-0.40	-0.03	0.99	-0.01	-0.03	-0.03	1.00	-0.14	-0.15	-0.16
latency_p99	0.47	0.51	0.25	-0.10	0.15	0.22	0.22	-0.14	1.00	0.94	0.91
latency_p90	0.44	0.54	0.23	-0.11	0.13	0.20	0.20	-0.15	0.94	1.00	0.99
latency_p50	0.39	0.52	0.18	-0.13	0.12	0.15	0.15	-0.16	0.91	0.99	1.00

Figure 3.9: Colinearity among metrics in the generated dataset

3.5.3 Analysis of the Generated Dataset

Figure 3.9 illustrates the colinearity among the metrics in the generated dataset after excluding errors_per_ns metric, which contains all-zero values. The plot shows that inherently related groups of metrics, such as disk read/write throughputs, as well as latency percentiles, are highly correlated. In addition, metrics such as request_throughput and tx_bytes_per_ns, as well as disk read/write throughputs and rx_bytes_per_ns, exhibit a strong positive correlation with each other. Outside of these groups, most other metrics do not show strong correlations with each other, indicating that each metric serves a unique purpose within the dataset. To simplify our analysis, we select a single metric from each identified group of correlated metrics to serve as a proxy for the others in the group. Based on this correlation analysis, we identify cpu_usage, rss, rx_bytes_per_ns, vsize, request_throughput, and latency_p50 as the subset of metrics with the lowest colinearity. Consequently, we will focus on these metrics for further analysis.

Figure 3.10 illustrates the distribution of normal data for each shortlisted metric, focusing on three instances of the location retriever microservice and one instance of each of the other microservices. The three instances of the location retriever correspond to different deployments in regions with varying user populations. In the latency_p50

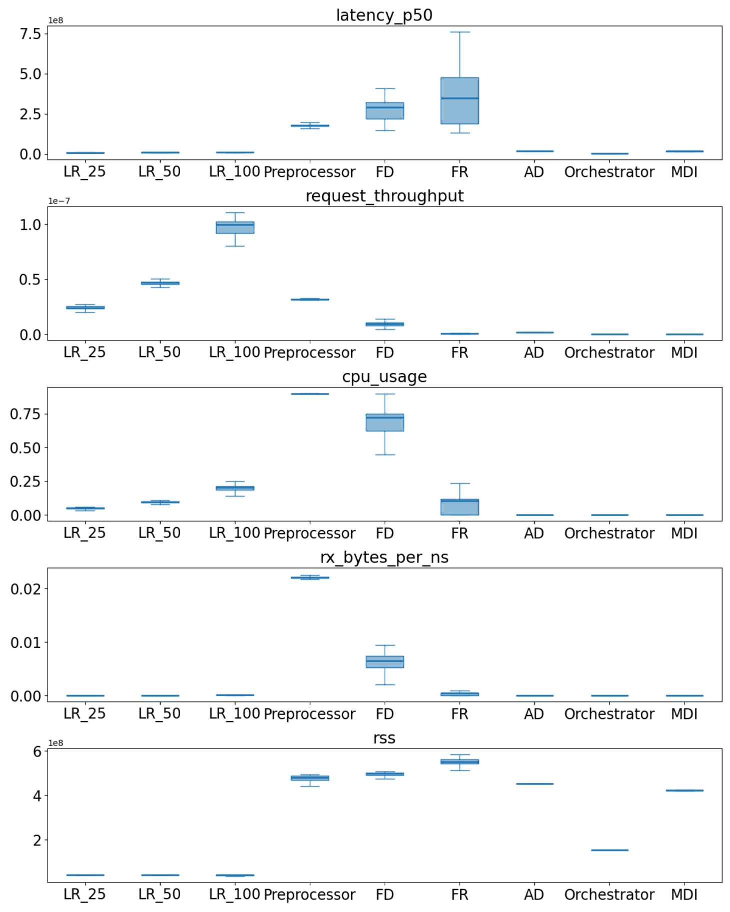
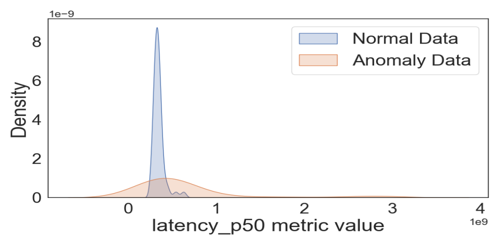


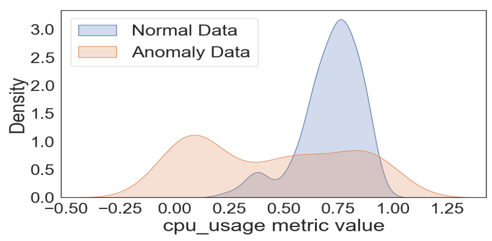
Figure 3.10: Distribution of normal data across shortlisted metrics

subplot, it is evident that the preprocessor, face detector, and face recognizer microservices experience the highest latency. This increased latency is attributed to their highly compute-intensive (HCI) nature, which requires more time to process and respond to individual requests. In contrast, the other microservices demonstrate lower latency due to their relatively lower compute intensity.

The second subplot represents the request_throughput metric. Here, we can observe that the preprocessor and location retriever microservices exhibit high request throughput (HTp), while the face detector shows moderate throughput (MTp). The other microservices fall into the low throughput (LTp) category. These observations confirm the expected QoS properties of the microservices, as listed in Table 3.2. The third subplot corresponds to the cpu_usage metric. Despite being HTp, the location retriever microser-



(a) Latency_p50 of the preprocessor



(b) CPU_usage of the face detector

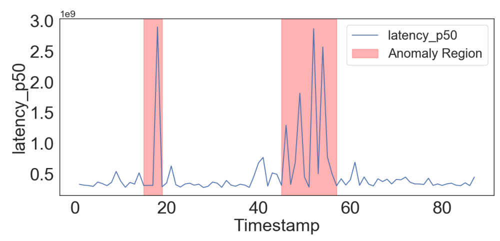
Figure 3.11: PDFs for normal and anomalous data distributions of selected metrics

vices result in low CPU usage since they are not computationally intensive. The preprocessor microservice shows the highest CPU usage due to its HCI nature and high request throughput. The face detector microservice, while also HCI, has moderate throughput and, therefore, has the second-highest CPU usage. The rest of the microservices have a low CPU usage due to their LTp nature.

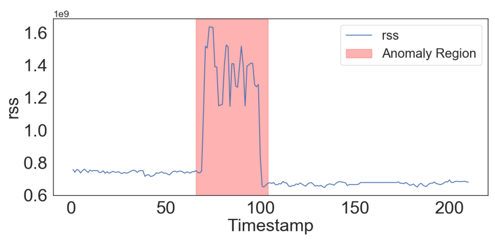
The rx_bytes_per_ns subplot confirms the bandwidth-intensive (BI) nature of the preprocessor microservice. The final subplot, which corresponds to the rss metric, indicates that three computer vision microservices, together with the anomaly detector and the missing data imputer (which also utilize ML models for processing), have high rss values, demonstrating significant memory usage. By comparing these subplots, we can see that the diversity of the selected applications allows our collected dataset to effectively capture the variations in QoS and resource requirements of the microservices, as expected from an edge dataset.

Figure 3.11 contains the Probability Density Functions (PDFs) for the normal and anomalous data distributions of two randomly selected metrics from the datasets of the preprocessor and face detector microservices. Subfigure 3.11(a) corresponds to latency_p50 of the preprocessor microservice while figure 3.11(b) corresponds to cpu_usage of the face detector microservice. Both subplots illustrate that the anomalous data overlaps with the distribution of normal data. This overlap proves that the anomalies present in our dataset are non-trivial and not merely outliers. Renjie et al. [138] have identified the presence of such non-trivial anomalies as a property of a good anomaly dataset.

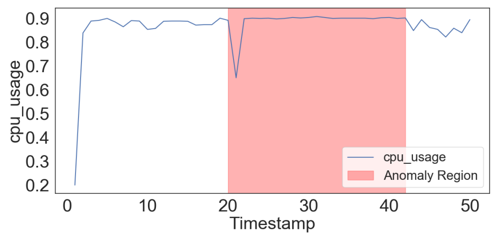
Furthermore, Figure 3.12 visualizes a few selected anomalies from the dataset. While certain anomalies are easily noticeable using their respective metrics - for example, user



(a) User surges using latency_p50



(b) Memory stress using rss



(c) CPU stress using cpu_usage

Figure 3.12: Visualization of selected anomalies from the dataset

surge anomalies are evident from the increase in the latency_p50 metric (Figure 3.12(a)), and memory stress is apparent from the rss metric (Figure 3.12(b)) - some anomalies, such as CPU stress, cannot be detected simply by looking at the cpu_usage metric (Figure 3.12(c)), especially when it occurs in compute-intensive microservices. During such scenarios, which are non-trivial to detect, algorithms that are capable of analyzing the higher-order relationships and behavior of several metrics are required to make an accurate detection.

Successful collection of the dataset was possible due to iAnomaly's use of an optimal set of open-source tools. In particular, leveraging Pixie as the monitoring tool allowed it to gather metric data from all three IoT applications, each using different communication protocols. In contrast, using a regular full-system emulator like iContinuum would only allow data collection from the location retrieval application, which uses HTTP for communication. Furthermore, iAnomaly's automated dataset generation capabilities led to an 87% reduction in code lines compared to using a regular full-system emulator such as iContinuum during our dataset generation. Notably, iAnomaly completely eliminated the need for human intervention during the data collection process, requiring only 31 lines of configurations per microservice, while iContinuum needs 307 lines of code and significant human involvement to generate the same dataset.

3.6 iAnomaly Dataset Generation Simulator

To conduct scalability experiments that reflect the large-scale nature of edge computing environments, generating datasets using the iAnomaly emulator alone is impractical due to the substantial computational resources required to model thousands of nodes. Consequently, we extend the iAnomaly toolkit with a dataset generation simulator capable of producing large-scale service dependency graphs representing microservice-based edge environments. In these graphs, nodes correspond to computational entities (such as microservices and edge devices), while edges capture communication and co-location dependencies. The simulator supports graph sizes ranging from 50 to 10,000 nodes.

To preserve realism, the simulator is designed based on the original iAnomaly framework and the released performance anomaly dataset. Specifically, the generated graphs are populated with anomalous and normal traces derived from real execution data produced by the iAnomaly emulator. This approach enables the simulator to retain realistic workload patterns and anomaly characteristics. Compared to purely synthetic simulators that rely on artificially generated data, the proposed simulator provides more representative and credible datasets for large-scale evaluations.

Algorithm 1 outlines the workflow of the proposed dataset generation simulator for constructing large-scale service dependency graphs with realistic performance and anomaly characteristics. The algorithm takes as input the desired number of nodes N, which represent computational entities in a microservice-based edge environment. First, the number of edge devices is determined using a predefined node-to-device ratio, and nodes are placed onto these devices using a suitable, user-defined placement strategy (e.g., communication-aware placement). This step models realistic resource distribution and deployment constraints in edge infrastructures. Next, nodes are assigned to multiple applications and organized according to predefined, application-specific microservice communication topologies, such as tree, mesh, pipeline, and Directed Acyclic Graph (DAG) structures. This enables the simulator to represent diverse application

Algorithm 1: iAnomaly dataset generation simulator for large-scale service dependency graphs

```
Input: Number of nodes $N$
Output: Service dependency graph $G = (V, E)$ populated with time-series
        metrics and anomaly labels
1  Determine the number of edge devices $D$ using a predefined node-to-device
     ratio;
2  Initialize node set $V$ with $|V| = N$ and empty edge set $E$;
3  Place nodes onto $D$ edge devices using a suitable, user-defined placement
     strategy (e.g., communication-aware placement).;
4  Assign nodes to applications and organize them according to predefined,
     application-specific microservice communication topologies (e.g., tree, mesh,
     pipeline, DAG);
5  Construct $E$ to reflect dependencies among computational entities;
6  foreach $v \in V$ do
7      Sample baseline (normal) time-series traces for $v$ from the iAnomaly dataset;
8      Inject anomalies in a randomized manner by sampling anomalous traces,
         following predefined anomaly injection policies (e.g., anomaly type, rate,
         duration);
9      Add controlled random noise to simulate measurement uncertainty and
         environmental variability;
10     Attach the resulting time-series metrics and anomaly labels to $v$;
11 end foreach
12 return $G = (V, E)$;
```

architectures commonly observed in real-world edge systems. Based on these topologies, service dependency edges are constructed to capture both communication and co-location relationships among nodes.

After the graph structure is formed, each node is populated with time-series performance data. Baseline behavior is generated by sampling normal traces from the iAnomaly dataset, while anomalies are introduced in a randomized manner using anomalous traces, following predefined injection policies. These policies control factors such as anomaly type, frequency, and duration, ensuring both variability and experimental reproducibility. Controlled random noise is further added to simulate measurement uncertainty and environmental fluctuations. Finally, the algorithm outputs a service dependency graph in which each node is annotated with realistic performance metrics and anomaly labels. By combining topology-aware graph construction with trace-driven anomaly injection, the proposed simulator generates scalable and representative datasets suitable for evaluating anomaly detection and RCL techniques in large-scale edge environments.

3.7 Summary

This chapter addressed the fundamental lack of realistic and reproducible datasets for performance anomaly research in edge computing environments. We presented iAnomaly, a full-system emulation toolkit that integrates Pixie, JMeter, and Chaos Mesh with an automated dataset generation orchestrator to enable systematic and scalable data collection. Using the proposed framework, we generated and released a publicly available performance anomaly dataset covering multiple microservice-based IoT applications and diverse anomaly types. Experimental analysis confirmed that the collected data captures key characteristics of real edge deployments and exhibits non-trivial anomalous behavior. In addition, we introduced a trace-driven dataset generation simulator that extends iAnomaly to support scalability experiments by producing service dependency graphs with up to 10,000 nodes populated using normal and anomalous traces derived from the released dataset. By providing both an extensible data generation platform and scalable, realistic datasets, this chapter establishes the empirical foundation required for the efficient model training and diagnostic techniques developed in subsequent chapters.

Chapter 4

Efficient Training Approaches for Performance Anomaly Detection Models in Edge Computing Environments

While the availability of representative datasets enables effective anomaly detection model development, training such models in large-scale edge environments remains challenging due to resource constraints and device heterogeneity. Existing approaches typically favor either high accuracy through device-specific models or improved efficiency through global models, resulting in suboptimal trade-offs. This chapter investigates efficient training strategies for performance anomaly detection in edge systems and proposes two clustering-based approaches: intra-cluster parameter transfer learning (ICPTL) and cluster-level model training (CM). By exploiting similarities in normal behavior across devices, the proposed methods reduce redundant training while preserving diagnostic accuracy. Extensive evaluations demonstrate that ICPTL achieves accuracy comparable to device-specific models with substantially reduced training cost, while CM further improves efficiency by limiting the number of deployed models. These results establish scalable training mechanisms suitable for large edge deployments.

This chapter is derived from:

- **Duneesha Fernando**, Maria A. Rodriguez, Patricia Arroba, Leila Ismail and Rajkumar Buyya, "Efficient Training Approaches for Performance Anomaly Detection Models in Edge Computing Environments", *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, Volume 20, Pages: 1-27, ISSN: 1556-4665, Association for Computing Machinery, New York, NY, USA, June 2025.

4.1 Introduction

Existing research studies on performance anomaly detection for edge computing environments advocate for using unsupervised multivariate time-series methods [11, 12, 16, 18], where a model learns the normal performance data distribution of microservices and identifies deviations as anomalies. Microservices deployed across devices in an edge infrastructure have heterogeneous QoS and resource requirements, resulting in heterogeneous normal data distributions. One approach to ML-based anomaly detection at the edge is to train a single, generic anomaly detection model (GM) using data from all microservices across all devices [12, 21, 22]. Training a single model is efficient in terms of training time and resource usage, and reduces the overhead associated with model management. However, it requires aggregating local (edge device) time series data in a centralized location, usually a cloud data center, thus considerably increasing network utilization and potentially causing network congestion [16]. Importantly, a generic model usually yields lower accuracy due to the difficulty of generalizing across the diverse normal data distributions. An alternative approach involves training a "model per edge device" (MPD) using data collected from the microservices deployed in that specific edge device [11, 16, 19, 20]. Since microservices within a single edge device usually have similar QoS and resource requirements [9, 98] and hence have similar normal data distributions, this approach results in models with higher accuracy, as the model is expected to specialize well across similar normal data distributions. Training as many models as edge devices consumes a significant amount of resources and results in longer aggregate training times, as well as increased model management overhead. While a model per device allows for each edge device to train its associated anomaly detection model, performing training, which is a resource-intensive process, might be challenging for most edge devices due to their resource-constrained nature.

The aforementioned approaches (which we identify as baselines) represent extremes in terms of accuracy and efficiency; a generic model (GM) has low accuracy but can be efficiently trained, while the model per device approach (MPD) yields high accuracy, but the training process is time-consuming and resource-hungry. Considering the resource constraints and large number of devices present in modern edge platforms, this research aims to bridge the gap between GM and MPD by finding a trade-off between the training efficiency of anomaly detection models and their accuracy. We propose to achieve this by conducting model training within clusters of edge devices with similar normal data distributions. To that end, we exploit the QoS-aware placement of microservice-based IoT applications [9, 10, 98] in edge computing environments to propose a similarity metric capable of clustering edge devices with similar normal data distributions. Furthermore, we introduce two clustering-based training approaches, namely, (1) intra-cluster parameter transfer learning-based model training (ICPTL) and (2) cluster-level model training (CM). ICPTL aims to match MPD's accuracy with fewer training cycles, while CM aims to further improve training efficiency by reducing the number of models. To the best of our knowledge, this is the first work to propose anomaly detection model training approaches tailored for the specific characteristics and constraints of edge environments with the aim of increasing training efficiency while achieving high model accuracy.

We conducted a comprehensive and reproducible evaluation of the proposed clustering based training approaches using a publicly available and widely-used performance anomaly dataset called the "Server Machine Dataset (SMD)" [72], which consists of devices with heterogeneous normal data distributions. Initially, we compared four common unsupervised multivariate time-series anomaly detection techniques for accuracy and efficiency (in terms of resource utilization and detection time) against the SMD dataset to determine the most suitable algorithm for the rest of our evaluations. Upon evaluating the proposed clustering-based training approaches using the identified algorithm, we demonstrate that the clustering approaches achieve a balance between accuracy and efficiency, as expected. To further validate the results, we conducted a small-scale experiment using data collected from a set of microservices with heterogeneous QoS and resource requirements deployed in an emulated edge computing environment introduced through chapter 3.

The rest of this chapter is organized as follows: Section 4.2 reviews the existing related works. Section 4.3 presents the clustering-based training approaches. Section 4.4 evaluates the performance anomaly detection algorithms and the proposed training approaches. Section 4.5 discusses how the proposed training approaches handle the dynamic nature of edge computing environments. Section 4.6 discusses the validity threats of our study. Section 4.7 concludes the chapter.

4.2 Related Work

In this section, we provide an overview of performance anomaly detection studies conducted in edge computing environments in terms of the algorithms, evaluation metrics, and training methods they have used. We also explore several related works on ML model training approaches employed in edge computing environments.

4.2.1 Performance anomaly detection in edge computing environments

Several works have proposed monitoring solutions for edge computing environments, some of which have incorporated alerting as a functionality [55, 71]. However, none of these monitoring solutions perform any advanced form of anomaly detection other than threshold-based alerting. Threshold-based alerting requires system administrators to manually define thresholds for each metric per device, which is not scalable given the large number of edge devices and the multitude of performance metrics being monitored. Moreover, these static thresholds become obsolete faster in dynamic edge computing environments. In contrast, multivariate time series anomaly detection algorithms can accurately detect more types of anomalies since they consider inter-metric correlations.

Building on the limitations of threshold-based alerting, recent research on performance anomaly detection for edge computing environments suggests using unsupervised multivariate time-series techniques for anomaly detection [11, 12, 16, 18]. Becker et al. evaluated ARIMA, BIRCH, LSTM, and EMA algorithms, all of which are unsupervised anomaly detection approaches [11]. Similarly, Skaperas et al. evaluated two change point (CP) detection approaches: Bayesian and cumulative sum (CUSUM), both of which can be identified as unsupervised approaches [18]. Tuli et al. explored several unsupervised approaches, including OCSVM, IF, DILOF, and USAD [16]. Although Soualhia et al. evaluated a group of supervised approaches, they emphasized the infeasibility of such methods due to the need for labeled normal and anomaly data for

Work	Anomaly Detection Algorithms	Anomaly Detection Evaluation Aspects		Model Training Approach	Main Objective of Work
		Accuracy	Efficiency		
[12]	ARIMA (Conventional), MC (ML), SVM (ML), RF (ML), NN (ML), BN (ML), TAN (ML)	✓	×	Generic Model (GM)	A framework to perform anomaly detection in edge environments.
[11]	ARIMA (Conventional), BIRCH (ML), LSTM (DNN), EMA (Conventional)	×	✓CPU utilization, detection delay	Model per device (MPD)	AIOPS framework for edge environments.
[16]	OCSVM (ML), IF (ML), DILOF (DL), ONLAD (ML), CAE-M (DL), USAD (DL), MAD-GAN (DL), SlimGAN (DL)	✓	✓Memory consumption, inference time	Model per device (MPD)	Propose memory-efficient anomaly detection approaches for edge devices.
[18]	CPD approaches: Bayesian (Conventional), CUSUM (Conventional)	✓	✓Resource utilization (CPU and memory consumption), detection delay	Model per device (MPD)	Framework that facilitates evaluation of CPD algorithms.
Our work	VAR (Conventional), IF (ML), AE (DNN), LSTM-AE (DNN)	✓	✓Detection time, resource requirements (CPU and memory)	Similarity-based clustering approaches	Propose efficient model training approaches for performance anomaly detection in edge environments.
BIRCH: Balanced Iterative Reducing and Clustering using Hierarchies, EMA: Exponential Moving Average, LSTM: Long Short-Term Memory, SVM: Support Vector Machine, RF: Random Forest, NN: Neural Network, MC: Markov Chain, BN: Bayesian Network, IF: Isolation Forest, TAN: Tree-Augmented Naive Bayes, OCSVM: One-Class Support Vector Machine, DILOF: Deep Isolation Forest, ONLAD: Online Anomaly Detection via Adaptive Learning, CAE-M: Convolutional Autoencoder-Mahalanobis, USAD: Unsupervised Anomaly Detection, MAD-GAN: Multi-scale Anomaly Detection with GANs, SlimGAN: Slim Generative Adversarial Network, CUSUM: Cumulative Sum Control Chart, VAR: Vector AutoRegression, AE: AutoEncoder, LSTM-AE: LSTM-AutoEncoder					

Table 4.1: A summary of performance anomaly detection studies in edge computing environments

training [12]. As anomaly data may not be available during the initial execution of applications and labeling can be costly, they suggest using unsupervised anomaly detection approaches. These algorithms are trained on unlabeled normal data to learn the normal data distribution and identify deviations from it as anomalies. Therefore, in our work, we utilize unsupervised multivariate time-series anomaly detection techniques.

Although research on performance anomaly detection in edge computing environments is still emerging, AI-based performance anomaly detection in cloud and general microservices environments is well-established [78, 85, 139–142]. According to Audibert et al. [78], existing anomaly detection algorithms can be broadly classified into three categories: (1) conventional methods, (2) machine learning (ML)-based methods, and (3) deep neural network (DNN)-based methods. Studies on AI-based performance anomaly detection in cloud and microservices environments typically evaluate proposed algorithms against representative techniques from all three categories. Similarly, research on performance anomaly detection in edge environments often adapts cloud anomaly detection approaches [11, 12, 16, 18], following the trend of assessing algorithms across these three groups. In line with this practice, section 4.4.2 evaluates common unsupervised multivariate time-series anomaly detection techniques using the SMD dataset to identify the most suitable algorithm for the rest of our evaluations. Four representative algorithms from each of the three categories are selected for this purpose.

Cloud/microservice performance anomaly detection research typically does not account for the constraints of edge computing environments and, therefore, only focuses on evaluating the accuracy of anomaly detection models. All those studies report only accuracy-related metrics such as precision, recall and F1-score [23, 78]. However, due to the resource-constrained nature of edge devices, edge performance anomaly detection studies evaluate both accuracy and efficiency during inference (in terms of resource utilization and detection time) when identifying a suitable performance anomaly detection algorithm to be deployed at the edge devices [11, 16, 18]. Consequently, in section 4.4.2, we evaluate the accuracy as well as resource utilization efficiency of four representative algorithms from each category of Audibert's taxonomy to determine the most suitable algorithm for the remainder of our evaluations.

In addition to evaluating the accuracy and efficiency of the algorithms during detection, considering the fact that these models are trained on resource-constrained edge devices, some of these studies have also assessed the time taken and resource utilization during the training process [12, 16]. Although Becker et al. do not evaluate the above aspect, they also propose training a model at the edge device [11]. As discussed in section 4.1, while this model per device (MPD) approach yields high accuracy, the presence of a large number of edge devices leads to a high aggregate training time and an increased total consumption of resources during the training process. On the other hand, Soualhia et al. propose training a generic anomaly detection model (GM) using data from all devices [12]. While this approach can be efficiently trained, it results in low accuracy. Notably, the GM approach used by Soualhia et al. aligns with the training strategies typically employed in AI-based performance anomaly detection studies in cloud environments and general microservices, which often overlook the impact of resource constraints during the training phase.

A comparison between prior works on performance anomaly detection in edge computing environments [11, 12, 16, 18] along with our proposed work is shown in Table 4.1. Under the "Anomaly Detection Evaluation Aspects" column of the table, checkmarks and crosses represent whether that particular study has considered accuracy and/or efficiency when conducting evaluations. Notably, none of these studies have tried to reach a tradeoff between the efficiency and accuracy of performance anomaly detection model training, which presents a gap in current research that we aim to explore further in this chapter.

4.2.2 Machine learning model training approaches in edge computing environments

Since none of the edge performance anomaly detection studies have developed approaches that strike a balance between the efficiency and accuracy of performance anomaly detection model training, we reviewed the literature on ML model training for Artificial Intelligence of Things (AIoT) applications as well [19–22]. Our goal was to identify the model training approaches used in these works. John et al. [19] and Raj et al. [20] discussed the generic and specialized model training approaches that we could also identify from edge performance anomaly detection literature, while Peltonen et al. [21]

and Psaromanolakis et al. [22] focused on training a generic model. As mentioned before, since these two training approaches achieve either high accuracy or high efficiency, we use them as the baselines for comparing our proposed clustering-based training approaches.

Additionally, a few works employ novel ML approaches, such as federated learning [143] and distributed ML [144], for model training in edge computing environments. Schneible et al. utilized federated learning in the context of supervised anomaly detection [143]. Since different edge devices encounter different anomaly data, they distribute the required updates via a centralized model to train the anomaly detection models of other edge devices with newly seen anomaly data. However, adopting this approach for unsupervised performance anomaly detection would result in each model learning the normal data distributions of all microservices deployed across all edge devices, ultimately leading to lower model accuracy.

On the other hand, Yang et al. [144] used distributed ML to detect network anomalies at edge devices by analyzing network packets. Their approach involves centrally training an ML model using multiple interconnected computing resources and then deploying it to infer anomalies at the edge device level. Although this approach makes the training process faster by parallelizing and utilizing more resources, it results in a generic model trained on data aggregated from all edge devices. This can lead to reduced accuracy when applied to performance anomaly detection model training, as the generic model is not capable of generalizing well across the heterogeneous normal data distributions of all microservices.

Thus, adapting ML approaches such as federated learning and distributed ML for training performance anomaly detection models results in the creation of a generic model where every model learns the normal data distribution of microservices deployed in other devices as well. Although this approach results in high accuracy during situations where all the collected data belong to the same independent and identically distributed (IID) distribution, in our case, since the data collected from different microservices belong to non-IIDs, this approach could drift the model away from expected behavior, resulting in low accuracy. Therefore, it is clear that simply adapting existing ML approaches to perform anomaly detection model training in edge computing environments does not allow us to strike a balance between efficiency and accuracy. Therefore, in this chapter, we propose a new approach to training anomaly detection models tailored to the specific characteristics and constraints of edge environments with the aim of increasing training efficiency while achieving high model accuracy.

4.3 Clustering-based Training Approaches

This study explores a hierarchical edge environment with multiple heterogeneous devices [136]. If there are N edge devices, the set of edge devices can be defined as E, where e_i represents the i^{th} edge device.

$$E = \{e_1, e_2, ..., e_N\} \tag{4.1}$$

Each device is equipped with a monitoring agent that gathers M performance and resource consumption metrics from each microservice deployed on that device. Consistent with prior research on performance anomaly detection at the edge [11, 12, 16, 22], this study also presumes that an unsupervised multivariate anomaly detection model, which analyzes M performance metrics, is deployed at each edge device. Normal performance data (i.e., data collected under non-anomalous conditions) is used to train unsupervised models, and the set of training data collected from device e_i for T timesteps can be defined as matrix D_{e_i},

$$D_{e_i} = [d_{m,t}]_{M \times T} \tag{4.2}$$

For each edge device e_i, the Model per Device (MPD) baseline approach trains a model by taking D_{e_i} as input. The approach optimizes the model parameters w by minimizing the loss function L. The resulting optimal model parameters are denoted as $w_{e_i}^{mpd}$, as shown in equation 4.3. Since a model is trained per edge device and microservices within a specific edge device are expected to have similar normal metric distributions (due to the QoS and resource requirement-aware placement of microservices) [9, 98], the accuracy of these models is expected to be high. However, because a large number of models need to be trained, this approach is not very efficient.

$$w_{e_i}^{mpd} = argmin_w L(w; D_{e_i}) \tag{4.3}$$

On the other hand, the generic model (GM) baseline approach takes the data belonging to all edge devices, denoted as D_E, as input and trains a single model by minimizing the loss function L to optimize the model parameters w, as shown in equation 4.4. The resulting optimal model parameters are denoted as w^{gm}. A single generic model compromises accuracy in favor of training efficiency; a single model is unlikely to generalize well across the heterogeneous normal data distributions of all microservices in an edge environment, yet can be trained with fewer resources when compared to training multiple specialized models.

$$w^{gm} = argmin_w L(w; D_E) \tag{4.4}$$

Since the above-explained baseline model training approaches are extremes in terms of model accuracy and training efficiency, it is important to reach a trade-off between accuracy and efficiency during the training process. This aim can be achieved by training a few models on groups of similar data, i.e., the heterogeneity of data used to train a single model should be minimal. To accomplish this, we use the concept of clustering to group edge devices with similar normal data distributions and then train the models within these clusters. Towards that, we exploit the QoS-aware placement of microservice-based IoT applications [9, 10, 98] to cluster edge devices with similar normal data distributions. Under a QoS-aware placement strategy, we assume that microservices with similar normal data distributions are placed in the same edge device or edge devices with similar computing, storage, and networking capabilities. Clustering is performed based on edge devices since the smallest unit of model deployment is the edge device level. Towards that, in section 4.3.1, we introduce a similarity metric capable of clustering edge devices with similar normal data distributions, followed by sections 4.3.2 and 4.3.3, where we propose the two clustering-based training approaches, Intra-cluster parameter transfer learning (ICPTL)-based model training and Cluster-level model (CM) training, respectively. Finally, in section 4.3.4, we discuss the positioning of the proposed approaches and the workflow for initial model deployment.

4.3.1 Similarity-based clustering

In this context, we assume that microservices are placed based on their QoS and resource requirements. Under such a QoS-aware placement strategy, we assume that microservices with similar normal data distributions are placed in the same edge device or edge devices with similar computing, storage, and networking capabilities. As a result, microservices deployed at different hierarchical levels show a variance in metrics. This variance is more pronounced in some metrics (e.g., CPU and memory consumption metrics) than in others (e.g., error counts, disk read/write throughput). For instance, microservices situated closer to the cloud generally have higher values for CPU and memory consumption metrics and lower values for latency than those closer to the IoT layer. Hence, we identify a subset of H such metrics, representative of the QoS and resource usage of microservices, suitable to perform device clustering from the M metrics.

The first step towards cluster formation is calculating the similarity between each pair of devices, as proposed in Algorithm 2. The proposed algorithm takes as input normal datasets collected from all edge devices $D_E = \{D_{e_1}, D_{e_2}, .., D_{e_i}, .., D_{e_N}\}$ and the list of H representative metrics. It first extracts the H representative metrics from each dataset D_{e_i}, resulting in $D^*_{e_i} = [d_{h,t}]_{H \times T}$. Next, each metric of D^*_E is standardized using min-max normalization as shown in step 5 of Algorithm 2. Subsequently, the algorithm generates a multivariate probability distribution P_{e_i} composed of the probability distributions for each of the H representative metrics, for each device e_i.

Once the multivariate probability distributions for all devices are available, the similarity distance $sim\_dist(e_i, e_j)$ between each pair of devices e_i and e_j is calculated by obtaining the L2-norm of the $JS\_distance$ between each representative metric of the probability distributions P_{e_i} and P_{e_j} corresponding to those devices. The equation corresponding to this step is shown in equation 4.5. The $JS\_distance$ between the probability distributions $p_{e_{i,h}}$ and $p_{e_{j,h}}$ corresponding to a single metric h, is obtained by calculating the square root of their $JS\_divergence$, as shown in equation 4.6. The $JS\_divergence$ between any two probability distributions, $\mathbb{P}$ and $\mathbb{Q}$, is the weighted sum of the forward and backward Kullback-Leibler (KL) divergence between those probability distributions (refer to equation 4.7). The equation for calculating the KL divergence $KL(\mathbb{P} \parallel \mathbb{Q})$ between any two probability distributions, $\mathbb{P}$ and $\mathbb{Q}$, is shown in equation 4.8 [145].

Algorithm 2: Similarity Graph Formation Algorithm

```
1: $Input$ : Normal datasets collected from all edge devices
   $D_E = \{D_{e_1}, D_{e_2}, .., D_{e_i}, .., D_{e_N}\}$
2: $Input$ : List of $H$ representative metrics
3: $Output$ : Similarity graph $SG$
4: Compose $D^*_E = \{D^*_{e_1}, D^*_{e_2}, .., D^*_{e_i}, .., D^*_{e_N}\}$ (where $D^*_{e_i} = [d_{h,t}]_{H \times T}$) by extracting the $H$
   representative metrics
5: Standardize $D^*_E$ s.t. $D^*_E[h,:] \leftarrow \frac{D^*_E[h,:]}{max(D^*_E[h,:]) - min(D^*_E[h,:])}, \quad \forall h; 0 < h \leq H$
6: Obtain $P_E = \{P_{e_1}, P_{e_2}, .., P_{e_i}, .., P_{e_N}\}$ using $D^*_E$, where $P_{e_i} = [p_{e_{i,1}}, p_{e_{i,2}}, .., p_{e_{i,H}}]$ is the
   multivariate probability distribution for each edge device $e_i$
7: Initialize similarity graph $SG = (\mathbb{V}, \mathbb{E})$, where $\mathbb{V} = [e_i, \forall i; 0 < i \leq N]$ and $\mathbb{E} = \emptyset$
8: for each device $e_i; 0 < i \leq N$ do
9:    for each device $e_j; i < j \leq N$ do
10:       $sim\_dist(e_i, e_j) = \sqrt{\sum_{h=1}^{H} JS\_distance(p_{e_{i,h}}, p_{e_{j,h}})^2}$
11:       $\mathbb{E} \leftarrow \mathbb{E} \bigcup \{(e_i, e_j, sim\_dist)\}$
12:    end for
13: end for
14: $Return$ : $SG$
```

$$sim\_dist(e_i, e_j) = \sqrt{\sum_{h=1}^{H} JS\_distance(p_{e_{i,h}}, p_{e_{j,h}})^2} \tag{4.5}$$

$$JS\_distance(p_{e_{i,h}}, p_{e_{j,h}}) = \sqrt{JS\_divergence(p_{e_{i,h}} || p_{e_{j,h}})} \tag{4.6}$$

$$JS\_divergence(\mathbb{P}||\mathbb{Q}) = \frac{1}{2} KL\left(\mathbb{P}||\frac{\mathbb{P}+\mathbb{Q}}{2}\right) + \frac{1}{2} KL\left(\mathbb{Q}||\frac{\mathbb{Q}+\mathbb{P}}{2}\right) \tag{4.7}$$

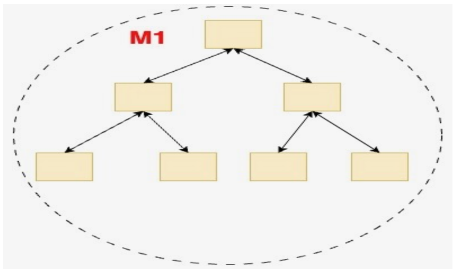
$$KL(\mathbb{P} \parallel \mathbb{Q}) = \sum_i \mathbb{P}(i) \log\left(\frac{\mathbb{P}(i)}{\mathbb{Q}(i)}\right) \tag{4.8}$$

Algorithm 3: Similarity-based Device Clustering Algorithm

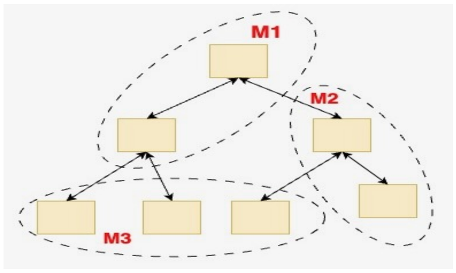
```
1: Input : Similarity graph $SG = (\mathbb{V}, \mathbb{E})$ (output of Algorithm 2)
2: Input : No.of clusters $K$
3: Output : Cluster to device mapping $cluster\_map$
4: Initialize $cluster\_map = \{c_k \rightarrow [e_k], \quad \forall k; 0 < k \leq N\}$
5: while $length(cluster\_map) > K$ do
6:     Find $e_{i,j}^{min} = (e_{i^*}, e_{j^*}, w_{i^*,j^*})$ s.t. $(e_{i^*}, e_{j^*}, w_{i^*,j^*}) = \underset{(e_i,e_j,w_{i,j}) \in \mathbb{E}}{\operatorname{argmin}} w_{i,j}$
7:     $c_i = cluster\_map^{-1}(e_i)$ // Look up the cluster to which the device $e_i$ belongs to
8:     $c_j = cluster\_map^{-1}(e_j)$ // Look up the cluster to which the device $e_j$ belongs to
9:     $c_i \leftarrow c_i \bigcup c_j$
10:    $cluster\_map \leftarrow cluster\_map \setminus c_j$
11:    $\mathbb{E} \leftarrow \mathbb{E} \setminus e_{i,j}^{min}$
12: end while
13: Return : $cluster\_map$
```

The output of Algorithm 2 is a complete graph, where a vertex represents an edge device e_i, and the weight of the edge between each pair of vertices e_i and e_j is the similarity distance $sim\_dist(e_i, e_j)$ between the corresponding edge devices. This graph is hereafter referred to as the similarity graph SG.

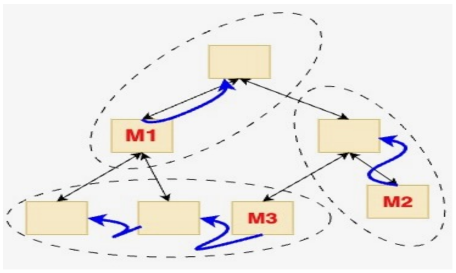
The similarity graph SG produced by Algorithm 2 is then used to generate clusters of devices with similar normal data distributions, as depicted in Algorithm 3. Furthermore, the number of clusters K is a hyperparameter and is required as input for this algorithm. In the first step, the *cluster_map*, which stores the mapping from devices to clusters, is initialized such that each edge device e_k is assigned to a unique cluster c_k. Therefore, initially, there are as many clusters as edge devices. Next, the algorithm identifies vertices corresponding to the shortest edge of the similarity graph SG, i.e., the edge devices e_i, e_j with the highest similarity, and merges clusters containing those edge devices as depicted in steps 6 to 11 of Algorithm 3. These steps are repeated until there are K clusters in the *cluster_map*. This algorithm, inspired by Kruskal's algorithm [146], ensures that clusters are created by grouping together devices corresponding to shorter edges. It returns the *cluster_map* with K clusters as the output. Through the proposed similarity-based clustering approach, we expect that edge devices having microservices with similar normal data distributions fall into the same cluster.



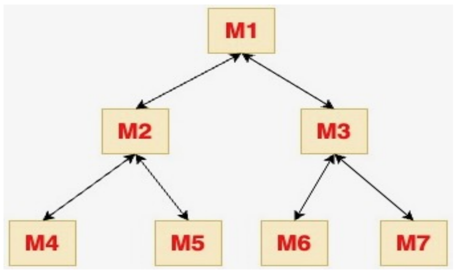
(a) Generic model



(b) Cluster-level model training



(c) Intra-cluster parameter transfer learning-based model training



(d) Model per device

Figure 4.1: Visual representation of the two clustering-based training approaches and the two baseline approaches

4.3.2 Intra-cluster parameter transfer learning(ICPTL)-based model training

The aim of the ICPTL approach is to achieve a similar level of accuracy as the MPD approach but with fewer training cycles. In the MPD approach, each anomaly detection model needs to learn the normal data distribution from its respective edge device. However, the normal data distributions among edge devices within the same cluster only differ slightly. Therefore, in the ICPTL approach, instead of training the model for each edge device from scratch, we utilize the concept of parameter transfer learning, which involves transferring the weights learned by one model (the source model) and its hyperparameters to initiate the training of a model for another edge device (the target model) [147]. For instance, if the source device is e_i and the target device is e_j, the ICPTL approach takes dataset D_{e_j} and model parameters of source model w_{e_i} as inputs and trains a target model by minimizing the loss function L to optimize the model parameters w. The resulting optimal model parameters are denoted as $w_{e_j}^{icptl}$, as shown in equation 4.9. This allows the target model to have an advantage at the beginning of its

Algorithm 4: Intra-cluster Parameter Transfer Learning-based Model Training Algorithm

```
1: Input : Similarity graph $SG = (\mathbb{V}, \mathbb{E})$ (output of Algorithm 2)
2: Input : Cluster to device mapping $cluster\_map$ (output of Algorithm 3)
3: Output : Models for all devices
4: for each cluster $c_k; 0 < k \leq K$ do
5:   Initialize $source\_set = \emptyset$
6:   Initialize $target\_set = cluster\_map[c_k]$
7:   Extract $SG_k = (\mathbb{V}_k, \mathbb{E}_k)$ where $\mathbb{V}_k = cluster\_map[c_k]$ and
     $\mathbb{E}_k = \{(e_i, e_j, w_{i,j}) \in \mathbb{E}; e_i, e_j \in \mathbb{V}_k\}$
8:   Find $e^{min}_{i,j} = (e_{i^*}, e_{j^*}, w_{i^*,j^*})$ s.t. $(e_{i^*}, e_{j^*}, w_{i^*,j^*}) = \underset{(e_i,e_j,w_{i,j}) \in \mathbb{E}_k}{\text{argmin}} \; w_{i,j}$
9:   $source \leftarrow e_i$
10:  $target \leftarrow e_j$
11:  $w_{source} = \text{argmin}_w L(w; D_{source})$
12:  $source\_set \leftarrow source\_set \bigcup source$
13:  $target\_set \leftarrow target\_set \setminus source$
14:  while $target\_set \neq \emptyset$ do
15:    $w_{target} = \text{argmin}_w L(w; D_{target}, w_{source})$
16:    $source\_set \leftarrow source\_set \bigcup target$
17:    $target\_set \leftarrow target\_set \setminus target$
18:    $\mathbb{E}_k \leftarrow \mathbb{E}_k \setminus e^{min}_{i,j}$
19:    Find $e^{min}_{i,j} = (e_{i^*}, e_{j^*}, w_{i^*,j^*})$ s.t. $(e_{i^*}, e_{j^*}, w_{i^*,j^*}) = \underset{(e_i,e_j,w_{i,j}) \in \mathbb{E}_k}{\text{argmin}} \; w_{i,j}$ and $e_i \in source\_set$
20:    $source \leftarrow e_i$
21:    $target \leftarrow e_j$
22:  end while
23: end for
24: Return : Models for all devices
```

training, as the source model has already learned the lower-level features required by the target model, thus reaching a satisfactory accuracy within a few epochs.

$$w^{icptl}_{e_j} = argmin_w L(w; D_{e_j}, w_{e_i}) \tag{4.9}$$

Algorithm 4 comprises the steps of the ICPTL approach. It consumes the similarity graph SG, which is the output of Algorithm 2 and the cluster to device mapping *cluster_map* which is the output of Algorithm 3 as inputs. For a given cluster c_k, the *source_set*, which stores the source devices with already trained models and the *target_set*, which stores the devices where the models are yet to be trained are first initialized. Next, the subgraph SG_k, which consists of edge devices in c_k as vertices, is extracted from the similarity graph SG. From SG_k, one of the vertices corresponding to the shortest edge, i.e., one of the edge devices e_i from the pair exhibiting the highest similarity, is arbitrarily selected as the *source* device. Next, a model is trained for that device by following the MPD approach as shown in step 11 of the algorithm. After training, the *source* device e_i is moved to the *source_set*. Thereafter, the model for the remaining device e_j (i.e., the *target* device) is trained using parameter transfer learning as shown in equation 4.9, and it also corresponds to step 15 of the algorithm. After this step, the *target* device e_j is also moved to the *source_set*. Subsequently, as shown in step 19, the algorithm identifies the next shortest edge from the remaining edges, such that the edge device corresponding to one of its vertices belongs to the *source_set*. Next, by utilizing the model for that edge device as the source model, the model for the edge device corresponding to the other vertex is trained using parameter transfer learning. These steps are repeated until the *target_set* becomes empty. At the end of the algorithm, a trained model will be available for all edge devices within c_k. Moreover, ICPTL-based model training can be performed simultaneously across clusters.

Figure 4.1(c) visually represents the ICPTL approach. It essentially identifies a Minimum Spanning Tree (MST) of the cluster as the sequence to perform parameter transfer learning among the edge devices. In the identified sequence, a model is traditionally trained only for the device at the starting point of the sequence, while the other devices in the sequence receive the model from the nearest neighbor device and train a few epochs until they reach satisfactory accuracy. While this approach entails training a model for each edge device, it requires fewer epochs during training compared to the MPD approach, thus resulting in reduced training resource requirements and training time.

4.3.3 Cluster-level model (CM) training

Through the CM approach, we aim to improve training efficiency further by reducing the number of models. To that end, we propose to train a model per cluster using normal data collected from all edge devices in the cluster. The CM approach takes the data belonging to all edge devices in cluster c_k, denoted as D_k, as input and trains a model for that cluster by minimizing the loss function L to optimize the model parameters w as shown in equation 4.10. The resulting optimal model parameters are denoted as w_k^{cm}. The CM approach is visually represented in Figure 4.1(b).

$$w_k^{cm} = argmin_w L(w; D_k) \tag{4.10}$$

Since similarity-based clustering ensures that devices having similar normal data distributions fall into one cluster, the model trained for each cluster can effectively generalize across all the edge devices in that cluster. This cluster-specific approach is expected to be more accurate than the generic model (GM) approach.

Since the number of clusters is always less than or equal to the number of edge devices, this means that fewer models need to be trained when using the CM training approach compared to the MPD approach. This results in reduced training time and resource requirements. Additionally, it is easier to manage a few models per cluster than a relatively large number of models per edge device. As a result, the CM approach is expected to be more efficient than the MPD approach.

4.3.4 Positioning of the proposed approaches and initial model deployment workflow

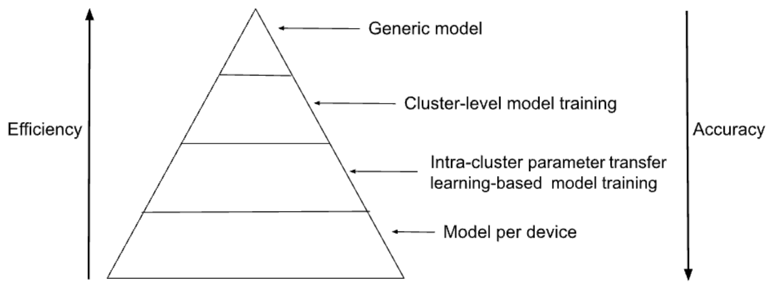

Efficiency
Generic model
Cluster-level model training
Intra-cluster parameter transfer learning-based model training
Model per device
Accuracy

Figure 4.2: Positioning of the two proposed clustering-based training approaches

Out of the two proposed approaches, the CM training approach is designed to provide more efficiency (in terms of training time, training resource requirements, number of epochs, and model management overhead) compared to ICPTL approach, while

ICPTL is designed to provide more accuracy than CM. Figure 4.2 depicts the positioning of the two proposed clustering-based training approaches amidst the two baseline model training approaches in terms of accuracy and efficiency.

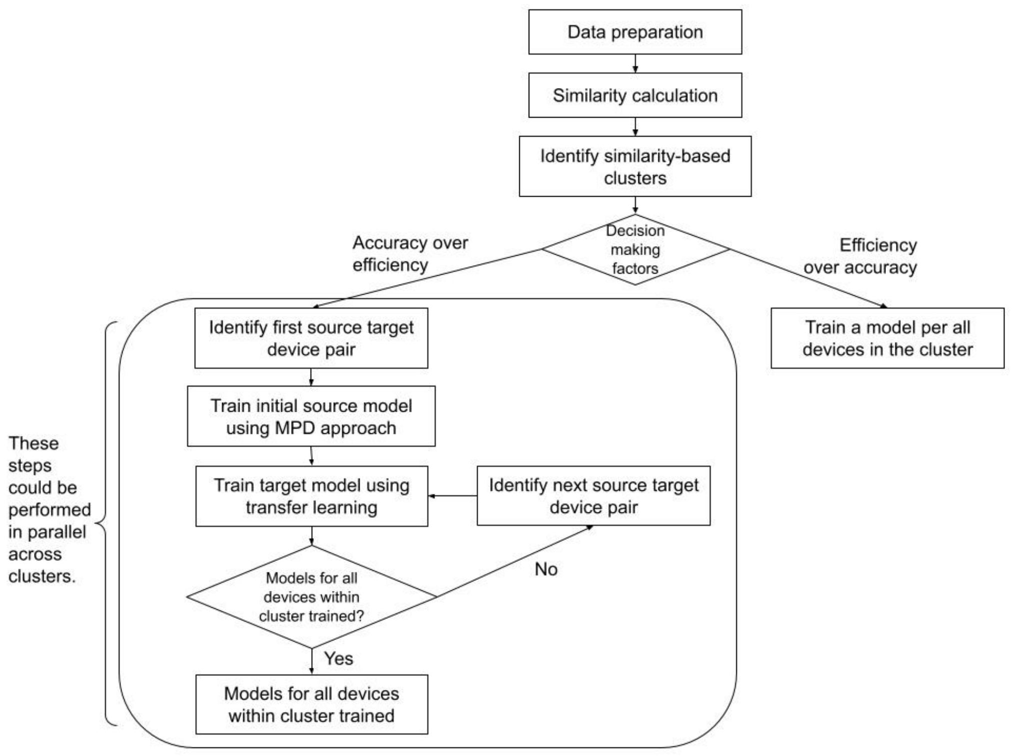

Data preparation
Similarity calculation
Identify similarity-based clusters
Decision making factors
Accuracy over efficiency
Efficiency over accuracy
Train a model per all devices in the cluster
Identify first source target device pair
Train initial source model using MPD approach
Train target model using transfer learning
Identify next source target device pair
Models for all devices within cluster trained?
No
Yes
Models for all devices within cluster trained
These steps could be performed in parallel across clusters.

Figure 4.3: Initial model deployment workflow

Figure 4.3 illustrates the workflow for training models during their initial deployment in edge computing environments. Following the initial cluster generation steps (outlined in Algorithms 2 and 3), the service provider can choose between the two proposed clustering-based training approaches. For scenarios prioritizing efficiency—such as reduced training time, lower training resource requirements, fewer epochs, and minimal model management overhead—the CM approach is recommended. Conversely, if higher accuracy is the primary objective, the ICPTL approach is more suitable.

4.4 Performance Evaluation

In this section, we discuss the evaluation results of the proposed clustering-based training approaches. First, we explain the details of the experimental setup used for evaluation. Following that, we compare four common unsupervised multivariate time-series anomaly detection techniques to identify the most suitable algorithm for the rest of our evaluations. Finally, we discuss the results of evaluating the two proposed clustering-based training approaches using the algorithm identified from the previous step against the baseline approaches derived from the state of the art.

4.4.1 Experimental Setup

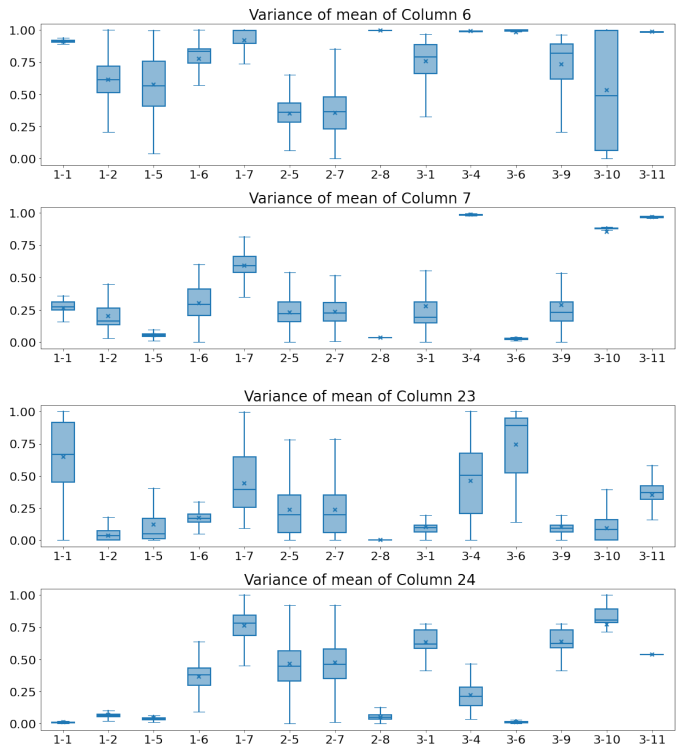


Figure 4.4: Distribution of normal data across SMD devices

In order to conduct a comprehensive and reproducible evaluation of the proposed approaches, we utilized a public performance anomaly dataset called the "Server Machine Dataset" (SMD), which is widely used in the anomaly detection literature [72, 78, 85]. The SMD contains both normal (i.e., data collected under non-anomalous conditions) and anomalous performance data (i.e., data collected under anomalous conditions) from 28 devices across 38 metrics. Following the suggestion of Li et al. [137], we focused on 14 devices without a concept drift between training (normal performance) data and testing (anomalous performance) data for our evaluation. The SMD dataset has also been utilized in assessing research on edge computing environments [16, 17], as the data from these devices exhibit heterogeneous normal data distributions akin to those in edge computing environments. This is apparent from the significant variance in mean values of certain selected metrics across the majority of devices, as shown in Figure 4.4. In our evaluation, we assume that each SMD machine is an edge device with a single microservice deployed in it and that the data reported for each machine (edge device) belongs to its respective microservice.

In order to further verify the results obtained on SMD, we also collected performance anomaly data from a set of microservices with heterogeneous QoS and resource requirements deployed in an emulated edge computing environment. By using the iAnomaly toolkit in chapter 3, we emulated 10 heterogeneous edge devices using 10 VMs with different resource capacities (2vCPU/8GB, 4vCPU/16GB, 8vCPU/32GB) on Melbourne Research Cloud[1] and emulated the network (by following the communication link parameters used by Pallewatta et al.[136] in their experiments) to represent the heterogeneity of network bandwidth between the edge devices. The iAnomaly toolkit is a full-system emulator with iContinuum [131] (a well-tested edge emulator for discrepancies compared to real-world scenarios) at its core. In addition to that, iAnomaly has extensively verified that both the normal and anomalous data that it generates closely resemble the data generated from real edge environments. We deployed microservices with heterogeneous resource requirements representing a diverse set of IoT applications (from lightweight sensor data processing to heavy applications like camera face detection/recognition) on the emulated edge infrastructure by following the placement algorithm proposed by Pallewatta et al.[9]. Using Pixie[2], which is a lightweight monitoring tool suitable for edge monitoring, we collected both normal performance data as well

[1]https://dashboard.cloud.unimelb.edu.au/
[2]https://docs.px.dev/

as anomalous performance data under a diverse set of anomalies, such as resource saturation, service failures, network saturation, etc., at a granularity of 10 seconds. Normal workloads were generated using Jmeter[3] while anomalies were injected using Chaos Mesh[4]. Both normal and anomalous performance data are available for 12 metrics, covering both system and application metrics.

Experiments on both datasets, including hyperparameter tuning, were conducted on the Spartan[5] High Performance Computing (HPC) cluster. More details on training, including the hyperparameters selected for training, are explained in sections 4.4.2, 4.4.3, and 4.4.4.

4.4.2 Evaluating performance anomaly detection algorithms against the SMD dataset

Prior to evaluating the proposed clustering-based training approaches using the SMD dataset, in this section, we compare four common unsupervised multivariate time-series anomaly detection techniques to identify the most suitable algorithm for the rest of our evaluations in terms of accuracy and efficiency during inference.

As mentioned in section 4.2.1, while research on performance anomaly detection in edge computing environments is still emerging, AI-based performance anomaly detection techniques are well-established for cloud and microservices environments [78, 85, 139–142]. Audibert et al. [78] classify anomaly detection algorithms into three categories: (1) conventional methods, (2) machine learning (ML)-based methods, and (3) deep neural network (DNN)-based methods. Research on performance anomaly detection in edge environments often adapts cloud-based techniques [11, 12, 16, 18], benchmarking them across these categories. Therefore, in this section, we select four representative performance anomaly detection algorithms from these categories to identify the most suitable algorithm for further evaluation.

We selected the Vector AutoRegression (VAR) algorithm from conventional methods because it extends ARIMA to multivariate data, making it highly effective for time

[3]https://jmeter.apache.org/
[4]https://chaos-mesh.org/
[5]https://dashboard.hpc.unimelb.edu.au/

series modelling. ARIMA has consistently outperformed other techniques in multiple studies, including those by Soualhia et al. [12], Becker et al. [11], and Pitakrat et al. [148]. Furthermore, as highlighted by Audibert et al. [78], VAR is widely recognized as one of the most commonly used techniques for multivariate time series anomaly detection. Isolation Forest (IF) is a widely used ML algorithm frequently employed in anomaly detection studies [16, 72, 85, 139, 142]. It has also been identified as the top-performing technique in studies by Kardani et al. [139] and Ramamoorthi et al. [142]. These factors led us to select IF to represent the "ML-based methods" category. We selected two algorithms representing DNN-based methods: Autoencoder (AE) and LSTM-AutoEncoder (LSTM-AE). AE, as well as approaches that have extended the AE concept [16, 85, 149, 150], are a class of popular DNN-based methods. AE has also been recognized as the top-performing method in studies conducted by Audibert et al. [85] and Borghesi et al. [150]. In addition to AE, we also considered LSTM-AE, since LSTM layers are capable of capturing time-series dependencies better [11, 72, 78, 140]. Such approaches where AEs are enhanced with RNN layers have been identified as top-performing techniques in several studies conducted by Islam et al. [151, 152].

Therefore, we select the following four representative performance anomaly detection algorithms from each category.

1. AutoEncoder (AE) - DNN-based methods
2. Isolation Forest (IF) - ML-based methods
3. LSTM-AutoEncoder (LSTM-AE) - DNN-based methods
4. Vector AutoRegression (VAR) - conventional methods

We implemented the VAR algorithm using the Statsmodels[6] library. For the IF implementation, we began with the original approach provided by the Scikit-learn library for IF[7]. However, since it only considers the spatial dependencies among data points when performing anomaly detection, we modified it following Kardani et al. [139] who have incorporated the k-point moving average into the feature space of each sample. It

[6]https://www.statsmodels.org/
[7]https://scikit-learn.org

captures the temporal dependencies of the performance data by obtaining the average of a window of k-previous samples. The AE and LSTM-AE ML models were implemented using the Pytorch[8] library. We employed the Tree-structured Parzen Estimator [153], which is a Bayesian optimization technique, to tune the hyperparameters listed in Table 4.2 corresponding to each algorithm.

Table 4.2: Hyperparameters tuned for each anomaly detection algorithm

Machine Learning Model	**Tuned Hyperparameters**
AE	num_layers, window_size, hidden_size, batch_size, learning_rate
IF	n_estimators, max_features, max_samples, k (for the k-point moving average)
LSTM-AE	num_layers, window_size, batch_size, learning_rate

Table 4.3: Evaluation results (F1-score, AUC) of anomaly detection algorithms

	AE	**IF**	**LSTM-AE**	**VAR**
AUC	**0.89**	0.84	0.86	0.74
F1-score	**0.79**	0.69	0.73	0.50

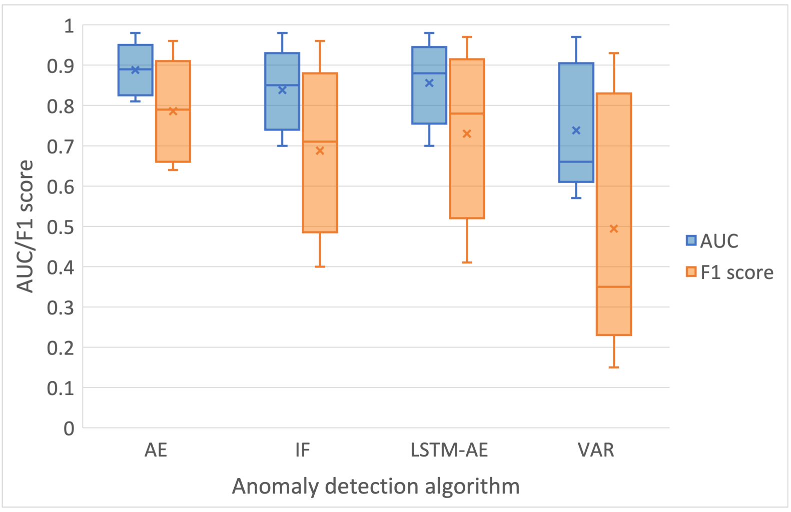


Figure 4.5: Distribution of AUC and F1 score of anomaly detection algorithms across devices in SMD

We use F1-score and AUC to evaluate the accuracy of performance anomaly detection algorithms similar to other works in the domain [139, 154]. F1-score is the harmonic

[8]https://pytorch.org/

mean between precision and recall, while AUC refers to the area under the receiver operating characteristic curve.

We evaluate the four anomaly detection algorithms on a random subset of devices of the SMD dataset by following the MPD approach. Table 4.3 shows the evaluation results in terms of average AUC and F1-scores. Figure 4.5 shows the distribution of AUC and F1-scores across devices for each anomaly detection algorithm. AE achieves the highest mean AUC (0.89) and F1-score (0.79) out of all algorithms. As shown in Figure 4.5, AE also has the lowest variance.

During our evaluation, we consider both the accuracy and efficiency of the four algorithms. The VAR algorithm does not require a training phase, but it has a high inference time. This results in high latency when detecting anomalies. It also has the least mean and highest variance for its AUC and F1-scores among the compared algorithms. Therefore VAR is the least suitable anomaly detection algorithm in terms of accuracy and efficiency. IF is a low overhead algorithm with low linear-time complexity and small memory requirements. However, IF-based approaches require a certain percentage of anomaly data for training to achieve their best possible accuracy [139]. This is also evident through the low mean and high variance of AUC and F1-scores obtained in this evaluation, where the IF algorithm was trained only on normal data. Out of the remaining two algorithms, AE has the best mean and least variance for its AUC and F1-scores. LSTM-AE also has a nearly equal mean but higher variance on its AUC and F1-scores than AE. LSTM-AE does not outperform the AE algorithm, although it is designed to capture time-series dependencies. Out of these two DNN-based approaches, when considering efficiency, the AE model is less complex, and its inference can be parallelized, unlike an LSTM-AE model. Therefore, we conclude that AE is the most suitable performance anomaly detection algorithm for the rest of our evaluations. AE models used in the rest of the evaluations are trained by minimizing the Mean Squared Error (MSE) loss between the original and reconstructed windows using the Adam optimizer. However, the proposed clustering-based training approaches are independent of the ML approach. Hence, a reader can use any other independent model to evaluate the proposed training approaches.

4.4.3 Evaluating clustering-based training approaches using the SMD dataset

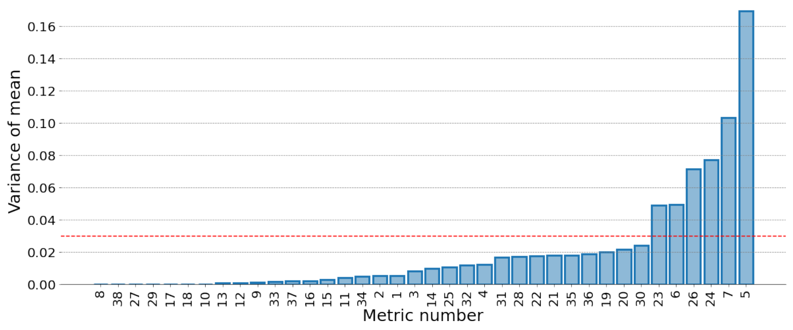


Figure 4.6: Variance of mean of each metric across SMD devices

In this section, we evaluate the proposed clustering-based training approaches on the SMD dataset. As the first step, we assign edge devices to clusters. However, since metrics are undefined in this dataset, we can not use domain knowledge to select the subset of metrics suitable for performing device clustering. Therefore, we use the variance of mean to identify significantly varying metrics across devices to identify the subset of metrics. The variance of mean represents how much the mean of each metric varies across devices. Figure 4.6 is a bar chart that represents the variance of mean of each metric across devices. As can be seen, columns 23, 6, 24, 26, 7, and 5 have a high variance of mean. However, column 5 has all-zero occurrences in 9 out of 14 devices. Hence, we leave it out of consideration for clustering.

Next, we obtain the collinearity between those metrics. Figure 4.7 contains the pairplot, which shows the collinearity between metrics shortlisted from the previous step. Since 24 and 26 are linearly correlated, one of those (26) was dropped from the set of metrics for clustering.

Then, we clustered the 14 devices based on the final set of metrics using Algorithms 2 and 3. All the algorithms corresponding to similarity-based clustering and clustering-based training approaches were implemented using Python. The same AE model implementation mentioned in section 4.4.2 was used for evaluations in this section as well. We employed the Tree-structured Parzen Estimator [153] Bayesian optimization technique, to optimize the hyperparameters of the AE model listed in Table 4.2 as well as to determine the optimal number of clusters (K) for Algorithm 3. Subsequent to hyperparameter

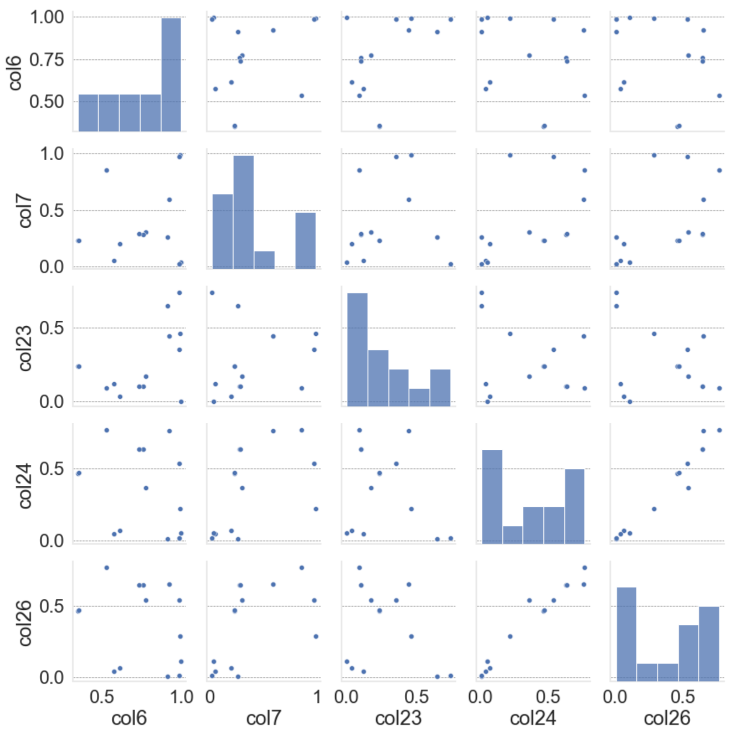


Figure 4.7: Collinearity between shortlisted metrics across SMD devices

tuning, the optimal number of clusters, K was identified to be five.

Next, we present the evaluation results for the two proposed clustering-based training approaches.

Table 4.4: Evaluation results of training approaches against SMD

	MPD	**ICPTL**	**CM**	**GM**
AUC (mean)	0.88	0.85	0.81	0.66
F1-score (mean)	0.77	0.7	0.65	0.35
Total training time	19min 40secs	8min 2secs	6min 8secs	47secs

Table 4.4 contains the accuracy (in terms of average AUC and F1-scores) and training time results of the two proposed clustering-based model training approaches and the two baseline approaches. Figure 4.8 shows the distribution of AUC and F1-scores across these training approaches. As predicted, the MPD approach yielded the highest average AUC (0.88) and F1-score (0.77), but it required the longest total training time (19min 40secs). On the other hand, the GM approach had the lowest average AUC (0.66) and F1-score (0.35) with the shortest total training time (47secs). As expected, the ICPTL

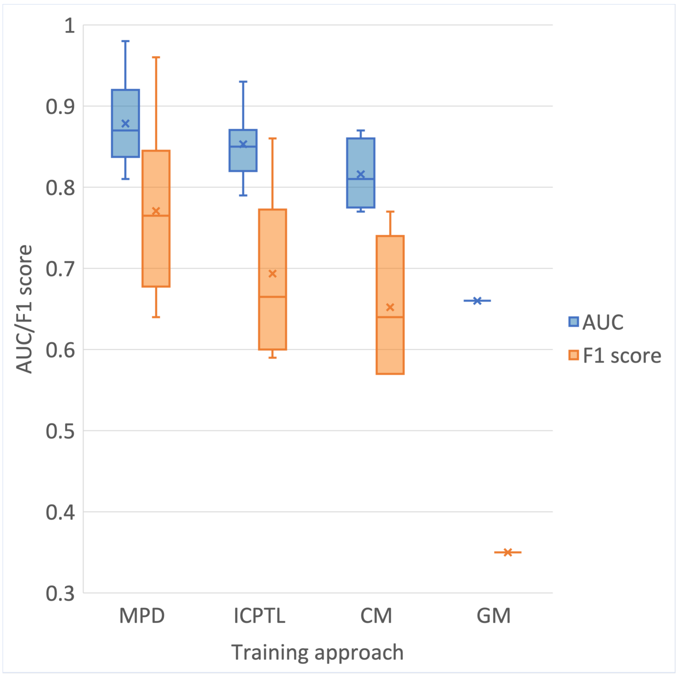
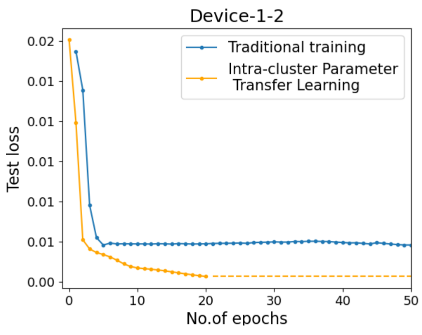


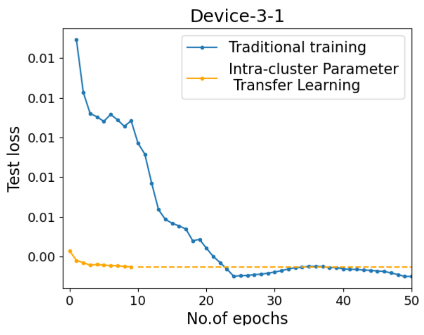
Figure 4.8: Distribution of AUC and F1 score of training approaches against SMD

approach had the second-highest mean AUC (0.85) and F1-score (0.7) while consuming only 40% of the total training time required by the MPD approach (8min 2secs). The CM approach has further increased training efficiency by consuming 23% less training time than the ICPTL approach (6min 8secs) while reducing the total number of models from 14 to 5. Furthermore, the CM approach is 16% and 30% more accurate in AUC (0.81) and F1-score (0.65), respectively, than the GM approach.

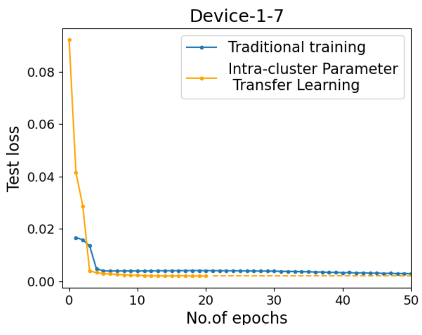
Next, we further explore how the ICPTL approach aims to achieve an accuracy similar to that of the MPD approach with fewer training cycles. In order to achieve this target, ICPTL should be able to converge to a lower test loss (similar to that of MPD) faster. Towards this, for a few selected devices, we compare the test loss for each epoch in the ICPTL approach against that of the MPD approach (for a duration of 50 epochs). For instance, Fig.4.9a depicts the scenario where the model for ”Device-1-2” was traditionally trained using the MPD approach as well as trained from the model for ”Device-1-5” using the ICPTL approach. In some scenarios (e.g., Fig. 4.9b, Fig. 4.9d), the test set loss obtained with parameter transfer learned model (at its 0th epoch, i.e., without performing any retraining) is already low. Even in some cases where the test set loss



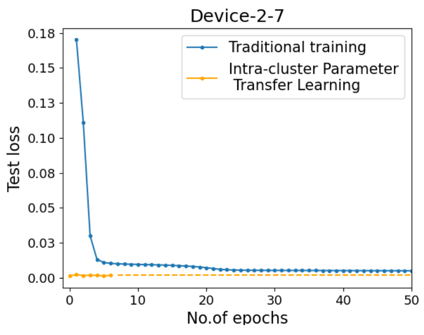
(a) 1–5 ⇒ 1–2 (cluster 1)



(b) 3–9 ⇒ 3–1 (cluster 2)



(c) 3–9 ⇒ 1–7 (cluster 2)



(d) 2–5 ⇒ 2–7 (cluster 0)

Figure 4.9: Comparison between test loss of the ICPTL approach and the MPD approach for selected devices

of the parameter transfer learning approach is high (e.g., Fig. 4.9a, Fig. 4.9c), it converges faster within the first few epochs itself. Regardless of the initial test loss value, the ICPTL approach can converge to a test loss similar to that of the MPD approach faster, thus reaching a satisfactory accuracy within a few epochs, i.e., with no/minimal retraining. Although we have provided results for a few selected scenarios, we could observe the same behavior in other devices as well. In all scenarios, we could observe that the ICPTL approach could reach the maximum accuracy (i.e., a low test loss value) before 20 epochs. Therefore, we set the maximum number of epochs to perform parameter transfer learning to be 20.

Furthermore, we deployed an algorithmic stopping approach that stops training when validation loss does not change significantly for five epochs. As a result, in 4 out of 9 scenarios, parameter transfer learning stops before 20 epochs. Fig. 4.10 presents

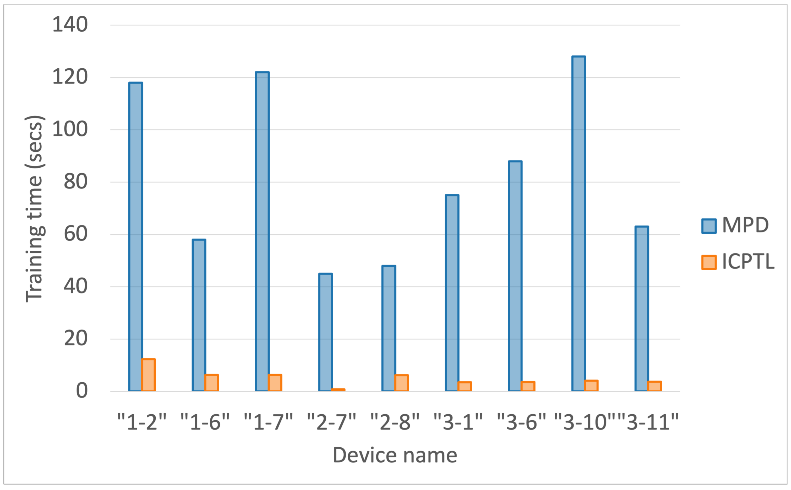


Figure 4.10: Training time comparison between ICPTL and MPD approaches against SMD

a comparison between the training time results of the MPD approach and the ICPTL approach. In terms of training time efficiency, all nine parameter transfer learning scenarios were completed within a maximum of 12.33 seconds, whereas the corresponding traditional model training took a minimum of 45 seconds. While training the models for the nine devices using the MPD approach took a total of 12 minutes and 25 seconds, all nine parameter transfer learning scenarios completed training, taking only a total of 47 seconds, which is only 6% of the training time required by the MPD approach.

Based on our findings, the ICPTL approach achieves a comparable level of accuracy to the MPD approach but requires fewer training cycles. The CM approach demonstrates higher efficiency than the ICPTL approach in terms of total training time and model management overhead while still maintaining a higher accuracy than the GM approach. These results are further validated in the next section by conducting a small-scale experiment using data collected from the emulated edge computing environment.

4.4.4 Evaluating clustering-based training approaches using the emulated dataset

In this section, we further evaluate the proposed clustering-based training approaches using the dataset collected from the emulated edge computing environment. First, we clustered the 10 edge devices based on the collected system and application metrics using Algorithms 2 and 3. We identified the optimal number of clusters, K to be three through hyperparameter tuning.

Table 4.5: Evaluation results of training approaches against the collected dataset

	MPD	**ICPTL**	**CM**	**GM**
AUC (mean)	0.88	0.88	0.86	0.73
F1-score (mean)	0.95	0.96	0.95	0.9
Total training time	13.98secs	4.88secs	3.8secs	1.14secs

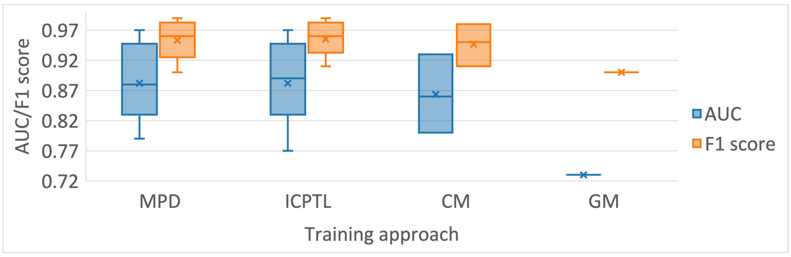


Figure 4.11: Distribution of AUC and F1 score of training approaches against the collected dataset

Table 4.5 contains the accuracy (in terms of average AUC and F1-scores) and training time results of the two proposed clustering-based model training approaches and the two baseline approaches. Figure 4.11 shows the distribution of AUC and F1-scores across these training approaches. During this evaluation, it can be observed that the ICPTL approach had reached the accuracy of the MPD approach (which is also the highest accuracy: average AUC = 0.88 and average F1-score = 0.96) while consuming only 34% of the total training time required by the MPD approach (4.88secs). The CM approach has further increased training efficiency by consuming 22% less training time than the ICPTL approach (4.88secs) while reducing the total number of models from 10 to 3. Furthermore, the CM approach is 13% and 5% more accurate in AUC (0.86) and F1-score (0.95), respectively, than the GM approach.

Thus, based on our experiment, we can assert that the ICPTL approach achieves the same accuracy as the MPD approach in a shorter training time. Furthermore, the CM approach is capable of reaching higher efficiency than the ICPTL approach while maintaining a greater accuracy than the GM approach.

4.4.5 Discussion

In the evaluations conducted on the Server Machine Dataset (SMD) (refer to section 4.4.3) and the emulated dataset (refer to section 4.4.4), we observed that our proposed clustering-based training approaches successfully strike a balance between accuracy and efficiency. In both experiments, the ICPTL approach achieved a level of accuracy comparable to the MPD approach but required a shorter training time. Meanwhile, the CM approach demonstrated higher efficiency, both in terms of total training time and the number of models to manage, than the ICPTL approach while still maintaining a higher accuracy than the GM approach.

Table 4.6: Comparison of efficiency metrics across all approaches in terms of the number of devices (N), the number of clusters (K), and the number of epochs (l,L) ($1 \leq K \leq N$, $l << L$)

Approach	**Model management overhead**	**Training time (in terms of the no.of epochs)**
GM	1	L
CM	K	$K * L$
ICPTL	N	$K * L + (N - K) * l$
MPD	N	$N * L$

Since the dataset size is constant for all approaches during evaluation, we can explain the training time results using parameters such as the number of devices, the number of clusters, and the number of epochs. In addition to training time, as shown in Table 4.6, we can also compare the number of models that need to be managed across all approaches. If we denote the number of devices as N, the MPD approach needs to train N models, while the GM approach needs to train only one model. If we consider the number of epochs to be L, the MPD approach has to train for $N * L$ number of epochs, whereas the GM approach has to train for only L number of epochs. This explains why the MPD approach has the longest training time, while the GM approach has the shortest training time observed in our evaluations.

Similarly, if the number of clusters is K ($K \leq N$), the CM approach requires training a total of K models. Consequently, to train K models, the CM approach requires $K * L$ number of epochs. On the other hand, the ICPTL approach necessitates training of N models. This approach requires training a source model per each cluster from scratch and fine-tuning the remaining models using a smaller number of epochs l (where $l << L$). Hence, ICPTL approach requires $K * L + (N - K) * l$ number of epochs to train the source and target models. Thus, the training time required for the two proposed approaches is greater than that of the GM approach but less than that of the MPD approach. Additionally, among these two proposed approaches, the CM approach requires less training time than the ICPTL approach.

The number of models to manage is an important factor in ML deployments, as these models require periodic retraining and fine-tuning. As mentioned earlier, the GM approach trains only one model, resulting in the lowest model management overhead. Both MPD and ICPTL approaches require training and managing N models, incurring the highest model management overhead. Among the two proposed approaches, the CM approach offers better model management efficiency since it is easier to manage K models (one model per cluster) compared to N individual models (one model per device) in the ICPTL approach.

Although the GM approach requires the least amount of training time, and has the lowest model management overhead, it also results in the least accuracy. In contrast, the MPD approach offers the highest accuracy at the cost of the longest training time. Thus, the proposed clustering-based model training approaches strike a balance between accuracy and efficiency of model training.

4.5 Dynamism Handling of Clustering-based Training Approaches

We proposed the clustering-based training approaches and evaluated them considering the scenario of training models for initial deployment in edge computing environments. However, in dynamic edge computing environments, new edge devices could be added after the first point of deployment, and existing devices could be removed from the infrastructure. Furthermore, the normal data distribution of microservices deployed in edge devices can change due to data drift or osmotic movement of microservices [47]. This dynamism of edge computing environments poses the requirement to retrain performance anomaly detection ML models. Through this section, we explain how our proposed clustering-based training approaches are capable of handling this requirement while maintaining their initial claims regarding accuracy and efficiency.

When a new edge device is added to the infrastructure, it will be assigned a cluster. In the ICPTL approach, the model corresponding to that device will retrain from its nearest neighbor. In the CM approach, the existing model of the cluster can be used unless cluster composition is changed.

When an existing edge device is removed from the infrastructure, ICPTL approach does not require any actions. Whereas, in the CM approach, the model can be retrained only if the cluster composition is significantly changed.

Data drift or osmotic movement can sometimes change the cluster to which an edge device is assigned. In that case, in the ICPTL approach, the model corresponding to the device can retrain from its new nearest neighbor's model. In contrast, the CM approach is required to check for changes in cluster composition of both source and destination clusters and retrain if cluster composition has significantly changed. Inclusion of the above-mentioned dynamism handling capabilities in edge environments ensures a self-adaptive edge/microservices environment for detecting performance anomalies.

4.6 Threats to Validity

Internal validity specifically refers to whether an experimental treatment or condition provides sufficient evidence to support the established claims. In our evaluation, we used the Server Machine Dataset (SMD) along with our own emulated performance anomaly dataset to conduct experiments. As explained in section 4.4.1, we selected the publicly available and widely-used SMD dataset for a comprehensive and large-scale evaluation of our proposed clustering-based training approaches. This choice was primarily due to the heterogeneous normal data distributions exhibited by the SMD devices, which align with the expectations of a real edge environment. Additionally, many other edge-related studies have utilized the SMD dataset for evaluation [16, 17], making it the most relevant publicly available dataset that includes real anomalies.

However, during the SMD-based evaluation, we made an assumption that each SMD machine serves as an edge device with a single microservice deployed on it and that the data reported for each machine (edge device) is associated with its respective microservice. To overcome the effect of this assumption on our claim and to further validate our results, we created our own emulated edge dataset. In generating this dataset, we deployed multiple microservices on each edge device to better replicate the microservice deployments found in real edge environments. However, since this dataset is synthetically generated, we believe that presenting results from both of these complementary datasets demonstrates the effectiveness of our proposed approaches.

External validity primarily concerns the generalizability of treatment or condition outcomes. A key question that arises is whether these approaches can be applied to cloud servers and cloud applications. The answer is that the proposed approaches could be utilized for cloud servers and cloud applications, provided that the edge properties are met. A prime example of this is SMD, where cloud servers exhibit characteristics similar to edge environments. However, these approaches may be less effective in cloud environments, as achieving a trade-off between accuracy and efficiency is often more crucial in edge settings than in cloud applications. Nevertheless, the proposed approaches are expected to perform effectively in edge-cloud integrated computing environments.

4.7 Summary

This chapter examined the challenge of efficiently training anomaly detection models in heterogeneous and resource-constrained edge environments. To balance accuracy and training cost, we proposed two clustering-based strategies: ICPTL, which leverages parameter transfer learning along cluster topologies to reduce training cycles, and CM, which constructs shared models at the cluster level. Comprehensive evaluations on public and emulated datasets demonstrated that ICPTL achieves near-optimal accuracy with significantly reduced training time, while CM further enhances scalability by reducing model proliferation. These findings show that edge-aware training strategies can support timely and accurate anomaly detection, thereby enabling reliable downstream diagnostic processes.

Chapter 5

A Decentralized Root Cause Localization Approach for Edge Computing Environments

Accurate anomaly detection must be complemented by timely root cause localization (RCL) to enable effective system diagnosis in edge environments. However, existing RCL approaches are predominantly centralized and incur substantial latency and communication overhead when applied to distributed edge infrastructures. This chapter proposes a decentralized RCL framework that performs localization directly at the edge device level using Personalized PageRank (PPR). By organizing microservices into communication- and colocation-aware clusters, the proposed method confines most anomaly propagation within local regions and enables efficient intra-cluster diagnosis. For cross-cluster scenarios, a lightweight P2P approximation mechanism is introduced to support coordinated reasoning with minimal overhead. In addition, a novel anomaly scoring mechanism is developed to enhance the accuracy of localization in heterogeneous edge environments. Experimental results on the publicly available edge dataset, MicroCERCL, show that the proposed approach achieves high localization accuracy while significantly reducing diagnosis latency compared to a centralized PPR baseline. In addition, scalability experiments conducted on large-scale datasets generated using the iAnomaly framework show that the proposed decentralized approach consistently outperforms the centralized baseline in terms of localization efficiency under increasing system scale.

This chapter is derived from:

- **Duneesha Fernando**, Maria A. Rodriguez, and Rajkumar Buyya, "A Decentralized Root Cause Localization Approach for Edge Computing Environments", *IEEE Transactions on Services Computing (TSC)*, pp. 1–14, February 2026, doi: 10.1109/TSC.2026.3668228.

5.1 Introduction

Microservices in IoT applications deployed across edge computing environments are typically placed based on both QoS requirements and communication frequency. While QoS-aware placement ensures that resource and latency constraints are satisfied in heterogeneous edge infrastructures, communication-aware placement further aims to co-locate microservices that frequently interact, reducing end-to-end latency and network overhead [52]. Together, these deployment strategies create tightly coupled execution regions in which faults and performance anomalies tend to propagate locally.

Existing approaches for RCL in microservices primarily focus on cloud environments [23–25]. At the time of writing, there is only one study on RCL aimed at edge computing environments, known as MicroCERCL [52]. Both cloud-based RCL approaches and MicroCERCL rely on telemetry data collected from decentralized monitoring agents and transmitted to a central location for analysis. While effective in cloud settings, applying such a centralized approach at the edge increases data transfer times, thereby degrading localization speed. This highlights an opportunity to develop solutions that execute RCL closer to edge devices. To address this gap, our research proposes a decentralized RCL approach that performs the analysis directly at the edge device level.

Graph-based methods represent the current state of the art in RCL [23–25]. Within this space, unsupervised heuristic approaches that exploit graph centralization play a particularly important role. In an RCL setting, centralization measures help identify the most "influential" or "central" node in the anomaly propagation graph, under the assumption that the true root cause will often lie at or near the center of anomalous interactions. These approaches are lightweight and offer high interpretability, making them well-suited for resource-constrained edge environments when compared to more computationally intensive supervised DL methods, such as GNNs [52]. Among the unsupervised centralization techniques, the PPR algorithm has proven to be a particularly effective method for RCL [15, 101]. Building upon this foundation, we employ PPR as the core algorithm in our work.

Edge environments, however, are characterized by a large number of highly distributed devices, creating a complex problem space that graph-based approaches must navigate. This complexity can significantly increase localization times [14, 26, 27]. To address this challenge, our proposed PPR-based decentralized RCL approach groups microservices into clusters based on their communication and colocation dependencies. This means that when an anomaly occurs, it typically propagates within the identified cluster boundaries. In such cases, the source microservice can be located by executing PPR only within that cluster. This reduces the search space that the PPR algorithm needs to explore, leading to shorter localization times compared to the traditional centralized approaches.

In rare instances where anomalies may propagate outside of the cluster, we propose an inter-cluster P2P approximation process. In our inter-cluster P2P approximation process, clusters exchange the average of their anomaly scores, which serves as a compact representation of the anomaly state of each cluster. This exchanged value is incorporated into the receiving cluster's anomaly score list as an approximate indicator of the other cluster's influence. Each cluster then executes its PPR-based RCL algorithm independently. By doing so, we limit communication overhead while still retaining the ability to capture inter-cluster anomaly propagation across iterations.

Existing PPR-based RCL approaches designed for cloud environments typically compute anomaly scores by quantifying their correlation with anomalous response time metrics [101, 102]. While this works well in cloud settings, such correlation-based scoring is less effective in IoT and edge environments, where anomalies can originate from heterogeneous layers of the infrastructure. As a secondary contribution, we extend the conventional PPR framework by introducing a novel anomaly scoring mechanism tailored to the characteristics of edge environments. To the best of our knowledge, this is the first work to propose an RCL approach tailored to the characteristics and limitations of edge environments while specifically trying to reduce the impact of scalability on localization time through decentralization.

We evaluated our proposed decentralized RCL approach using a publicly available dataset released alongside the MicroCERCL paper [52]. Evaluations performed on the dataset using PPR with our proposed anomaly scoring mechanism demonstrate that the decentralized RCL approach achieves the same level of accuracy as its centralized counterpart, while significantly reducing localization time in scenarios of single-cluster anomaly propagation, as well as in rare instances where anomalies propagate across multiple clusters. In addition, scalability experiments conducted on large-scale datasets generated using the iAnomaly framework introduced through chapter 3 show that the proposed decentralized approach consistently outperforms the centralized baseline in terms of localization efficiency under increasing system scale.

This chapter makes the following key contributions:

- We propose a fully decentralized RCL framework for edge computing environments based on PPR, enabling efficient localization without centralized data aggregation.
- We introduce a novel anomaly scoring mechanism tailored to the characteristics of edge infrastructures, capturing anomaly triggers across microservice, device, and network layers.
- We develop a communication- and colocation-aware clustering strategy that limits anomaly propagation within clusters, reducing localization latency.
- We design an inter-cluster P2P approximation and aggregation mechanism to support robust localization under multi-cluster anomaly propagation.
- We conduct extensive evaluations on the MicroCERCL benchmark and large-scale iAnomaly datasets, demonstrating improved accuracy, scalability, and efficiency over centralized baselines.

The rest of this chapter is organized as follows: Section 5.2 reviews current research on RCL in microservice-based systems, discusses their deployment models, presents an overview of the Personalized PageRank algorithm, and examines the efficiency aspects of existing RCL techniques. Section 5.3 introduces our proposed decentralized RCL methodology, including the novel anomaly scoring mechanism, the communication- and colocation-based clustering approach, and the decentralized execution of the PPR algorithm. Section 5.4 reports experimental results on both the MicroCERCL benchmark and large-scale datasets generated using iAnomaly, evaluating localization accuracy, efficiency, and scalability with respect to system size and cluster count. Section 5.5 concludes the chapter.

5.2 Background and Related Work

In this section, we provide an overview of existing RCL techniques, examine their deployment patterns and the challenges of adapting cloud-based approaches to edge environments, present the fundamentals of the PPR algorithm, and discuss the efficiency aspects of existing RCL techniques.

5.2.1 Root cause localization techniques

RCL in microservices environments leverages various data sources to identify performance issues, categorized into metric-based, trace-based, log-based, and multi-source approaches [23, 24]. Metric-based RCL focuses on KPIs like CPU utilization and Queries Per Second (QPS), providing direct insights into potentially faulty modules. It stands out for its efficiency and adaptability compared to log-based approaches, which are often voluminous, unstructured, heterogeneous in format, and complex to analyze in real-time [25, 105].

Trace-based RCL , while useful for understanding request flows, is often too coarse-grained to effectively diagnose deeper system issues, such as resource bottlenecks or internal microservice misbehaviors [108]. Additionally, these methods encounter scalability challenges, as they require expensive and time-consuming data collection and processing. Furthermore, they require instrumentation of source code, whereas metrics can be collected without intrusive modifications.

Emerging multi-source RCL techniques aim for a comprehensive view by integrating metrics, logs, and traces, but add complexity due to differences in data formats, granularity, and temporal alignment [25, 112]. In some cases, one modality may even dominate or obscure the others. In contrast, metric-based approaches offer a structured and simpler framework, making them practical and effective for RCL investigations in dynamic edge-cloud environments.

RCL techniques that rely on metrics as the primary source of data can be broadly divided into two categories: correlation-driven statistical approaches and graph-based approaches [24, 25]. Correlation-driven statistical approaches are typically triggered when an anomaly detection module identifies unusual response times in the frontend microservice. These methods then compute correlations between the anomalous response times and metrics from other microservices, ultimately reporting the metric with the highest correlation and its corresponding microservice [53, 113]. While these methods are suitable for deployment in cloud environments, where web applications are primarily hosted, they fall short when applied at the edge, where anomalies can arise from any metric across various levels of the infrastructure. Moreover, these methods generally struggle with accuracy due to their reliance on simple pairwise correlations, which may overlook multi-service dependencies and indirect causal paths.

In contrast, graph-based RCL methods, which represent the current state of the art, explicitly model service dependencies and anomaly propagation using graph structures constructed after anomaly detection [15, 52, 101, 102]. Two main types of graphs are commonly used: causal graphs, which represent direct cause–effect relationships between metrics, and topological graphs, which capture service interactions based on deployment and runtime traces [23, 24]. This work focuses on topological graphs, as they can be easily constructed using deployment information and traces, making them practical for modeling microservice dependencies, unlike causal graphs that are difficult to construct accurately in dynamic edge-cloud environments.

Once constructed, these graphs can be analyzed using either unsupervised heuristic methods (e.g., centrality-based approaches) or supervised DL methods (e.g., GNNs) [23, 25]. Centrality-based methods exploit graph properties to identify the most "influential" or "central" node within the anomaly propagation graph, under the assumption that the true root cause often lies near the center of anomalous interactions. These methods are lightweight, interpretable, and require no labeled training data or pre-trained models, making them highly suitable for resource-constrained and heterogeneous edge environments [24]. In contrast, supervised methods such as GNNs are often computationally expensive, demand large labeled datasets, and require retraining to adapt to new system configurations, which limits their practicality at the edge. Among other unsupervised centralization techniques such as degree centrality (identifies nodes with the largest number of anomalous connections), betweenness centrality (highlights nodes acting as bridges in anomalous paths), and eigenvector-based methods (emphasize nodes connected to other highly anomalous nodes), PPR has emerged as a particu-

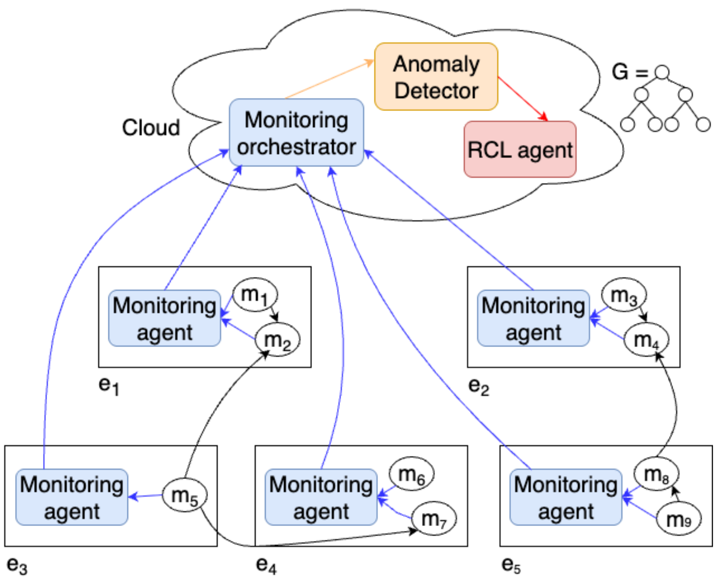


Figure 5.1: Centralized root cause localization in edge environments

larly effective method for RCL [15, 101]. Unlike simple degree or betweenness measures, PPR captures both local and global structural dependencies, enabling it to rank potential root causes more robustly even in large and heterogeneous graphs. Therefore, we adopt the PPR algorithm as the foundation for our RCL solution.

In the next section, we discuss how existing RCL solutions are typically deployed and highlight the challenges that arise when these cloud-based approaches are applied in edge computing environments.

5.2.2 Centralized deployment of existing root cause localization solutions

As shown in Figure 5.1, the edge infrastructure consists of a set of edge devices

$$\mathcal{E} = \{e_1, e_2, \ldots, e_{|\mathcal{E}|}\}.$$

Let the set of microservices be:

$$\mathcal{M} = \{m_1, m_2, \ldots, m_{|\mathcal{M}|}\}.$$

Each microservice $m \in \mathcal{M}$ is deployed on exactly one edge device. The deployment mapping can be defined as

$$\delta : \mathcal{M} \to \mathcal{E},$$

where $\delta(m)$ gives the device on which microservice m is deployed.

Each edge device $e \in \mathcal{E}$ has a monitoring agent that gathers performance and resource consumption metrics from the microservices deployed on that device. Both cloud-based RCL studies and MicroCERCL send the collected telemetry data to a central location, such as the cloud, for analysis. While this approach works well in the cloud, where data transfer times are relatively low, applying it at the edge increases data transfer times, which negatively impacts the speed of localization. Furthermore, transferring data to a central location does not align well with the network instability characteristic of cloud-edge environments [52]. Therefore, it is essential to develop solutions that enable RCL to be performed closer to the edge devices in edge computing environments.

In situations where graph-based RCL approaches are in place, the centralized RCL module maintains a topology graph

$$\mathcal{G} = (\mathcal{V}, \mathcal{L}),$$

which captures the structure and interactions of the monitored system.

The set of vertices V consists of both microservices and the edge devices on which they are deployed:

$$\mathcal{V} = \mathcal{M} \cup \mathcal{E}$$

The set of edges L represents relationships between these vertices and is composed of two distinct types:

$$\mathcal{L} = \mathcal{L}_{\text{comm}} \cup \mathcal{L}_{\text{dep}}$$

1. Communication edges (L_{comm}) capture interactions between microservices, derived from trace data:

   $$\mathcal{L}_{\text{comm}} \subseteq \mathcal{M} \times \mathcal{M},$$

   where $(m_i, m_j) \in \mathcal{L}_{\text{comm}}$ if microservice m_i communicates with m_j.

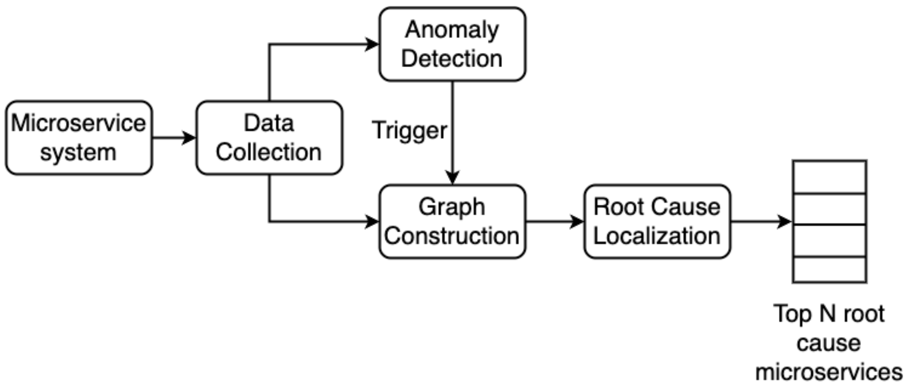


Figure 5.2: Overview of the root cause localization process

2. Deployment edges (L_{dep}) link each microservice to the edge device on which it is hosted, obtained from the deployment configuration:

$$\mathcal{L}_{\text{dep}} = \{(m, \delta(m)) \mid m \in \mathcal{M}\}.$$

Together, these vertices and edges form a topology that reflects both the logical communication between microservices and their physical placement across edge devices, enabling the RCL process to reason over the system's structure.

In cloud environments, where primarily web applications are deployed, continuous anomaly detection is typically performed only on the response times of user-facing microservices [15, 101–103]. When an anomaly is detected in any of those metrics, it triggers the RCL process. In contrast, many edge applications do not interact directly with end users and often follow architectures such as publish–subscribe or streaming. Therefore, instead of relying solely on user-facing response times, anomaly detection must be performed at multiple levels across the infrastructure—including microservice, device, and network layers—to ensure effective RCL in edge environments. In this context, anomalies detected at any level should trigger the RCL process. Once an anomaly is identified, the edges and vertices of the topology graph ($\mathcal{G}$) are populated with the anomalous metrics from the detection window. This graph serves as input for the PPR algorithm. Figure 5.2 demonstrates an overview of the RCL process from data collection to anomalous services ranking. The next subsection explains the PPR algorithm in detail.

5.2.3 Personalized PageRank algorithm for root cause localization

Although the topology graph ($\mathcal{G}$) populated with anomalous metrics serves as the input for the RCL process, the PPR algorithm specifically operates on two key inputs: the personalization vector ($\mathcal{S}$) and the transition probability matrix ($\mathcal{P}$) [155].

The personalization vector $S \in \mathbb{R}^{|V|}$ represents the prior probability distribution over the nodes, encoding our belief regarding their likelihood of being the root cause. In the proposed framework, S is derived from anomaly scores assigned to nodes and edges in the topology graph. Formally, for a node $v_i \in V$:

$$S_i = \frac{a(v_i)}{\sum_{v_j \in V} a(v_j)}$$

where $a(v_i)$ denotes the anomaly score of node v_i. This ensures that S is a valid probability distribution:

$$\sum_{i=1}^{|V|} S_i = 1.$$

The transition probability matrix $P \in \mathbb{R}^{|V| \times |V|}$ captures the normalized probabilities of transitioning between nodes based on edge weights. For an edge $l_{ij} \in L$ connecting nodes v_i and v_j, its weight is determined by its anomaly score $a(l_{ij})$. The transition probability is defined as:

$$P_{ij} = \frac{a(l_{ij})}{\sum_{v_k \in \mathcal{N}(v_i)} a(l_{ik})},$$

where $\mathcal{N}(v_i)$ denotes the set of neighbors of v_i. Consequently, P is a row-stochastic matrix, ensuring:

$$\sum_{j=1}^{|V|} P_{ij} = 1 \quad \forall i.$$

PPR-based RCL approaches designed for cloud environments typically compute the anomaly scores for the nodes and edges of the constructed topology graph—required for calculating P and S—by quantifying their correlation with anomalous response time metrics [101, 102]. Specifically, for each anomalous node or edge, the correlation between its observed metric values and the anomalous response time metric is measured, thereby assigning higher scores to components exhibiting stronger associations with the observed performance degradation. However, IoT applications would benefit from a novel anomaly scoring mechanism tailored to address the triggers that arise from different levels of edge infrastructure.

With P and S defined, the PPR algorithm iteratively refines a ranking vector $r \in \mathbb{R}^{|V|}$, representing the steady-state probabilities of each node being the root cause. The update rule is given by:

$$r^{(k+1)} = \alpha P r^{(k)} + (1 - \alpha) S,$$

where:

- $\alpha \in (0, 1)$ is the damping factor, controlling the trade-off between following graph transitions ($Pr^{(k)}$) and teleporting to the personalization distribution (S);
- $r^{(k)}$ denotes the ranking vector at iteration k.

The algorithm converges when the difference between successive iterations falls below a predefined threshold:

$$\|r^{(k+1)} - r^{(k)}\| < \varepsilon,$$

where ε denotes the desired precision. Convergence is achieved using *power iteration*, with the number of iterations influenced by both α and ε. Lower values of α typically result in faster convergence due to increased reliance on the personalization vector.

Given that our approach seeks to improve localization efficiency by reducing localization time, we next review the efficiency aspects of existing RCL techniques.

5.2.4 Efficient root cause localization techniques

Training time, localization time, and resource overhead are the most important efficiency metrics considered in studies on cloud RCL [25]. These metrics hold equal or even greater significance at the edge. By utilizing the PPR algorithm, we have eliminated the need for training and reduced resource overhead due to its lightweight nature. However, our proposed decentralized RCL approach primarily targets the reduction of localization time.

Localization time can be discussed in relation to the time complexity of the PPR algorithm. The time complexity of the PPR algorithm for a given topology graph $\mathcal{G} = (\mathcal{V}, \mathcal{L})$ depends on the size of the graph, which can be expressed in terms of its number of edges [156]. At each iteration, the algorithm performs a sparse matrix–vector multiplication, whose computational cost is proportional to the number of edges $|\mathcal{L}|$. Therefore, the per-iteration complexity is $\mathcal{O}(|\mathcal{L}|)$. If k denotes the number of iterations required to reach convergence, the total time complexity becomes $\mathcal{O}(k \cdot |\mathcal{L}|)$. Thus, the overall convergence time of the PPR algorithm scales linearly with both the number of edges and the number of iterations needed to achieve convergence.

In contrast to conventional PageRank algorithm, which distributes teleportation probabilities equally across all nodes, the PPR approach biases the algorithm towards nodes and edges with higher anomaly scores [155]. This personalization effectively prioritizes microservices and interactions that are more likely to be the root cause at the outset, thereby reducing the number of iterations required for convergence and improving localization efficiency. Edge environments, however, consist of a large number of devices and are highly distributed, creating a complex problem space for graph-based approaches to navigate [14, 26, 27]. In such environments, where the underlying dependency graph can become extremely large, reducing the effective search space—and hence the number of edges considered during computation—is crucial for minimizing localization time. This is where our proposed decentralized RCL approach contributes, by narrowing the search space through cluster-based graph partitioning, which indirectly reduces the number of edges involved in PPR computation and thus accelerates convergence.

A comparison of cloud and edge RCL techniques in terms of their focus on efficiency (and accuracy as well) is shown in Table 5.1. The majority of studies on cloud RCL have primarily focused on improvements in accuracy, evaluating this single aspect [24, 27, 29]. Some research has also addressed efficiency, particularly the localization times of their proposed methods. For instance, AAMR [102], MicroEGRCL [106], and Grace [110] claim to provide faster inference times. However, none of these studies have specifically designed their approaches with efficiency as a priority. In addition, the only study focused on edge RCL , MicroCERCL, did not incorporate efficiency considerations

Work	Cloud-only/ Cloud-Edge	Centralized/ Decentralized	Accuracy Focus	Efficiency Focus	Remarks
[27], [24], [29]	Cloud-only	Centralized	✓	×	Focused solely on accuracy
[102], [106], [110]	Cloud-only	Centralized	✓	△	Claim faster inference; efficiency not a design priority
MicroHECL [53]	Cloud-only	Centralized	△	✓	Efficient traversal & pruning; may compromise accuracy
PDiagnose [109]	Cloud-only	Centralized	△	✓	Vote-based localization; improves efficiency; may compromise accuracy
MicroCERCL [52]	Cloud–Edge	Centralized	✓	×	Accuracy focus; long inference time reported
Our work	Cloud–Edge	Decentralized	✓	✓	Graph-based; balances accuracy and efficiency
✓= Primary focus / addressed; △= Partially addressed / may compromise; ×= Not addressed / ignored					

Table 5.1: Summary of cloud and edge RCL techniques: accuracy vs efficiency focus

into its approach. Consequently, during efficiency evaluations, it was found that its inference time was longer than that of unsupervised heuristic approaches, largely due to the complexity of the network [52].

MicroHECL [53] and PDiagnose [109] are cloud RCL techniques that specifically aim to improve efficiency, particularly by providing faster localization speeds. Both approaches eliminate the need for training, similar to our chosen method. They are centralized approaches; MicroHECL achieves efficiency by efficiently traversing the service dependency graph and using pruning techniques to eliminate irrelevant service calls during anomaly propagation chain analysis, which further enhances efficiency. On the other hand, PDiagnose tries to reach efficiency by removing the computationally heavy dependency graph-building phase and utilizing a vote-based localization process on an anomaly queue. However, both approaches adopt relatively simplistic strategies that may compromise localization accuracy. Given the need to strike an appropriate balance between effectiveness and efficiency, our proposed decentralized method leverages a graph-based approach to enhance accuracy while reducing localization time. Additionally, cloud efficiency techniques like pruning, as used in MicroHECL, complement our proposed approach.

Building on the insights from existing RCL techniques and their efficiency consid-

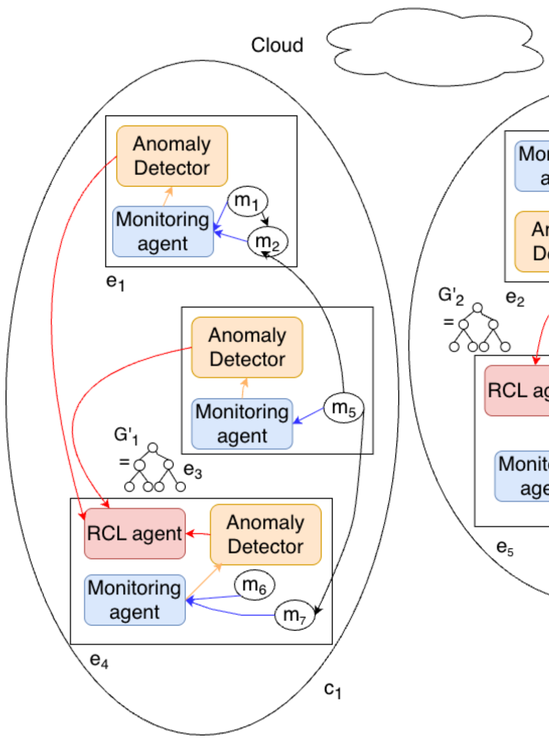


Figure 5.3: Decentralized root cause localization in edge environments

erations, the next section details our proposed decentralized RCL methodology, which aims to reduce localization time while maintaining high accuracy.

5.3 Methodology

Our proposed decentralized RCL approach aims to minimize the need for the PPR algorithm to traverse the entire graph by clustering edge devices that frequently communicate with each other. This means that the PPR algorithm only needs to explore the cluster where the anomaly has propagated. As a result, the search space is reduced, leading to shorter localization times. Additionally, since our approach conducts localization as close to the edge device level as possible, it further decreases localization time by minimizing data transfer delays.

We define the set of clusters as

$$\mathcal{C} = \{c_1, c_2, \ldots, c_{|\mathcal{C}|}\}.$$

Each edge device $e \in \mathcal{E}$ is assigned to exactly one cluster, and we can represent the mapping of edge devices to clusters as

$$\gamma : \mathcal{E} \rightarrow \mathcal{C},$$

where $\gamma(e)$ indicates the cluster to which edge device e belongs. We will provide further details on the clustering algorithm in subsection 5.3.2.

As illustrated in Figure 5.3, our approach deploys an anomaly detection module at each edge device. This module utilizes a Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) clustering-based anomaly detection model. BIRCH is an unsupervised, lightweight technique that is widely used for anomaly detection in multiple RCL studies [52, 101–103], and it also suits the properties of edge environments. It analyzes the performance and resource consumption metrics collected by the monitoring agent assigned to each edge device. Unlike cloud-based applications that primarily deploy web applications, anomaly detection for IoT applications—which are not necessarily user-facing—should analyze all metrics obtained from the monitoring agent.

In our context, where anomaly detection occurs at multiple levels throughout the infrastructure, we consider any detected anomalies as triggers for RCL. In the proposed approach, the RCL module corresponding to each cluster is placed at the node with the highest computational power within the cluster. Each RCL module maintains a topology subgraph G'_i that corresponds to its cluster c_i. When an anomaly is detected, the edges and vertices of the topology subgraph G'_i are populated with the anomalous metrics gathered from the microservices deployed in that cluster.

Subsequently, we form the personalization vector S_i and the transition probability matrix P_i corresponding to the topology subgraph by using a novel anomaly scoring mechanism, which we introduce in subsection 5.3.1. The novel anomaly scoring mechanism is tailored to address triggers arising from different levels of the edge infrastructure. Following this, we can execute the PPR algorithm within the cluster. Our approach assumes that in the majority of scenarios, an anomaly will propagate within a single cluster, which means we only need to explore that specific cluster. However, in rare cases where anomalies may propagate outside of the cluster, we provide an inter-cluster P2P approximation process that the clusters can follow to localize the root cause microservice. This strategy also aims to reduce localization time by parallelizing the graph traversal across clusters and reducing inter-process communication overhead through one time exchange of approximate anomaly scores. The inter-cluster P2P approximation process will be further explained in subsection 5.3.3.

The upcoming sections are organized as follows: Section 5.3.1 introduces the novel anomaly scoring mechanism tailored for edge environments while section 5.3.2 explains the proposed communication and colocation-based clustering approach, followed by the topology subgraph formation algorithm. Section 5.3.3 explains the decentralized execution of the PPR algorithm for both single cluster and multiple cluster anomaly propagation scenarios.

5.3.1 Novel anomaly scoring mechanism for edge environments

PPR-based RCL approaches designed for cloud environments typically compute the anomaly scores for the nodes and edges of the constructed topology graph—required for calculating P and S—by quantifying their correlation with anomalous response time metrics [101, 102]. Specifically, for each anomalous node or edge, the correlation between its observed metric values and the anomalous response time metric is measured, thereby assigning higher scores to components exhibiting stronger associations with the observed performance degradation. However, in IoT applications, where anomalies may arise from various levels of edge infrastructure, we introduce a novel anomaly scoring mechanism designed specifically to meet these requirements.

Anomaly score for microservices

Each microservice m is assigned an anomaly score that aggregates two main influences:

- Anomalous metric influence

- Incoming anomalous edge influence

Thus, the anomaly score for a microservice m, denoted by $\text{AS}(m)$, is computed as:

$$\text{AS}(m) = \underbrace{\text{AS}_{\text{metric}}(m)}_{\text{Metric-induced}} + \underbrace{\text{AS}_{\text{edge}}(m)}_{\text{Edge-induced}}$$

Anomalous metric influence Let A denote a BIRCH cluster of anomalous time series metrics associated with microservice m. We define the anomalous metric influence score of microservice m, denoted by $\text{AS}_{\text{metric}}(m)$, as:

$$\text{AS}_{\text{metric}}(m) = \sum corr(A) \tag{5.1}$$

where $corr(A)$ is the pairwise correlation among all metric time series in cluster A computed using a direction-invariant correlation metric such as the absolute Pearson correlation metric.

Incoming anomalous edge influence To capture the performance impact on downstream services due to microservice m, we consider all anomalous incoming edges represented as microservice pairs (m_d, m), where $m_d \in D$ is the downstream microservice sending requests to the target microservice m. D is the set of all downstream microservices from m with an anomalous incoming edge to m. For each such edge, we compute the correlation between the edge's anomalous response time (represented by the 90th percentile of latency) and each anomalous metric $x_i \in A$ of the target microservice m and obtain the maximum of these correlation values. The sum of all such maximum correlation values is assigned to the incoming anomalous edge influence score of microservice m, denoted by $\text{AS}_{\text{edge}}(m)$, as shown in equation 5.2.

$$\text{AS}_{\text{edge}}(m) = \sum_{m_d \in D} \max_{x_i \in A} corr(\text{RT}_{m_d \to m}, x_i) \tag{5.2}$$

This formulation enables the anomaly scoring mechanism to account for both internal anomalies and anomalous behavior observed at communication boundaries, capturing the cascading effect of faults within microservice architectures. These microservice-level scores are used to form the personalization vector S.

Anomaly score for edges

The anomaly score for each inter-service edge represented as a microservice pair (m_s, m_t), denoted by $\text{AS}(m_s, m_t)$, is computed using the same formulation as the anomalous metric influence score presented in equation 5.1.

$$\text{AS}(m_s, m_t) = \sum corr(B) \tag{5.3}$$

B denotes a BIRCH cluster of anomalous edge-level metrics (e.g., different percentiles of response time) associated with the communication from microservice m_s to m_t, while $corr(B)$ is the pairwise correlation among all metric time series in cluster B computed using a direction-invariant correlation metric such as the absolute Pearson correlation metric.

These edge-level scores are subsequently used to populate the transition probabilities in the matrix P, guiding the flow of anomaly information through the system topology during the RCL phase.

This novel anomaly scoring mechanism, specifically designed for edge computing environments, is utilized in our proposed decentralized RCL approach, which is explained in the upcoming sections.

5.3.2 Communication and colocation-based clustering

Anomalies can propagate through communication and colocation dependencies in microservice architectures [14, 15]. Our proposed clustering approach aims to group microservices that are both colocated and frequently communicate with one another, so that when an anomaly occurs, it propagates within the identified cluster boundary in most cases.

Algorithm 5 presents our proposed communication and colocation-based clustering approach. It obtains the centralized topology graph $\mathcal{G}' = (\mathcal{V}, \mathcal{L}_{\text{comm}})$ consisting of the set

Algorithm 5: Communication and colocation-based clustering algorithm

```
1: *Input* : Centralized topology graph $G' = (V, L_{comm})$ where $V$ is the set of
   microservices $M$ and edge devices $E$, and $L_{comm}$ is the set of communication edges
   between microservices
2: *Input* : Function $\delta(m)$ which maps each microservice $m$ to its deployment node
3: *Input* : Threshold to merge clusters $\tau$
4: *Output* : Cluster to edge device mapping $cluster\_map$
5: Initialize $cluster\_map$ such that for each deployment node $e_i$, there is a cluster
   $c_{e_i} = \{m \in V \mid \delta(m) = e_i\}$
6: For each pair of clusters $(c_{e_i}, c_{e_j})$, define the inter-cluster communication weight
   $w(c_{e_i}, c_{e_j}) = \sum_{\substack{m_p \in c_{e_i} \\ m_q \in c_{e_j}}} freq(m_p, m_q)$, where $freq(m_p, m_q)$ is the communication
   frequency between microservices $m_p$ and $m_q$
7: **repeat**
8:   Find the cluster pair $(c_p, c_q)$ with maximum weight $w(c_p, c_q)$
9:   **if** $w(c_p, c_q) > \tau$ **then**
10:     Merge $c_p$ and $c_q$ into a new cluster $c_{new} = c_p \cup c_q$
11:     Update $cluster\_map = (cluster\_map \setminus \{c_p, c_q\}) \cup \{c_{new}\}$
12:     Recalculate inter-cluster weights $w(c_{new}, c_i)$ for all $c_i \in cluster\_map \setminus \{c_{new}\}$
13:   **end if**
14: **until** all inter-cluster weights $w(c_i, c_j) \leq \tau$
15: *Return* : $cluster\_map$
```

of microservices M and the set of edge devices E as vertices, and the set of communication edges between microservices L_{comm}, together with function $\delta(m)$ which maps each microservice m to its deployment node and τ which is the threshold to merge clusters as inputs. As an edge device is the smallest unit that groups colocated microservices, the algorithm starts by initializing $cluster\_map$, which maintains the cluster to edge device mapping, such that there is a cluster corresponding to microservices deployed in each edge device. However, since microservices are placed primarily to satisfy QoS requirements, rather than based on communication frequency, there would be communication dependencies between initial clusters. Therefore, our algorithm next attempts to group such devices with high communication frequencies. As detailed in step 6 of Algorithm 5, we calculate the inter-cluster communication weight for each pair of clusters. This

Algorithm 6: Topology subgraph formation for decentralized RCL modules

```
1: *Input*: Final $cluster\_map$ from Algorithm 5
2: *Input*: Centralized topology graph $G' = (V, L_{comm})$
3: *Output*: Set of topology subgraphs $\{G'_i\}$, one for each cluster $c_i$
4: **for** each cluster $c_i$ in $cluster\_map$ **do**
5:     $V_{c_i} \leftarrow \{m \in V \mid \delta(m) \in c_i\}$ {Vertices: microservices in cluster $c_i$}
6:     $L_{c_i} \leftarrow \{(m_p, m_q) \in L_{comm} \mid m_p \in V_{c_i} \wedge m_q \in V_{c_i}\}$ {Intra-cluster edges}
7:     $proxy\_nodes \leftarrow \emptyset$
8:     **for** each $(m_p, m_q) \in L_{comm}$ where $m_p \in V_{c_i}$ and $m_q \notin V_{c_i}$ **do**
9:         Create a new shadow node $s_{m_q}$ representing $m_q$ inside cluster $c_i$
10:        $proxy\_nodes \leftarrow proxy\_nodes \cup \{s_{m_q}\}$
11:        $L_{c_i} \leftarrow L_{c_i} \cup \{(m_p, s_{m_q})\}$ {Redirect outgoing edge to proxy node}
12:    **end for**
13:    $V_{c_i} \leftarrow V_{c_i} \cup proxy\_nodes$
14:    Form topology subgraph $G'_i = (V_{c_i}, L_{c_i})$
15: **end for**
16: *Return*: Set of topology subgraphs $\{G'_i\}$, one for each cluster $c_i$
```

weight represents the communication frequency between the microservices deployed in those clusters. Subsequently, the algorithm performs an iterative merging procedure. At each step, it identifies the pair of clusters with the highest inter-cluster communication weight. If this weight exceeds the threshold τ, the two clusters are merged into a single cluster, and the $cluster\_map$ is updated accordingly. The inter-cluster communication weights involving the newly formed cluster are then recalculated. This process continues until no inter-cluster weight exceeds the threshold τ, ensuring that clusters are only merged when strong communication relationships exist. Finally, the algorithm outputs the updated $cluster\_map$, which contains the final cluster-to-edge device mapping.

Once the final set of clusters is obtained, we construct the topology subgraph for the decentralized RCL module corresponding to each cluster. This process is explained in Algorithm 6. It obtains the $cluster\_map$ which is the output of Algorithm 5 together with the centralized topology graph $\mathcal{G}' = (\mathcal{V}, \mathcal{L}_{\text{comm}})$ consisting of the set of microservices M and the set of edge devices E as vertices, and the set of communication edges between microservices L_{comm}, as inputs. For a given cluster c_i, the vertices of its topology subgraph, G'_i, consist of all microservices deployed on the edge devices assigned to c_i. The intra-cluster edges are defined as the communication edges between these microservices in the centralized topology graph, G'. In addition, to preserve the connectivity information with external microservices while ensuring that the sub-topology remains self-contained, each communication edge originating from a microservice in c_i to a microservice outside c_i is redirected to a shadow node placed within G'_i. This shadow node symbolically represents the remote endpoint and serves as the destination for the redirected outgoing edge. The resulting topology subgraph therefore captures both the internal structure of the cluster and its interaction points with the rest of the system, enabling the decentralized RCL module to operate independently while still considering external dependencies.

To illustrate how an inter-cluster communication edge is represented using a shadow node in the source cluster, we refer to Figure 5.4. In this example, clusters c_1 and c_2 are connected by an inter-cluster communication edge. The topology subgraph G'_1 for cluster c_1 contains as vertices all microservices deployed within c_1, along with the communication edges between them. Similarly, the topology subgraph G'_2 for cluster c_2 contains all microservices deployed within c_2 and their internal communication edges. In addition, G'_2 includes a shadow node representing the destination microservice in c_1 that is the endpoint of the outgoing inter-cluster link from c_2. The outgoing edge from the originating microservice in c_2 is redirected to this shadow node, allowing G'_2 to capture the external communication dependency without directly incorporating microservices from outside its own cluster.

The above-mentioned inter-cluster communications are expected to be infrequent, and the occurrence of anomalies along such inter-cluster edges is even rarer. Each cluster is deliberately constructed to capture the majority of communication and colocation relationships, thereby ensuring that most anomalies propagate entirely within a single cluster. Consequently, RCL can typically be performed within the cluster in which the anomaly is propagated. Upon anomaly detection, the corresponding topology subgraph G'_i is populated with the anomalous metrics associated with its microservices. Subsequently, the personalisation vector S_i and the transition probability matrix P_i are constructed using the novel anomaly scoring mechanism introduced in section 5.3.1.

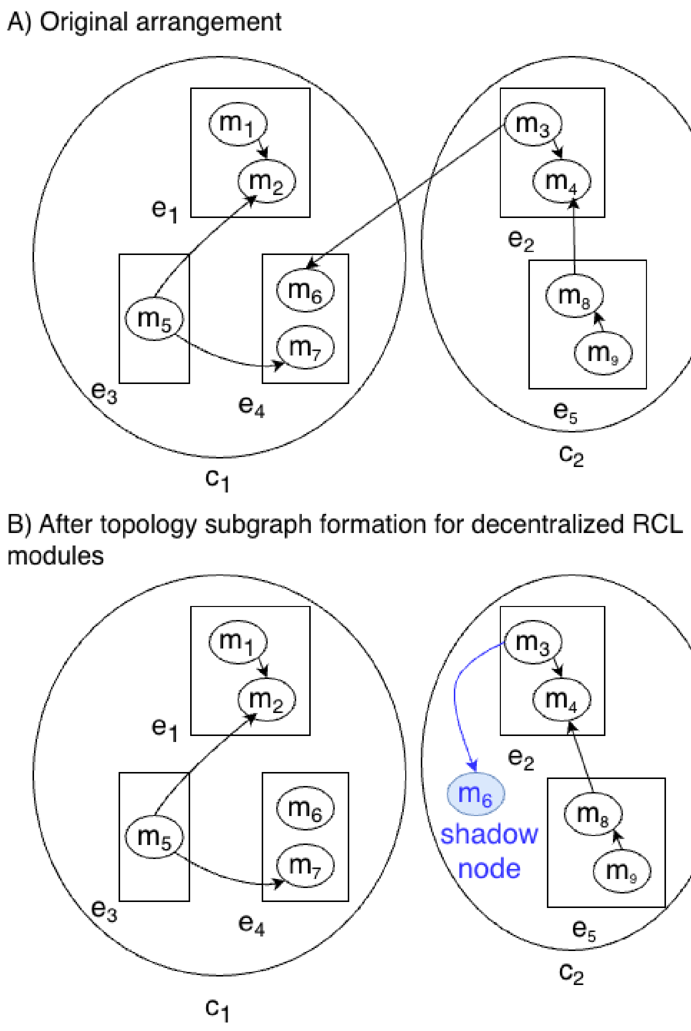


Figure 5.4: Inter-cluster communication edge representation using a shadow node

5.3.3 Decentralized execution of the Personalized PageRank algorithm

There are two possible scenarios when performing PPR, depending on whether the incoming edges to shadow nodes are detected as anomalous. In most cases, due to communication and colocation-based clustering, anomaly propagation remains confined within a single cluster. In such cases, PPR can be executed locally within the cluster according to the formulation in subsection 5.2.3, iterating until convergence and returning the microservice with the highest probability in r. In this scenario, both P_i and S_i are constructed using only the original (non-shadow) nodes in the cluster.

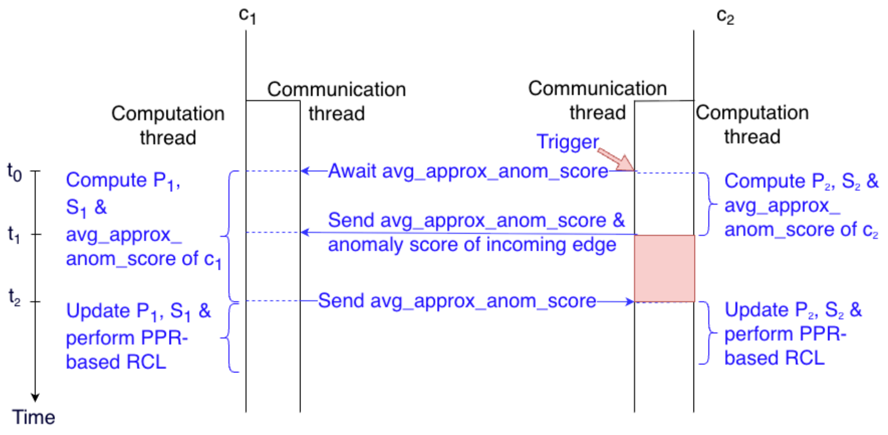

c1
c2
Computation thread
Communication thread
Communication thread
Computation thread
Trigger
t0
t1
t2
Time
Compute P1, S1 & avg_approx_anom_score of c1
Await avg_approx_anom_score
Send avg_approx_anom_score & anomaly score of incoming edge
Send avg_approx_anom_score
Update P1, S1 & perform PPR-based RCL
Compute P2, S2 & avg_approx_anom_score of c2
Update P2, S2 & perform PPR-based RCL

Figure 5.5: Inter-cluster P2P approximation process

Although the clustering strategy is designed to minimize inter-cluster anomaly propagation, rare cases may still occur where anomalies traverse less frequent communication paths spanning multiple clusters. These cases are detected when incoming edges to shadow nodes are identified as anomalous. When this happens, the anomaly propagation path between clusters is considered disconnected, and the inter-cluster P2P approximation process is triggered.

Consider the example in Figure 5.4 together with the corresponding interaction diagram in Figure 5.5. At time t_0, the computation threads of both clusters c_1 and c_2 independently begin computing their respective P_i and S_i. If an anomaly is detected on the incoming edge to the shadow node in c_2, that indicates a disconnection in the anomaly propagation path and triggers the inter-cluster p2p approximation process. At that time (i.e., $t_0 + \delta_1$), the communication thread of cluster c_2 sends a *wait_message* to c_1, indicating its intent to provide an average approximate anomaly score. This approximation is used to reduce communication overhead while still capturing the anomaly influence of c_2 across iterations. While the communication thread is sending the *wait_message*, the RCL module (i.e., the computation thread) in c_2 continues to compute the anomaly scores required to form P_2 and S_2 and uses them to calculate its average approximate anomaly score as:

$$\text{avg\_approx\_anom\_score} = \frac{\sum_{x \in S} x}{|S|} \tag{5.4}$$

After completing the computation (i.e., at time t_1), this value is sent to c_1 along with the anomaly score of the incoming edge to the shadow node.

Upon receiving the *wait_message* from c_2 at time $t_0 + \delta_2$, c_1 continues to compute P_1 and S_1, while awaiting c_2's average approximate anomaly score. It also calculates its own average approximate anomaly score in the same way as equation 5.4 and sends it to c_2 when ready at time t_2. Note that one cluster must wait for the other to complete this computation, since $t_1 \neq t_2$. In the depicted example, $t_2 > t_1$, and thus c_2 waits, as indicated by the red block between t_1 and t_2.

Once both clusters have exchanged their average approximate anomaly scores, they update their P_i and S_i (a straightforward update to the corresponding matrix and vector) by incorporating the other cluster as a node. Both clusters then run their PPR-based RCL algorithms independently, as in the standard scenario. This procedure is a modified version of the JXP algorithm [157], an efficient and decentralized PageRank approximation used in P2P web search networks. Without loss of generality, this communication protocol is extendable to scenarios with more than two clusters.

Finally, during decision making, one cluster c_i (e.g., c_1 in our case) typically identifies a microservice within its boundary as the root cause, while other clusters (e.g., c_2) point to c_i as the origin of the anomaly. In this case, the microservice identified by c_i is selected as the final root cause. While this is the majority case, in practical scenarios (expected to be extremely rare) where multiple clusters identify distinct local microservices as the most likely causes, an inter-cluster result aggregation mechanism (explained next) is applied to reconcile these decisions.

After completing their independent PPR executions, each cluster c_i produces a ranking vector

$$r_i = [r_i^{m_1}, r_i^{m_2}, \ldots, r_i^{m_k}, r_i^{c_1}, r_i^{c_2}, \ldots, r_i^{c_n}]$$

where the first k elements correspond to its internal microservices (m_j) and the remaining n elements represent connected clusters (c_j). The term $r_i^{c_j}$ denotes the anomaly likelihood assigned to cluster c_j by cluster c_i.

Next, each cluster integrates its internal anomaly profile and its inferred external influences from other connected clusters to construct its global perspective vector as follows:

$$r_i^{\text{global}} = [r_i^{\text{local}}, r_i^{c_1 \text{ influence}}, r_i^{c_2 \text{ influence}}, \ldots, r_i^{c_n \text{ influence}}]$$

Here, $r_i^{\text{local}} = [r_i^{m_1}, r_i^{m_2}, \ldots, r_i^{m_k}]$, which corresponds to the internal anomaly profile, represents the anomaly likelihoods of microservices within c_i. For each connected cluster c_j, the influence component $r_i^{c_j \text{ influence}}$ is obtained as:

$$r_i^{c_j \text{ influence}} = r_i^{c_j} \cdot r_j^{\text{local}}$$

The idea behind this formulation is to distribute the anomaly likelihood assigned to cluster c_j (by cluster c_i) proportionally across c_j's local anomaly scores, capturing how c_i perceives c_j's influence.

Finally, the global perspectives from all clusters are averaged to derive a unified anomaly probability vector:

$$r_i^{\text{combined}} = \frac{1}{N} \sum_{i=1}^{N} r_i^{\text{global}}$$

The microservice with the highest likelihood in r_i^{combined} is selected as the final root cause. This aggregation ensures that all participating clusters contribute to the final decision, enhancing robustness in rare cases of inter-cluster anomaly propagation.

5.4 Performance Evaluation

In this section, we discuss the evaluation results of the proposed decentralized RCL approach. First, in subsection 5.4.1, we explain the details of the experimental setup used for evaluation together with implementation details. Next, in subsection 5.4.2, we discuss the results of evaluating the proposed method against its centralized counterpart, which is commonly referenced in the existing literature [15, 101] and serves as our baseline approach. Following this, in subsection 5.4.3, we conduct a further analysis of our decentralized results in the context of both single-cluster and multi-cluster anomaly propagation cases. We then investigate the impact of increasing cluster counts on localization accuracy in subsection 5.4.4, followed by a large-scale scalability analysis on iAnomaly-generated datasets in subsection 5.4.5. Finally, subsection 5.4.6 discusses the overall findings, with a particular focus on efficiency, communication overhead, and scalability in edge environments.

5.4.1 Experimental setup and implementation details

We evaluated the proposed decentralized RCL approach using two complementary datasets: the publicly available MicroCERCL dataset[1] [52] and large-scale datasets generated using the iAnomaly framework introduced through chapter 3.

MicroCERCL dataset: This dataset represents the first large-scale benchmark for cloud–edge collaborative microservice systems and remains the most comprehensive hybrid deployment dataset available to date. It contains data collected from 81 microservices belonging to four applications: SockShop, Hipster, Bookinfo, and the newly introduced AI-Edge. These microservices are deployed across four cloud servers and two groups of two edge servers, following a communication frequency-based application placement strategy.

To reflect realistic production environments, the dataset integrates a wide range of anomalies. Using ChaosMesh, the authors injected application-level anomalies, such as CPU resource exhaustion in containers, memory leaks, and network latency. Additionally, Linux kernel traffic control (TC) was employed to simulate kernel-level network failures—including packet loss, duplication, corruption, disorder, delay, and jitter—between cloud and edge nodes. Consequently, the dataset captures both communication- and colocation-induced anomaly propagation patterns.

From the available dataset, we selected 383 scenarios that provide adequate diversity in fault types. Among them, 237 scenarios (denoted SH) correspond to cases where the root-cause microservice belongs to the SockShop application, while the remaining 146 scenarios (denoted HH) involve root causes from the Hipster application. For each scenario, we extracted trace information to construct topology graphs, and corresponding metrics were used for anomaly detection and RCL .

[1]https://github.com/WDCloudEdge/MicroCERCL

iAnomaly-based datasets: Although MicroCERCL provides high deployment realism and anomaly diversity, its graph sizes are limited for analyzing scalability under increasing system complexity. To evaluate computational efficiency under larger input sizes, we therefore employed the dataset generation simulator provided by iAnomaly, which is influenced by the iAnomaly emulator to preserve execution realism.

The iAnomaly framework enables the generation of service dependency graphs ranging from 50 to 2,500 nodes, thereby supporting systematic scalability analysis. Unlike purely synthetic simulators, the iAnomaly generator populates graphs using anomalous traces derived from real execution data, ensuring realistic temporal and causal characteristics. The underlying workloads span diverse IoT applications, from lightweight sensor pipelines to computation-intensive services such as face detection and recognition, and are executed under a wide range of injected anomalies such as resource saturation, service failures, and network congestion.

Implementation details: All experiments, including hyperparameter tuning, were conducted on the Spartan HPC cluster[2]. The implementation was performed in Python 3.10 using PyTorch 2.2[3], SciPy 1.13[4], and Scikit-learn 1.1[5].

To construct decentralized subgraphs, we first applied Algorithm 5 to cluster the edge devices in each scenario based on their communication and colocation dependencies. The resulting cluster assignments were then used to generate corresponding topology subgraphs using Algorithm 6. Both algorithms were implemented in Python. To determine the optimal threshold for merging clusters (denoted as τ), which is a key hyperparameter required for Algorithm 5, we employed Tree-structured Parzen Estimator (TPE)-based Bayesian optimization [153]. In the MicroCERCL experiments, all scenarios produced between 2 and 4 clusters per scenario.

For anomaly detection, we adopted the BIRCH clustering-based algorithm from Scikit-learn, following the configuration recommended by the MicroCERCL authors. Specifically, the anomaly sensitivity threshold β was set to 0.07, balancing anomaly detection

[2]https://dashboard.hpc.unimelb.edu.au/
[3]https://pytorch.org/
[4]https://scipy.org/
[5]https://scikit-learn.org/

accuracy and noise reduction.

The proposed anomaly scoring mechanism (Section 5.3.1) was implemented using the SciPy library. Subsequently, the PPR-based RCL algorithm—used in both decentralized and centralized variants—was implemented in Python. Its hyperparameters, α and ε, were tuned using the same TPE-based Bayesian optimization approach [153].

To support reproducibility and facilitate future research, we have publicly released the full implementation, configuration files, and preprocessing scripts[6].

The next subsection presents a detailed comparison between our decentralized RCL approach and its centralized baseline.

5.4.2 Evaluation against the centralized baseline

Before evaluating the decentralized RCL approach, we first validate the centralized variant of our method, which utilizes the PPR-based localization algorithm. This step ensures that our implementation achieves comparable performance to the existing MicroCERCL benchmark [52], thereby establishing it as a reliable baseline for subsequent comparison.

Table 5.2: Comparison of centralized RCL results on the MicroCERCL dataset

Application	**Acc@1**	**Acc@2**	**Acc@3**	**Acc@5**	**Acc@10**	**MAR**	**MRR**
MicroCERCL (GNN-based centralized)							
HH	0.632	0.756	0.796	0.833	0.896	–	–
SH	0.607	0.732	0.792	0.849	0.907	–	–
Our baseline (PPR-based centralized)							
HH	0.4315	0.6507	0.8082	0.9726	1.0000	2.2877	0.6352
SH	0.4979	0.7300	0.8270	0.9451	0.9958	2.1688	0.6817

Table 5.2 compares the results of our centralized PPR-based approach with those reported in the MicroCERCL paper, where a GNN-based centralized RCL model was used. The performance is evaluated using the standard Accuracy@k metric, which measures the proportion of test cases where the true root-cause microservice appears within the top-k ranked predictions. In our evaluation, we report results for $k = 1, 2, 3, 5$, and 10, covering both precise and broader localization ranges. In addition, we report the Mean

[6]`https://github.com/Cloudslab/efficient_rcl_framework`

Average Rank (MAR), which indicates the average position of the true root cause (lower values are better), and the Mean Reciprocal Rank (MRR), which reflects how early the correct root cause appears in the ranked list (higher values are better).

As shown in Table 5.2, our centralized PPR-based approach demonstrates competitive localization performance relative to MicroCERCL's GNN-based model. While the top-1 accuracy (Accuracy@1) is moderately lower, the performance steadily improves at larger values of k, reaching near-perfect Accuracy@10 in both application cases. This trend indicates that PPR effectively ranks the true root-cause microservice within a small set of top candidates, even without model training or feature learning.

The average ranking performance, reflected by MAR and MRR, further confirms this observation. For both applications, the MAR values of 2.29 (HH) and 2.17 (SH) indicate that the true root cause typically appears near the second position in the ranked list. Meanwhile, MRR values of 0.64 and 0.68 imply that in many scenarios the correct root cause appears at rank 1 or 2, thus remaining consistently close to the top of the ranking list. Overall, these results establish our centralized PPR-based implementation as a strong and reliable baseline for comparison with the proposed decentralized RCL approach.

Having established the centralized PPR-based RCL model as a valid baseline, we next evaluate the performance of our proposed decentralized RCL approach. This evaluation focuses on two key aspects: localization accuracy and localization efficiency. Similar to the centralized evaluation, we use the Accuracy@k (for $k = 1, 2, 3, 5$, and 10), Mean Average Rank (MAR), and Mean Reciprocal Rank (MRR) metrics to measure localization accuracy. In addition, to assess the efficiency improvement achieved through decentralization, we introduce two new metrics: Average Time Reduction Percentage and Total Time Reduction Percentage.

The Average Time Reduction Percentage quantifies the mean percentage reduction in localization time across all scenarios. It is defined as

$$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{t^{(i)}_{\text{centralized}} - t^{(i)}_{\text{decentralized}}}{t^{(i)}_{\text{centralized}}} \times 100\right),$$

where $t^{(i)}_{\text{centralized}}$ and $t^{(i)}_{\text{decentralized}}$ denote the localization times under the centralized and decentralized settings for the i^{th} scenario, respectively, and N is the total number of scenarios. This metric reflects the average per-scenario improvement achieved through decentralization.

The Total Time Reduction Percentage, on the other hand, considers the aggregate time across all scenarios, defined as

$$\frac{\sum t_{\text{centralized}} - \sum t_{\text{decentralized}}}{\sum t_{\text{centralized}}} \times 100.$$

Together, these two metrics capture both per-scenario consistency (Average Time Reduction Percentage) and overall efficiency gain (Total Time Reduction Percentage).

Table 5.3: Centralized vs. decentralized RCL performance

Application	**Acc@1**	**Acc@2**	**Acc@3**	**Acc@5**	**Acc@10**	**MAR**	**MRR**	**Avg. Time Red. (%)**	**Total Time Red. (%)**
Centralized baseline									
HH	0.4315	0.6507	0.8082	0.9726	1.0000	2.2877	0.6352	–	–
SH	0.4979	0.7300	0.8270	0.9451	0.9958	2.1688	0.6817	–	–
Proposed decentralized approach									
HH	0.5616	0.7671	0.8767	0.9863	1.0000	1.8630	0.7295	17.41	21.34
SH	0.6076	0.8650	0.9325	0.9831	1.0000	1.6582	0.7733	33.22	33.59

As shown in Table 5.3, the decentralized approach consistently outperforms the centralized baseline across both application groups (HH and SH). In terms of accuracy, the decentralized method improves Accuracy@1 from 0.43 (HH) and 0.50 (SH) to 0.56 and 0.61, respectively, while achieving even higher scores for larger k values. Both MAR and MRR values also demonstrate notable gains, indicating that the true root-cause microservice appears higher in the ranked list under decentralized processing. This improvement in accuracy can be attributed to the noise reduction effect introduced by decentralization: by performing RCL within smaller, communication-aware clusters, each subgraph captures more localized dependencies and avoids the propagation of irrelevant information from distant microservices.

In terms of efficiency, the results indicate substantial time reductions, with 17–21% improvement for HH and over 33% for SH in both average and total time reduction percentages. These results highlight that decentralization not only maintains but, in many cases, enhances localization accuracy, while significantly reducing localization time, thus achieving a more effective balance between accuracy and efficiency.

These results collectively confirm that decentralization leads to clear gains in both accuracy and efficiency compared to the centralized baseline. However, the extent of these improvements can depend on how anomalies propagate within the system. To better understand the influence of anomaly propagation characteristics, the next subsection analyzes the decentralized results in greater depth by separating the evaluation into single-cluster and multi-cluster cases.

5.4.3 Detailed analysis of decentralized results

While the previous subsection presented the overall performance of the decentralized RCL approach, a more granular analysis could provide deeper insights into its behavior under different anomaly propagation patterns. Specifically, we categorize the 383 evaluation scenarios (237 SH and 146 HH) into single-cluster propagation and multi-cluster propagation cases. This categorization enables us to assess whether the decentralized approach consistently maintains localization accuracy and efficiency regardless of whether the anomaly remains confined to a single cluster or propagates across multiple clusters.

Table 5.4: Decentralized RCL performance broken down by anomaly propagation type

Propagation Type	**Case**	**# Scen.**	**Acc@1**	**Acc@2**	**Acc@3**	**Acc@5**	**Acc@10**	**MAR**	**MRR**	**Avg. Time Red. (%)**	**Total Time Red. (%)**
Single-cluster	HH	105	0.5333	0.7429	0.8476	0.9810	1.0000	1.9714	0.7079	23.12	24.02
	SH	206	0.5680	0.8447	0.9223	0.9806	1.0000	1.7379	0.7488	33.89	34.44
Multi-cluster	HH	41	0.6341	0.8293	0.9512	1.0000	1.0000	1.5854	0.7846	2.77	14.07
	SH	31	0.8710	1.0000	1.0000	1.0000	1.0000	1.1290	0.9355	28.79	27.41

Table 5.4 summarizes the results for both categories, reporting the same set of accuracy and efficiency metrics — Accuracy@k (for $k = 1, 2, 3, 5$, and 10), Mean Average Rank (MAR), Mean Reciprocal Rank (MRR), and the average/total time reduction percentages.

The majority of scenarios (311 out of 383) fall under the single-cluster anomaly propagation category, where the anomaly originating from the root-cause microservice propagates only within its local cluster. In these cases, our decentralized method demonstrates a substantial efficiency improvement — achieving over 23–34 % reduction in localization time on average — while maintaining strong accuracy levels (Accuracy@3 >0.84 for both HH and SH). These results demonstrate that when the anomaly impact is localized, decentralized inference efficiently uses intra-cluster communication to identify the root cause with minimal overhead.

For the remaining multi-cluster propagation cases (72 out of 383), where anomalies propagate across clusters and the algorithm engages in inter-cluster P2P approximation, the decentralized approach continues to perform competitively. Accuracy metrics remain high (Accuracy@1 ≥ 0.63 for HH and ≥ 0.87 for SH), validating that the cross-cluster coordination process is effective. Among the 72 multi-cluster propagation cases, a very small subset required the additional inter-cluster result aggregation step—specifically when both participating clusters independently identified local anomalies as potential root causes. In these cases, the aggregation step ensured consistent decision-making without sacrificing accuracy. The resulting accuracy values remained stable (Accuracy@1 ≥ 0.63 for HH and ≥ 0.87 for SH), indicating that the inter-cluster result aggregation mechanism effectively reconciles results from different clusters.

As expected, the efficiency gains are somewhat lower (2.77–28.79%) in cross-cluster scenarios due to the additional latency introduced by inter-cluster communication. The impact of the inter-cluster result aggregation mechanism, however, is minimal: since it requires only a single exchange of low-dimensional ranking vectors, its communication overhead is negligible compared to the total P2P coordination time. The slight reduction in overall efficiency is therefore primarily attributed to inter-cluster message synchronization rather than to aggregation itself. Despite this, the method still achieves a notable reduction in localization time while maintaining, or even improving, localization accuracy compared to the centralized baseline.

Overall, this breakdown illustrates that the proposed decentralized approach adapts well to both local and distributed anomaly propagation scenarios, sustaining accuracy while offering consistent reductions in localization time.

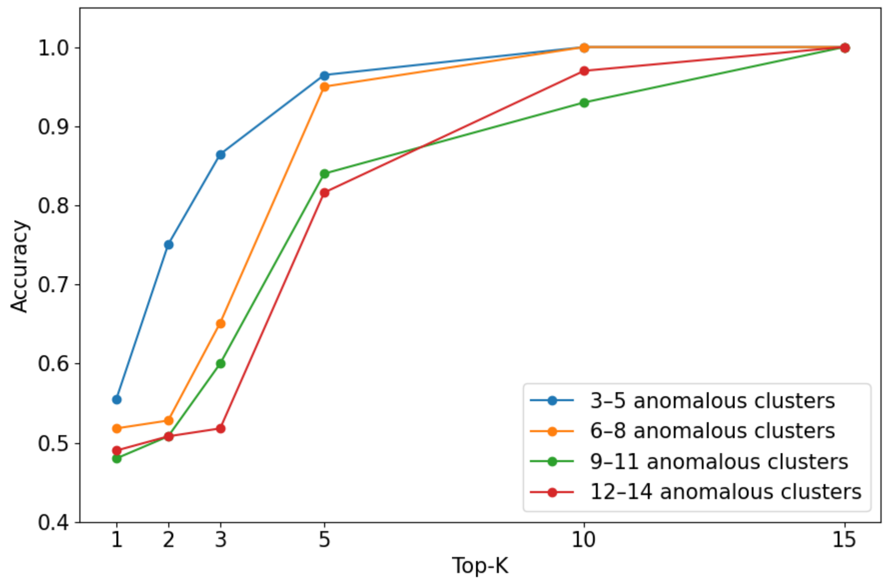


Figure 5.6: Accuracy@K under increasing numbers of anomalous clusters on MicroCERCL (obtained by relaxing clustering constraints to form up to 14 clusters).

5.4.4 Impact of cluster count on localization accuracy

The MicroCERCL evaluation scenarios typically produce 2–4 clusters under our default communication- and colocation-aware clustering algorithm. To understand how localization accuracy behaves as the number of clusters increases, we conduct an additional sensitivity analysis in which we relax the colocation constraint during clustering, resulting in finer-grained partitions and a larger number of clusters per scenario. This stress test allows us to evaluate accuracy under up to 14 clusters on MicroCERCL, without introducing a new synthetic benchmark for accuracy reporting.

We group scenarios into buckets according to the number of anomalous clusters involved in the propagation (3–5, 6–8, 9–11, and 12–14 anomalous clusters) and report Accuracy@K as the number of candidates K increases. Figure 5.6 summarizes the results. Overall, the top-1 accuracy decreases as the number of anomalous clusters grows, reflecting the increased ambiguity introduced by distributed propagation and weaker within-cluster signals. However, the method remains effective at producing short candidate lists: Accuracy@10 remains high across buckets and reaches 100% by top-K (here K=15) in all cases, substantially reducing the manual search space for SRE diagnosis workflows.

This trend indicates that increasing the number of clusters introduces a measurable accuracy–granularity trade-off: finer partitioning improves scalability and reduces local computation, but may weaken the signal concentration that benefits ranking-based localization. We therefore report this behavior as a validity threat and emphasize that cluster granularity should be selected carefully in practice, balancing scalability benefits against potential degradation in top-1 localization accuracy.

5.4.5 Scalability analysis on the iAnomaly dataset

To evaluate scalability and localization efficiency beyond MicroCERCL, we conduct additional experiments using the iAnomaly-based dataset generation simulator, which allows us to systematically vary the size of the service dependency graph from 50 to 2,500 nodes. This controlled setup enables a direct comparison between the centralized PPR baseline and the proposed decentralized RCL approach under increasing graph scale.

To ensure that the evaluation reflects non-trivial and practically relevant settings, we focus exclusively on multi-cluster anomaly propagation scenarios, where coordination across multiple clusters is required. Single-cluster cases are excluded from this analysis, as they can be efficiently handled by local inference and do not meaningfully stress the scalability of the proposed framework.

We report the average end-to-end localization latency under three configurations: (i) Centralized PPR, where the full graph is processed as a single instance; (ii) Decentralized (fixed clustering), where the graph is partitioned into a fixed number of clusters ($|C|=5$) and PPR is executed locally within each cluster, with inter-cluster coordination enabled when required; and (iii) Decentralized (adaptive clustering), where the number of clusters is chosen proportional to graph size (approximately one cluster per 10 nodes), thereby keeping cluster sizes bounded as the graph grows.

Figure 5.7 shows the resulting latency trends. As the number of nodes increases, the centralized baseline exhibits a pronounced increase in localization time, reflecting the growing cost of PPR iterations over an increasingly large dependency graph. The decentralized approach with a fixed number of clusters mitigates this growth, but still shows

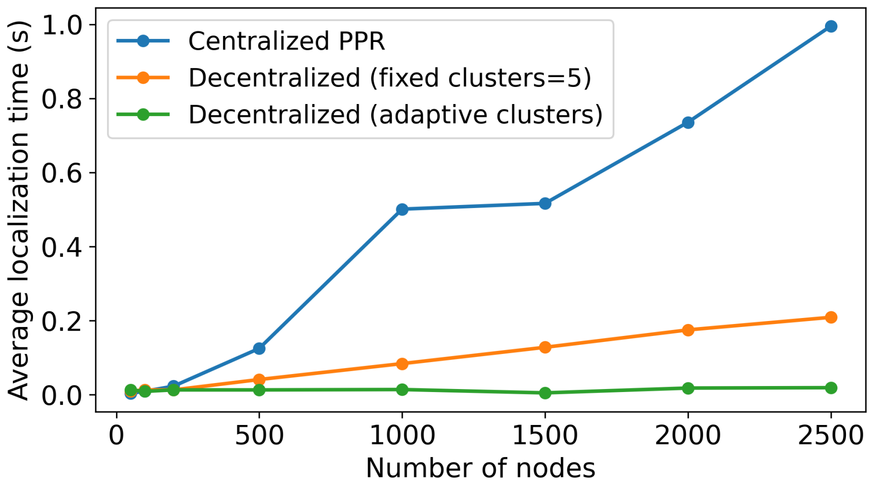


Figure 5.7: Scalability comparison on the iAnomaly dataset

gradual increases in latency, as cluster sizes (and hence the number of intra-cluster edges processed by PPR) expand with overall graph scale.

In contrast, the adaptive clustering configuration maintains near-constant localization time across all evaluated scales. In this setting, the number of clusters increases proportionally with graph size (approximately one cluster per 10 nodes), thereby explicitly evaluating scalability under growing $|C|$. By keeping the expected cluster size approximately bounded, the effective search space per local PPR execution remains stable, and the overall latency is dominated by local computation rather than global graph size. As a result, the proposed decentralized approach consistently outperforms the centralized baseline in terms of localization efficiency under increasing system scale.

5.4.6 Discussion

In summary, the evaluation results demonstrate that the proposed decentralized RCL approach achieves comparable or higher localization accuracy than the centralized baseline, while significantly reducing localization time. The approach performs consistently well across both single-cluster and multi-cluster anomaly propagation scenarios, confirming its effectiveness under diverse anomaly propagation conditions.

The results also validate that the inter-cluster P2P approximation process—designed for multi-cluster anomaly propagation—introduces minimal communication overhead and exerts negligible impact on localization time. This process requires only a one-time exchange of average approximate anomaly scores between connected clusters, involving a limited number of lightweight message exchanges overall (*wait_message* and *avg_approx_anom_score*). Empirical evidence confirms that these exchanges incur almost no waiting periods, as most computations proceed in parallel across clusters, ensuring efficient utilization of resources during coordination.

From a scalability perspective, the communication overhead of the inter-cluster coordination process grows with the connectivity of the cluster adjacency graph rather than directly with the total number of clusters. Let C denote the number of clusters and E_C the number of inter-cluster links involved in an anomaly propagation path. Since C is typically much smaller than the number of microservices N in large-scale edge environments, this design decouples coordination overhead from direct dependence on system size. For each affected link, the P2P approximation requires a lightweight *wait_message* and a two-way exchange of scalar average anomaly scores. Consequently, the total number of coordination messages is proportional to the number of inter-cluster links involved in the propagation. In the worst case of a fully connected cluster graph, the number of such links grows quadratically with C. However, practical microservice deployments are typically sparse, and our communication-aware clustering strategy explicitly minimizes inter-cluster dependencies, resulting in near-linear growth with respect to the number of clusters in most scenarios. Furthermore, the optional result aggregation step is triggered only in rare cases and exchanges only compact ranking vectors, introducing an additional number of messages proportional to C', where $C' \leq C$ denotes the number of anomalous clusters participating in the aggregation process. As a result, communication overhead remains modest even as the number of clusters increases.

Moreover, in the rare cases where multiple clusters identify distinct local anomalies, the results show that the inter-cluster result aggregation mechanism effectively reconciles these outcomes with minimal overhead. Since this step operates on compact ranking vectors rather than raw monitoring data, both communication and computation costs remain negligible. This confirms that the efficiency benefits of decentralization are preserved even in cross-cluster anomaly propagation scenarios.

The additional scalability experiments conducted using the iAnomaly-generated datasets further confirm the efficiency advantages of the proposed decentralized approach under increasing system scale. In particular, the decentralized design consistently outperforms the centralized baseline in terms of localization latency as graph size grows, demonstrating its suitability for large and highly distributed edge deployments.

Finally, beyond the observed improvements in localization time due to faster convergence of the PPR-based RCL approach, the decentralized design is expected to further reduce latency in real deployments by executing RCL at the edge device level. This minimizes data transfer delays and lowers communication overhead compared to traditional centralized approaches, where large volumes of monitoring data must be transported to a central node for analysis.

5.5 Summary

This chapter proposed a decentralized root cause localization (RCL) approach for microservice-based IoT applications deployed in edge computing environments. By leveraging the PPR algorithm within communication- and colocation-aware clusters, the proposed approach reduces localization time without compromising accuracy. The evaluation on MicroCERCL shows that decentralization improves both localization accuracy and efficiency relative to a centralized PPR baseline, and the detailed analysis confirms consistent gains under both single-cluster and multi-cluster anomaly propagation scenarios. Furthermore, scalability experiments conducted using large-scale datasets generated with the iAnomaly framework demonstrate that the proposed decentralized approach consistently outperforms the centralized baseline in terms of localization efficiency under increasing system scale. These results motivate the need for similarly scalable approaches to joint localization and fault analysis, which is addressed in the next chapter.

Chapter 6

A Cascaded Graph Neural Network for Joint Root Cause Localization and Analysis in Edge Computing Environments

While decentralized RCL improves diagnostic efficiency, comprehensive anomaly diagnosis also requires identifying the underlying fault type. Recent GNN-based approaches have demonstrated strong performance for joint localization and analysis, but their reliance on centralized processing over full-system graphs leads to high inference latency and limited scalability in large, distributed edge environments. This chapter proposes a cascaded GNN framework that enables hierarchical and distributed diagnosis through communication-driven clustering. The framework employs a proposal network that operates on individual communication-aware clusters to generate cluster-level outputs, followed by an output network which considers each cluster as a node and operates over the graph of clusters to produce graph-level outputs by considering inter-cluster connectivity. By restricting message passing to structured subgraphs and compact cluster-level representations, the proposed approach significantly reduces computational complexity while preserving critical dependency information. Evaluations on MicroCERCL and large-scale datasets generated using the iAnomaly simulation framework demonstrate that the cascaded architecture achieves diagnostic accuracy comparable to centralized GNN baselines with near-constant inference latency, enabling scalable joint diagnosis in edge environments.

This chapter is derived from:

- **Duneesha Fernando**, Maria A. Rodriguez, and Rajkumar Buyya, "A Cascaded Graph Neural Network for Joint Root Cause Localization and Analysis in Edge Computing Environments", *submitted to IEEE Transactions on Services Computing (TSC), March 2026* (Under Review, available on arXiv:2603.01447).

6.1 Introduction

Root cause analysis (RCA) is a critical component of automated operations in microservices-based systems, as accurate anomaly detection and Root Cause Localization (RCL) alone are insufficient to support timely and effective mitigation. All RCA studies perform RCL, while the analysis can be conducted at various levels [25, 105]. Some RCA studies focus on identifying the specific metrics responsible for the anomaly [102, 103, 108], while others concentrate on determining the type of fault [28–30]. Additionally, there is a more detailed level of RCA that examines issues at the source code level, which is valuable for developing long-term solutions to performance problems [112]. In this work, we focus on the type of RCA that combines fault-type diagnosis together with RCL. This level of analysis provides essential information for autonomous AIOps systems to address performance issues effectively [28].

Although RCL and RCA are often discussed separately, the two tasks are interconnected. They both stem from the same anomalous event and rely on similar signals such as temporal performance patterns, inter-service dependencies, and anomaly propagation behavior. For this reason, recent studies advocate performing RCL and RCA jointly rather than in isolation [28–30]. Supervised GNNs currently represent the state of the art for joint RCL and RCA [110]. By learning RCL and RCA jointly, the model can share early feature extractors (such as Convolutional Neural Network (CNN) layers and Graph Convolutional Network (GCN) layers), allowing it to capture useful temporal patterns and inter-service dependencies that help both tasks. This enables faster and more accurate localization and diagnosis than training separate models [29, 30].

Despite their effectiveness, GNNs have high computational complexity [14, 24]. In edge computing environments, which consist of a large number of devices and microservices, this translates to forming graphs with a large number of nodes as input for the GNN. The presence of such large-scale graphs amplifies the computational cost of message passing, resulting in slower diagnosis and delayed mitigation.

To address this scalability challenge, we propose a cascaded GNN architecture designed for joint RCL and RCA, specifically aiming large-scale edge deployments. Our approach groups microservices into clusters based on their communication dependencies, with each cluster representing a highly interacting group of microservices. The cascaded GNN consists of two networks: the proposal network (P-Net) and the output network (O-Net). P-Net operates on individual clusters and produces a cluster-level output, which reduces the search space by focusing on a confined area. O-Net takes the latent representation of each cluster produced by P-Net and inter-cluster edges to create a graph representing the edge system, as input. By treating each cluster as a node, O-Net further minimizes its search space. As both networks work with reduced search spaces compared to a traditional GNN, the overall complexity of our cascaded network is lower. Although operating within a reduced receptive field may result in some information loss relative to a traditional GNN, our clustering technique effectively groups microservices based on communication dependencies, which mitigates this loss. This ensures that accuracy is maintained while improving efficiency. To the best of our knowledge, ours is the first work to introduce a computationally efficient cascaded GNN framework for joint RCL and RCA in edge computing environments, explicitly addressing the scalability and latency constraints present in large, distributed edge–cloud systems.

We evaluated our proposed cascaded GNN approach on the publicly available MicroCERCL benchmark dataset [52] as well as on large-scale datasets generated using the iAnomaly simulation framework introduced through chapter 3. Experimental results on MicroCERCL show that the cascaded model achieves diagnostic accuracy comparable to a centralized GNN baseline, while scalability experiments on iAnomaly demonstrate near-constant inference time, in contrast to the centralized model whose latency increases with graph size.

The rest of this chapter is organized as follows. Section 6.2 reviews the background and related work on RCL and analysis, joint RCL/RCA techniques, and efficiency-oriented diagnosis methods. Section 6.3 presents a centralized GNN architecture for joint RCL and RCA that serves as a baseline for comparison. Section 6.4 introduces the proposed cascaded GNN framework, including the communication-driven clustering strategy and hierarchical inference design. Section 6.5 reports the experimental evaluation and scalability analysis on both real-world and large-scale simulated datasets. Finally, Section 6.6 concludes the chapter.

6.2 Background and Related Work

In this section, we provide an overview of the background and related work on RCL and analysis in microservice-based edge and cloud systems. We first review existing RCA approaches from the perspective of diagnostic granularity, ranging from coarse-grained localization to fine-grained fault type and code-level analysis. We then discuss recent techniques that perform RCL and RCA jointly, with a particular focus on supervised graph-based methods that represent the current state of the art. Finally, we examine prior efforts on improving the efficiency and scalability of RCA techniques, highlighting the limitations of existing solutions in large-scale edge environments and motivating the need for our cascaded GNN framework.

6.2.1 Levels of Root Cause Analysis

RCA in microservices-based systems spans a spectrum of diagnostic granularity, ranging from coarse-grained localization to fine-grained diagnosis. At the coarsest level, RCL, also referred to as coarse-grained localization, aims to identify the microservice responsible for an observed performance anomaly. RCL is generally considered as the first and most fundamental step of RCA, and is performed, either explicitly or implicitly, by almost all RCA approaches [25, 105]. A number of studies focus exclusively on this task, including MicroRCA [101], MicroGBPM [15], MicroCERCL [52], and MicroEGRCL [106]. While coarse-grained localization can often be achieved rapidly, identifying only the faulty microservice offers limited diagnostic value, as the underlying cause of the anomaly remains unknown, forcing SREs to manually investigate issues such as memory exhaustion, CPU contention, or abnormal request patterns [107, 108]. Moreover, mitigation actions at this level, such as restarting or redeploying an entire service, may fail to address the true root cause, introduce unnecessary disruption to dependent services, and prolong recovery times compared to fine-grained mitigation efforts [105].

Beyond microservice-level localization, RCA techniques increasingly incorporate finer-grained analysis. One intermediate level is culprit metric localization, which seeks to identify the specific performance metrics (e.g., CPU usage, latency, memory consumption) that contribute to the detected anomaly. Representative approaches in this category include MicroDiag [103], AAMR [102], HeMIRCA [108], CausalRCA [105], PDiagnose [109], and Wu et al. [107]. This level of analysis enables more targeted mitigation actions, such as resource scaling or throttling, which often result in faster recovery and lower system disruption than service-level restarts [105]. Both RCL-only approaches and those combining RCL with culprit metric localization are typically formulated as ranking problems, where microservices or metrics are ordered according to their likelihood of being the root cause.

A more advanced level of RCA focuses on fault type identification, which aims to determine the underlying cause of the anomaly itself. At this level, anomalies are attributed to specific fault categories such as CPU hogging, memory leaks, network delay, or I/O contention. Approaches including DiagFusion [30], MicroIRC [29], DejaVu [28], Brandon et al. [111], and MicroHECL [53] explicitly address this problem by classifying anomalies into predefined fault types. Fault type identification offers a key advantage over metric-level diagnosis: it directly maps observed anomalous behavior to semantically meaningful system faults, enabling more precise and automated mitigation actions. Consequently, this class of approaches is commonly formulated as a classification problem.

At the finest granularity, RCA operates at the source-code level, where the objective is to localize performance anomalies to specific code regions or components. Nezha [112] exemplifies this category by identifying performance issues directly at the code level. While such approaches are valuable for developing long-term and permanent fixes, they often require extensive instrumentation of application source code or runtime systems, introducing significant overhead in production environments [107]. As a result, source-code-level RCA is generally less suitable for online diagnosis and real-time mitigation, particularly in latency-sensitive systems.

In this work, we focus on an RCA setting that combines fault type identification with RCL. This level of diagnosis provides essential information for autonomous AIOps systems to effectively address performance issues and aligns closely with the notion of actionable diagnosis [28]. To be truly actionable, a fault localization solution must provide two critical pieces of information simultaneously: (1) where the failure occurs (the faulty component, via RCL), and (2) what kind of failure occurs (the fault type). Together, this combination is often referred to as a failure unit [28]. Different fault types (e.g., memory leaks versus CPU exhaustion) require fundamentally different mitigation strategies. An approach that only localizes the faulty service cannot autonomously determine whether to scale resources, adjust scheduling policies, or apply memory management techniques. By jointly identifying both the faulty component and the fault type, RCA results become directly actionable, enabling AIOps systems to move beyond simple alerting and toward targeted, automated mitigation strategies [23].

6.2.2 Joint RCL and RCA techniques

Although RCL and RCA are often treated as separate tasks, they are interconnected. The location of the fault and the nature of the fault both originate from the same anomalous event and are driven by shared system signals, including temporal performance patterns, inter-service dependencies, and anomaly propagation behavior. For this reason, approaches that combine RCL with fault type identification have gained increasing attention in recent years [28–30, 105].

Among the three main categories of RCA techniques, 1) deep learning (DL), 2) pattern recognition, and 3) causal inference, as identified by Wu et al. [103], joint RCL and RCA has predominantly been explored using DL and pattern-based methods [105]. DL approaches, in particular, have focused on supervised GNNs, which currently represent the state of the art for joint RCL and fault type identification [110]. Representative examples include DiagFusion [30], which employs GNNs for joint diagnosis, MicroIRC [29], which combines a personalized random walk algorithm with a GNN model, and DejaVu [28], which utilizes GATs. On the other hand, pattern-based approaches, such as the study by Brandon et al. [111], perform joint diagnosis by matching anomalous subgraphs against a predefined library of fault patterns.

Supervised GNNs are now the state of the art in joint RCL and RCA due to their ability to model complex inter-service dependencies. While some may argue that supervised methods struggle to identify previously unseen fault types, empirical evidence indicates that a significant proportion of failures in operational systems are recurring, with reports suggesting this figure can be as high as 74% [25]. Although supervised approaches typically require substantial labeled data and training effort, purely unsupervised methods often fail to achieve fully automated, end-to-end RCA [52]. For these reasons, GNN-based supervised approaches can serve as a practical and effective starting point for automating AIOps, especially in edge environments.

6.2.3 Efficient RCL and RCA techniques

Work	**Cloud-only/Cloud-Edge**	**GNN**	**RCL/RCA Type**	**Accuracy Focus**	**Efficiency Focus**	**Remarks**
[15], [89], [14], [108], [103], [105], [107], [29]	Cloud-only	Mixed	RCL; RCA (Metric / Fault Type)	✓	×	Focused solely on accuracy
[106], [101], [102], [110], [111], [28], [30], [158], [112]	Cloud-only	Mixed	RCL; RCA (Metric / Fault / Resource Type)	✓	△	Claim faster inference; efficiency not a design priority
PDiagnose [109]; MicroHECL [53]	Cloud-only	No	RCL; RCA (Metric / Fault Type)	△	✓	Focused on improving efficiency; may compromise accuracy
MicroCERCL [52]	Cloud–Edge	Yes	RCL only	✓	×	Accuracy focus; long inference time reported
Kalinagac et al. [27]	Cloud–Edge	No	RCA (Fault Type)	✓	×	Focused solely on accuracy
Our work	Cloud–Edge	Yes	Joint RCL + RCA (Fault Type)	✓	✓	Cascaded GNN with clustering; balances accuracy and efficiency
✓= Primary focus / addressed; △= Partially addressed / may compromise; ×= Not addressed / ignored						

Table 6.1: Summary of cloud and edge RCL/RCA techniques: accuracy vs efficiency focus

Despite their effectiveness, GNNs are known to incur non-trivial computational overhead, primarily due to the cost of iterative message passing over graph structures [14, 24]. In edge computing environments, which often consist of a large number of distributed devices, microservices, and their interactions, this results in graphs with a large number of nodes and edges being provided as input to the GNN. As the graph size increases, the computational cost of GNN inference grows proportionally with the number of nodes, edges, and message-passing layers, leading to longer diagnosis latency. In particular, large-scale graphs amplify the cost of neighborhood aggregation and feature propagation, which must be performed for each node across multiple GNN layers. This added complexity can slow down root cause identification and fault diagnosis, delay-ing mitigation actions. This delay is particularly problematic in latency-sensitive edge environments where timely response to anomalies is critical.

To address this scalability challenge, we propose a cascaded GNN architecture for joint RCL and RCA, specifically targeting large-scale edge deployments. The framework consists of two sequential networks operating on clusters formed from highly interacting microservices based on communication dependencies. By applying both stages of the cascaded model to reduced and structured subgraphs, the proposed approach significantly limits the effective search space, thereby lowering computational complexity compared to conventional GNNs. Communication-based clustering preserves critical dependency information, minimizing information loss associated with reduced receptive fields and maintaining diagnostic accuracy. As a result, the proposed framework achieves an effective balance between efficiency and accuracy.

A comparison of cloud and edge RCL/RCA techniques in terms of their focus on efficiency (and accuracy as well) is shown in Table 6.1. The majority of studies on cloud RCL/RCA have primarily focused on improvements in accuracy, evaluating this single aspect [15], [89], [14], [108], [103], [105], [107], [29]. Some research has also addressed efficiency, particularly the localization times of their proposed methods. For instance, AAMR [102], MicroEGRCL [106], and Grace [110] claim to provide faster inference times. However, none of these studies have specifically designed their approaches with efficiency as a priority. In addition, existing edge-oriented solutions, including MicroCERCL [52] and the RCA framework proposed by Kalinagac et al. [27], do not explicitly incorporate efficiency considerations. In particular, MicroCERCL reports longer inference times than unsupervised heuristic methods, largely due to the complexity of the network [52].

MicroHECL [53] and PDiagnose [109] are cloud RCL/RCA techniques that specifically aim to improve efficiency, particularly by providing faster localization speeds. MicroHECL achieves efficiency by efficiently traversing the service dependency graph and using pruning techniques to eliminate irrelevant service calls during anomaly propagation chain analysis, which further enhances efficiency. On the other hand, PDiagnose tries to reach efficiency by removing the computationally heavy dependency graph-building phase and utilizing a vote-based localization process on an anomaly queue.

However, both approaches adopt relatively simplistic strategies that may compromise diagnostic accuracy. Given the need to strike an appropriate balance between effectiveness and efficiency, our proposed method leverages a GNN-based approach to reduce diagnosis time while maintaining high accuracy.

Overall, existing RCA techniques have made substantial progress in improving diagnostic accuracy, particularly through supervised GNN-based joint RCL and RCA models. However, most of these approaches are designed under a centralized processing assumption and operate on full-system graphs, which leads to high computational overhead in large-scale edge environments. To better understand the strengths and limitations of such centralized designs, we first review a representative centralized joint RCL/RCA GNN architecture in the next section. This analysis provides the foundation for introducing our proposed cascaded framework, which explicitly addresses the scalability and latency challenges identified in prior work.

6.3 Centralized Joint RCL/RCA GNN Architecture

In this section, we present a representative centralized GNN architecture for joint RCL and fault type identification, which serves as a baseline for subsequent comparison with our proposed cascaded framework. We first describe the graph formulation used to model edge computing environments and their microservice dependencies (Section 6.3.1), followed by the construction of GNN input representations from node- and edge-level time-series metrics (Section 6.3.2). We then review common design principles of centralized GNN-based joint RCL/RCA models (Section 6.3.3). Finally, we introduce the overall centralized architecture adopted in this work (Section 6.3.4).

6.3.1 Graph formulation for edge computing environments

Let the edge infrastructure consist of a set of edge devices

$$\mathcal{E} = \{e_1, e_2, \ldots, e_{|\mathcal{E}|}\},$$

and a set of microservices deployed within the environment

$$\mathcal{M} = \{m_1, m_2, \ldots, m_{|\mathcal{M}|}\}.$$

Each microservice $m_i \in \mathcal{M}$ is deployed on exactly one edge device. The deployment mapping can be defined as

$$\delta : \mathcal{M} \to \mathcal{E},$$

where $\delta(m_i)$ gives the device on which microservice m_i is deployed.

GNN-based analysis requires the system to be represented as a graph, where nodes capture computational entities and edges reflect structural or runtime relationships. In the context of edge-cloud microservice systems, we model the environment using a topology graph

$$\mathcal{G} = (\mathcal{V}, \mathcal{L}),$$

where the set of vertices $\mathcal{V}$ consists of both microservices and the edge devices on which they are deployed:

$$\mathcal{V} = \mathcal{M} \cup \mathcal{E}$$

The set of edges $\mathcal{L}$ represents relationships between these vertices and is composed of two distinct types:

$$\mathcal{L} = \mathcal{L}_{\text{comm}} \cup \mathcal{L}_{\text{dep}}$$

1. Communication edges (L_{comm}) capture runtime interactions among microservices, derived from trace data:

   $$\mathcal{L}_{\text{comm}} \subseteq \mathcal{M} \times \mathcal{M},$$

   where $(m_i, m_j) \in \mathcal{L}_{\text{comm}}$ if microservice m_i communicates with m_j.

2. Deployment edges (L_{dep}) link each microservice to the edge device on which it is hosted, obtained from the deployment configuration:

   $$\mathcal{L}_{\text{dep}} = \{(m_i, \delta(m_i)) \mid m_i \in \mathcal{M}\}.$$

This unified heterogeneous graph structure enables joint reasoning over system execution behavior and physical deployment constraints.

Both nodes and edges are enriched with time-series features derived from monitoring metrics. Microservice nodes incorporate both resource-level metrics (e.g., CPU utilization, memory usage, disk I/O) and application-level metrics reflecting service behavior, whereas edge-device nodes are represented solely by resource-level metrics that describe device-level resource consumption. Edge features capture communication behavior (e.g., p50/p90/p99 latency percentiles), reflecting runtime dependencies and potential performance bottlenecks.

6.3.2 GNN input representation

From a modeling perspective, a GNN operates on three fundamental components: node features, graph connectivity, and (optionally) edge features. Let the node feature tensor be defined as

$$X \in \mathbb{R}^{|\mathcal{V}| \times F \times T},$$

where each node is represented by F feature dimensions observed over T timesteps. Since the graph contains $|\mathcal{V}|$ nodes, the dimensions of the resulting feature tensor become $(|\mathcal{V}|, F, T)$.

The structural connectivity of the system is encoded using the edge index

$$\mathcal{L} \subseteq \mathcal{V} \times \mathcal{V},$$

capturing communication and deployment relationships among nodes.

Link-level properties and communication statistics are incorporated through the edge feature tensor which is defined as

$$E \in \mathbb{R}^{|\mathcal{L}| \times F_e \times T},$$

where each edge is characterized by F_e feature dimensions measured over T timesteps. Given $|\mathcal{L}|$ edges in the graph, the dimensions of tensor E become $(|\mathcal{L}|, F_e, T)$.

These components together constitute the input to the GNN model, enabling representation learning over both the structural and temporal characteristics of the topology graph.

6.3.3 Background on centralized GNNs for joint RCL/RCA

When examining supervised GNN-based approaches employed by RCL and fault type identification studies, two dominant modeling paradigms can be observed. In the first paradigm, a single static graph is constructed, where temporal dependencies are learned first by extracting compact representations from time-series data, followed by spatial dependency learning using a GNN [28–30]. In the second paradigm, a sequence of graphs is constructed (typically one graph per timestamp or time window) where spatial features are learned independently for each graph, and temporal dependencies are subsequently modeled across the graph sequence using temporal aggregation or sequence models [52, 110].

In the first category of approaches, temporal feature extraction is commonly performed using techniques such as LSTMs, CNNs or other sequence encoders[28, 52, 89]. The learned temporal embeddings are then used as node- and/or edge-level features for graph-based learning. We adopt this first paradigm due to its advantages in terms of computational efficiency, model scalability, and clear separation of temporal and structural learning, which is particularly beneficial for large-scale microservice systems [28, 30].

Following temporal feature extraction and message passing through the GNN layers, task-specific prediction heads are applied to support both RCL and fault type identification (RCA). These tasks differ in granularity and objective: RCL aims to identify the specific microservice responsible for the anomaly, whereas RCA determines the underlying anomaly type or fault category.

Within GNN-based frameworks, the RCL problem is typically formulated as a node-level ranking task [30, 52, 89]. The model produces a score for each microservice node, which is transformed into a series of probability values over all nodes (e.g., via a softmax activation), allowing the most likely faulty service to be ranked highest. In contrast, fault type identification is commonly framed as a graph-level multi-class classification task, where a graph-level readout layer aggregates learned node representations (e.g., using

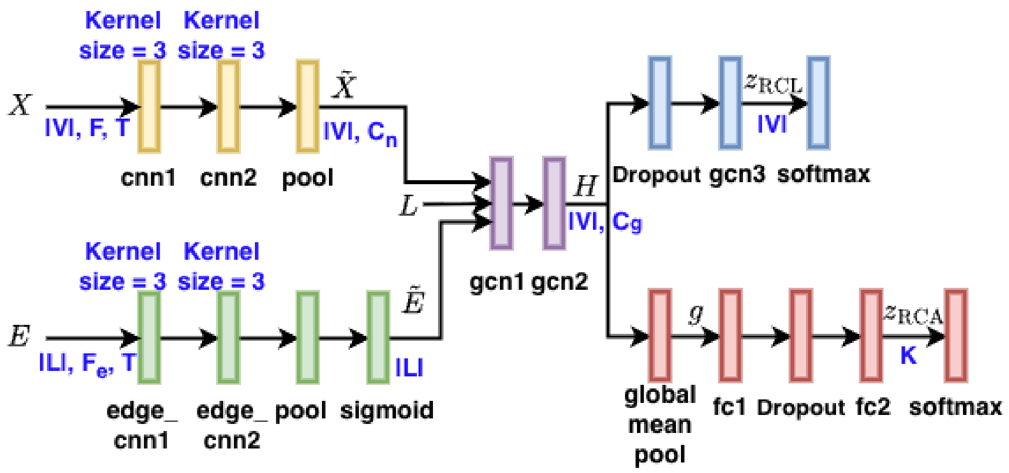


Figure 6.1: Centralized GNN architecture for joint RCL–RCA

mean/max pooling or attention-based pooling), followed by a classification head that predicts the probability of each fault type [29, 30].

Following this design, two output heads are employed: a node-level prediction head for RCL and a graph-level prediction head for RCA, which can be trained either independently or jointly [30, 89]. While some studies unify these outputs into a single prediction task [28, 29], we adopt a dual-head formulation.

6.3.4 Overview of centralized joint RCL/RCA GNN architecture

Our baseline GNN architecture as shown in Figure 6.1, follows the first modeling paradigm described in subsection 6.3.3, where temporal patterns are learned prior to graph-based spatial reasoning. The architecture is composed of three sequential stages: (1) temporal feature extraction from nodes and edges, (2) spatial dependency learning using GCN-based message passing, and (3) two task-specific prediction heads for RCL and RCA. The early layers (temporal encoders and structural graph encoders) are shared across both tasks, enabling the network to jointly leverage temporal metric evolution and microservice dependency structure.

Stage 1 - Temporal Feature Extraction

To obtain compact representations from node- and edge-level time-series metrics, we employ separate 1D-CNN based temporal encoders, consistent with the feature processing approach discussed previously.

(a) Node feature encoder. The input node tensor $X \in \mathbb{R}^{|\mathcal{V}| \times F \times T}$ is passed through two CNN layers, followed by pooling across the temporal dimension:

$$X \xrightarrow{\text{CNN}_1} \xrightarrow{\text{CNN}_2} \xrightarrow{\text{Temporal Pool}} \tilde{X} \in \mathbb{R}^{|\mathcal{V}| \times C_n}, \tag{6.1}$$

where C_n is the output embedding dimension after temporal pooling. The pooling operation aggregates temporal information into a fixed-length representation, allowing downstream GNN layers to operate on time-compressed embeddings rather than raw sequences.

(b) Edge feature encoder. Similarly, the edge attribute tensor $E \in \mathbb{R}^{|\mathcal{L}| \times F_e \times T}$ is processed using two CNN layers, followed by pooling across the temporal dimension:

$$E \xrightarrow{\text{CNN}_1} \xrightarrow{\text{CNN}_2} \xrightarrow{\text{Temporal Pool}} \tilde{E} \in \mathbb{R}^{|\mathcal{L}|}. \tag{6.2}$$

A sigmoid function is then applied to reduce edge weights into a singular value in the range of $[0, 1]$, facilitating the learning of necessary information for subsequent message passing stages.

These embeddings form the initial node and edge features for the GNN layers.

Stage 2 - Spatial Graph Learning

The temporally encoded embeddings are passed through two stacked GCN layers to learn spatial dependencies across the topology graph:

$$(\tilde{X}, L, \tilde{E}) \xrightarrow{\text{GCN}_1} \xrightarrow{\text{GCN}_2} H \in \mathbb{R}^{|\mathcal{V}| \times C_g}, \tag{6.3}$$

where C_g is the hidden dimension of the resulting structural representation. The edge embeddings $\tilde{E}$ are used as edge weights, enabling the GCN to incorporate communication intensity or latency sensitivity when propagating information across the service graph.

These GCN layers act as structural encoders, shared by both downstream tasks. This shared representation is critical because both localization and fault type inference benefit from understanding inter-service influence patterns and anomaly propagation dynamics.

Stage 3 - Task-Specific Prediction Heads

After spatial encoding, the network branches into two independent output heads.

(a) RCL Head – Node-level ranking. Designed to identify the faulty microservice, this branch operates directly on node embeddings:

$$H \xrightarrow{\text{Dropout}} \xrightarrow{\text{GCN}_3} z_{\text{RCL}} \in \mathbb{R}^{|\mathcal{V}|}, \tag{6.4}$$

z_{RCL} is the vector of logit values corresponding to all nodes in the graph. A softmax activation converts the logit values into probabilities, identified as $\hat{p}_{\text{RCL}}$.

(b) RCA Head – Graph-level multi-class classification. Here, node embeddings are aggregated via global mean pooling to obtain a graph-level representation g, which is then processed through two fully connected layers with dropout to generate class logits:

$$H \xrightarrow{\text{GlobalMeanPool}} g \xrightarrow{\text{FC}_1} \xrightarrow{\text{Dropout}} \xrightarrow{\text{FC}_2} z_{\text{RCA}} \in \mathbb{R}^{K}, \tag{6.5}$$

where K is the number of anomaly/fault types. A softmax activation converts the logit values into probabilities required for fault diagnosis, identified as $\hat{p}_{\text{RCA}}$.

For all hidden GCN and CNN layers explained previously, we employ Exponential Linear Unit (ELU) as the activation function.

Loss Formulation

Subsequently, both prediction heads are optimized simultaneously using a joint loss function,

$$\mathcal{L}_{\text{total}} = \lambda_{\text{RCL}} \cdot \mathcal{L}_{\text{RCL}} + \lambda_{\text{RCA}} \cdot \mathcal{L}_{\text{RCA}}, \tag{6.6}$$

where $\mathcal{L}_{\text{RCL}}$ and $\mathcal{L}_{\text{RCA}}$ denote node-level and graph-level negative log-likelihood losses, for RCL and RCA, respectively, and λ_{RCL} and λ_{RCA} are weighting coefficients.

Let $\hat{y}_{\text{RCL}}$ denote the one-hot encoded vector indicating the ground-truth root cause node. RCL loss for a single graph is defined as

$$\mathcal{L}_{\text{RCL}} = -\frac{1}{|\mathcal{V}|}\sum_{i=1}^{|\mathcal{V}|} y^i_{RCL} \log(p^i_{RCL}), \tag{6.7}$$

where y^i_{RCL} is the i^{th} element of $\hat{y}_{\text{RCL}}$, and p^i_{RCL} denotes the predicted probability of node i being the root cause, and $|\mathcal{V}|$ is the number of nodes in the graph.

Similarly, for the RCA task, let $\hat{y}_{\text{RCA}}$ denote the one-hot encoded vector indicating the fault type over K fault classes. RCA loss for a single graph is defined as

$$\mathcal{L}_{\text{RCA}} = -\frac{1}{|\mathcal{K}|}\sum_{j=1}^{|\mathcal{K}|} y^j_{RCA} \log(p^j_{RCA}), \tag{6.8}$$

where y^j_{RCA} is the j^{th} element of $\hat{y}_{\text{RCA}}$, and p^j_{RCA} denotes the predicted probability of fault type j being the root cause.

Unless stated otherwise, we set $\lambda_{\text{RCL}} = \lambda_{\text{RCA}} = 0.5$ in our implementation. Joint optimization encourages the shared GCN encoders to learn representations useful to both localization and diagnosis tasks, improving convergence speed and predictive accuracy.

6.4 Cascaded Joint RCL/RCA GNN Architecture

While the baseline centralized GNN model is effective, it does not scale efficiently for large microservice deployments. The computational cost of each GCN layer is $\mathcal{O}(N^2)$, where $N = |\mathcal{V}|$ denotes the number of nodes in the graph [14, 24]. Since multiple such layers are employed to perform global message passing, the overall computational overhead is further amplified. As the microservice ecosystem expands, every layer must process all nodes and their connections, leading to rapidly increasing diagnosis latency. In edge–cloud environments with hundreds or thousands of distributed services, this quadratic complexity becomes a critical bottleneck, making centralized inference computationally expensive and impractical for real-time diagnosis.

To overcome this challenge, we propose a cascaded GNN architecture designed for joint RCL and RCA, specifically targeting large-scale edge deployments. Our method leverages communication-driven clustering, grouping microservices into clusters based on their interaction intensity. Such clustering serves two purposes: (i) it reduces graph size for the initial inference stage, and (ii) it aligns with real-world anomaly propagation patterns, as performance anomalies typically spread along communication paths. As a result, when an anomaly occurs, its effects are more likely to be confined within the boundaries of the identified cluster, allowing our architecture to diagnose root causes more efficiently without compromising accuracy.

In the following, we first describe our communication-driven clustering strategy for partitioning large service graphs into highly interacting communities (Section 6.4.1). We then present the overall cascaded GNN architecture, including the design of the proposal network and output network (Section 6.4.2), followed by details of their joint optimization. Finally, we analyze the computational efficiency of the proposed framework and compare it with the centralized baseline (Section 6.4.3).

6.4.1 Communication-driven clustering

For cluster formation, we adopt the Louvain community detection algorithm [159], a modularity-optimizing method well suited for large-scale networks. Algorithm 7 presents our communication-driven clustering algorithm. Given the system topology graph $G = (V, L)$, where $\mathcal{V} = \mathcal{M} \cup \mathcal{E}$ consists of microservices and edge devices, and $\mathcal{L} = \mathcal{L}_{\text{comm}} \cup \mathcal{L}_{\text{dep}}$ contains both communication and deployment relationships, we derive a communication-only subgraph for clustering. Specifically, we construct

$$\mathcal{G}_{\text{comm}} = (\mathcal{M}, \mathcal{L}_{\text{comm}}),$$

i.e., we extract a subgraph containing only microservice nodes and their communication edges, since anomaly propagation primarily follows runtime call paths. Louvain clustering is applied to $\mathcal{G}_{\text{comm}}$ to obtain a node-to-cluster assignment:

$$\phi : \mathcal{M} \rightarrow \{c_1, c_2, \ldots c_{|\mathcal{C}|}\},$$

Algorithm 7: Communication-driven clustering algorithm

```
1: Input: System graph $\mathcal{G} = (\mathcal{V}, \mathcal{L})$, deployment mapping $\delta : \mathcal{M} \to \mathcal{E}$
2: Output: Cluster assignments $\phi(m)$ for microservices and $\psi(e)$ for edge devices
3: Construct communication subgraph $\mathcal{G}_{\text{comm}} = (\mathcal{M}, \mathcal{L}_{\text{comm}})$
4: Apply Louvain modularity optimization to obtain microservice communities
   $\{c_1, c_2, \dots, c_{|\mathcal{C}|}\}$
5: for each community $c_i$ do
6:     for each microservice $m$ in $c_i$ do
7:         Assign $\phi(m) = c_i$
8:     end for
9: end for
10: /* Edge-device cluster assignment */
11: for each edge device $e \in \mathcal{E}$ do
12:     Let $\mathcal{M}_e = \{m \in \mathcal{M} \mid \delta(m) = e\}$ {microservices deployed on $e$}
13:     Determine dominant microservice cluster
            $c^* = \arg\max_{c_i} |\{m \in \mathcal{M}_e \mid \phi(m) = c_i\}|$
14:     Assign $\psi(e) = c^*$
15: end for
16: Return: $\phi(m)$ for all $m \in \mathcal{M}$, $\psi(e)$ for all $e \in \mathcal{E}$
```

where $|\mathcal{C}|$ is the number of detected clusters.

To extend clustering to devices, we assign each edge device to the cluster that contains the majority of its deployed microservices. Formally, for a device e, let $\mathcal{M}_e = \{m \in \mathcal{M} \mid \delta(m) = e\}$,

We select the dominant community

$$\psi(e) = \arg\max_{c_i} |\{\, m \in \mathcal{M}_e \mid \phi(m) = c_i \,\}| \,.$$

resulting in a consistent cluster structure across both microservices and edge infrastructure. These communication-driven clusters serve as the foundation of our cascaded GNN pipeline.

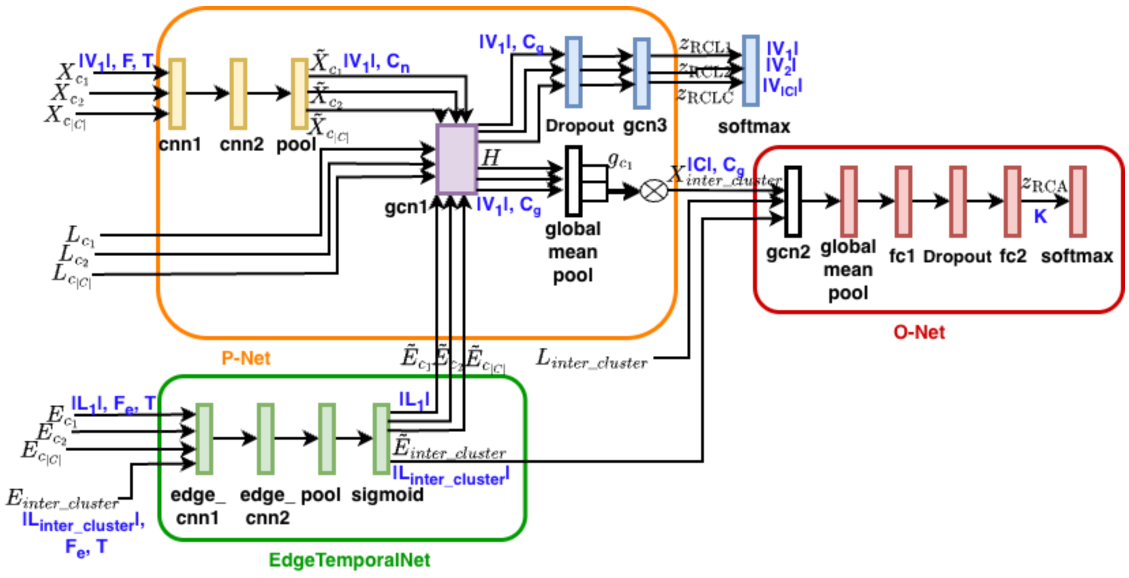


Figure 6.2: Cascaded GNN architecture for joint RCL–RCA

6.4.2 Overview of cascaded joint RCL/RCA GNN architecture

As shown in Figure 6.2, the cascaded GNN is composed of two main components: the proposal network (P-Net) and the output network (O-Net). We divided the baseline GNN into two networks: P-Net, which operates on individual clusters and generates cluster-level outputs, and O-Net, which considers each cluster as a node and operates over the graph of clusters and produces graph-level outputs by considering inter-cluster connectivity. In addition, the EdgeTemporalNet module, which consists of the edge feature encoder component of the baseline GNN architecture, is represented as an independent component since it is shared by both P-Net and O-Net for temporal edge feature embedding. The primary motivation behind this cascading design is to reduce the problem search space handled by each network, thereby lowering the computational burden on the GCN layers within those networks. This strategy aims to decrease overall diagnosis latency while still maintaining a high level of accuracy.

Proposal Network (P-Net)

To reduce the search space and improve computational efficiency, P-Net operates on individual clusters. Each cluster represents a highly communicating group of microservices as explained in Subsection 6.4.1. The goal of P-Net is to learn intra-cluster dependencies and produce cluster-level outputs. It produces two such cluster-level outputs. The first is a cluster-level embedding that carries forward any necessary intra-cluster information for RCA. Embeddings from multiple such clusters are used as inputs in the subsequent O-Net. The second output is a vector of logit values corresponding to the nodes within the cluster. Since RCL is framed as a node-level ranking problem and most anomaly propagations are confined to clusters (i.e., all the information required for RCL is typically available within a cluster), it is sufficient to perform RCL at the cluster level. Hence, P-Net is assigned the task of performing RCL.

The inputs to P-Net are derived from the full graph attributes X,$\mathcal{L}$, and E, but restricted per cluster. Let $\{c_1, c_2, \dots c_{|C|}\}$ denote the set of clusters obtained during the clustering stage. For each cluster c_i in this set, we construct a corresponding subgraph with node features X_{c_i}, edge index $\mathcal{L}_{c_i}$, and edge features E_{c_i}, containing only the nodes and intra-cluster edges belonging to c_i. These cluster-specific inputs are then processed independently by P-Net. As illustrated in Figure 6.2, P-Net reuses the complete node feature encoder from the Temporal Feature Extraction stage of the baseline GNN. It also shares the EdgeTemporalNet-based edge feature encoder with O-Net as described earlier. The spatial graph learning component has a single GCN layer. The RCL prediction branch of the baseline model is retained up to the GCN layer, after which the cluster-level outputs are aggregated, and a global softmax is applied over all nodes (across all clusters) to produce the final RCL scores. Alongside node-level outputs for RCL, P-Net also produces the previously explained cluster-level embedding by applying global mean pooling to the GCN outputs (at its Spatial Graph Learning stage) within each cluster.

Output Network (O-Net)

O-Net considers each cluster as a node and the connectivity between such clusters as edges and produces a graph-level output. The output embeddings of P-Net form the node features, while edge features are formed by aggregating the information from inter-cluster edges. Since RCA is formulated as a graph-level classification problem, O-Net is assigned the responsibility for generating the final RCA prediction for the overall system.

For O-Net, the node features $X_{inter\_cluster}$ are the embeddings from P-Net, while the edge index $\mathcal{L}_{inter\_cluster}$ is constructed from inter-cluster edges (i.e., edges connecting nodes from different clusters), with indices now mapped to cluster IDs rather than node IDs. The corresponding edge features $E_{inter\_cluster}$ are first processed by the EdgeTemporalNet to extract compact time-series representations. When multiple edges exist between a pair of clusters, their encoded features are aggregated using global mean pooling to form a single inter-cluster edge representation, which is then passed to O-Net. As shown in Figure 6.2, O-Net consists of the architecture branch of the baseline model leading to the RCA head, and outputs class probabilities corresponding to fault types.

Joint Task Optimization

Similar to the baseline architecture, the RCL and RCA outputs of the cascaded model are also optimized jointly using the same multi-task loss function mentioned in Section 6.3.4, enabling both objectives to be learned simultaneously while allowing shared representations to benefit both tasks.

6.4.3 Computational efficiency analysis

To assess how effectively the cascaded design meets its objective, we can analyze its computational behavior relative to the baseline GNN. Although the proposed model also utilizes a total of three GCN layers (two in P-Net and one in O-Net, equivalent to the baseline), the key efficiency gain arises from where each layer operates. Instead of processing all N nodes in a single graph, P-Net operates within each cluster. Let the set of nodes be partitioned into clusters of sizes $n_1, n_2, \ldots, n_k$ such that $N = \sum_{i=1}^{k} n_i$. For a typical GCN layer with message passing complexity $O(N^2)$, processing clusters independently yields a total complexity proportional to $N = \sum_{i=1}^{k} n_i$ [14, 24]. Since

$$N^2 = \left(\sum_{i=1}^{k} n_i\right)^2 = \sum_{i=1}^{k} n_i^2 + 2\sum_{i \neq j} n_i n_j.$$

and all $n_i > 0$, it follows that:

$$N^2 > \sum_{i=1}^{k} n_i^2,$$

meaning the work performed by P-Net is strictly lower than processing the entire graph at once. Moreover, clusters can be processed independently and thus parallelized, further lowering inference latency. O-Net then operates on a compact graph with $|\mathcal{C}| \ll N$ nodes, leading to an additional drop in computation during the last GCN layer.

Despite the compression introduced by clustering, accuracy is preserved because most anomaly propagation remains localized within clusters (thereby allowing P-Net to operate with minimal context loss) and O-Net recovers cross-cluster information through aggregated inter-cluster edges. Thus, the cascaded architecture is capable of achieving the desired goal: reduced computational cost and faster diagnosis while retaining accuracy, as validated in the next section.

6.5 Performance Evaluation

In this section, we evaluate the proposed cascaded joint RCL/RCA GNN architecture in terms of both diagnostic effectiveness and scalability. We first describe the experimental setup, datasets, and implementation details for both MicroCERCL and iAnomaly (Section 6.5.1). We then report accuracy and latency results on MicroCERCL, comparing the cascaded model against the centralized baseline under a medium-scale setting (Section 6.5.2). Next, we analyze scalability on iAnomaly by varying graph size up to 10,000 nodes and measuring end-to-end inference time under different clustering configurations (Section 6.5.3). Finally, we conduct an ablation study to quantify the impact of joint learning, clustering strategy, and GNN layer type on performance (Section 6.5.4).

6.5.1 Experimental setup and implementation details

We evaluated the proposed cascaded GNN approach using the publicly available MicroCERCL dataset[1] [52]. This dataset represents the first large-scale benchmark for cloud–edge collaborative microservice systems and remains the most comprehensive hybrid deployment dataset available to date. It contains data collected from 81 microservices belonging to four applications: SockShop, Hipster, Bookinfo, and the newly introduced AI-Edge. These microservices are deployed across four cloud servers and two groups of two edge servers, following a communication frequency-based application placement strategy.

To reflect realistic production environments, the dataset integrates a wide range of anomalies. Using ChaosMesh, the authors injected application-level anomalies, such as CPU resource exhaustion in containers, memory leaks, and network latency. Additionally, Linux kernel traffic control (TC) was employed to simulate kernel-level network failures—including packet loss, duplication, corruption, disorder, delay, and jitter—between cloud and edge nodes. These injections enable systematic evaluation under heterogeneous and complex failure conditions.

The dataset contains 682 fault scenarios, providing substantial diversity in anomaly types and intensities. For our experiments, we adopted a training, validation, and testing split of 60:20:20. For each scenario, distributed trace data were extracted to construct service dependency graphs, while corresponding system and application metrics were used for anomaly detection and as node and edge features for the GNN models.

Although MicroCERCL represents a large-scale benchmark in terms of deployment realism and anomaly diversity, it can be regarded as a medium-scale dataset for efficiency-oriented evaluations. In particular, its graph sizes are insufficient to fully characterize the scalability behavior of RCL and RCA models under increasing system complexity. To evaluate the computational efficiency of the proposed approach under larger input sizes, we therefore employed a dataset generation tool based on the iAnomaly framework (introduced through chapter 3). It enables the generation of service dependency graphs ranging from 50 to 10,000 nodes, thereby supporting systematic scalability analysis. Unlike purely synthetic simulators that rely on artificial workloads, the iAnomaly

[1]https://github.com/WDCloudEdge/MicroCERCL

generator populates these graphs with anomalous traces derived from real execution data produced by the iAnomaly emulator. This ensures that the generated datasets retain realistic temporal and causal characteristics.

The underlying data originate from a diverse set of IoT applications, spanning lightweight sensor processing pipelines to computation-intensive services such as camera-based face detection and recognition. These applications are executed under a wide range of injected anomalies, including resource saturation, service failures, and network congestion. Consequently, the resulting datasets provide a realistic and scalable testbed for evaluating both the accuracy and efficiency of the proposed cascaded GNN framework.

Anomaly detection was performed on these datasets using the BIRCH clustering-based algorithm from Scikit-learn, following the configuration recommended by the MicroCERCL authors. Specifically, the anomaly sensitivity threshold β was set to 0.07, balancing anomaly detection accuracy and noise reduction.

Both the centralized and cascaded GNN models were implemented in Python 3.10 using PyTorch Geometric 2.6.1[2]. Model hyperparameters were optimized using the Tree-structured Parzen Estimator (TPE)-based Bayesian optimization method [153]. The proposed communication-driven clustering algorithm (Algorithm 7) was implemented in Python. All experiments, including hyperparameter tuning, were conducted on the Spartan HPC cluster[3]. Model training was performed using the Adam optimizer [160].

6.5.2 MicroCERCL evaluation results

Before comparing our proposed cascaded GNN approach against the centralized baseline GNN model, we first validate the accuracy of the centralized GNN baseline by benchmarking it against the results originally reported in the MicroCERCL paper. Notably, the MicroCERCL authors report only RCL performance for two applications (HH and SH) measured using top-K localization accuracy. Their results are shown in Table 6.2.

To verify that our centralized baseline is competitive, we first train it to perform RCL only. As shown in Table 6.3, our model achieves significantly higher top-K localization

[2]https://pytorch-geometric.readthedocs.io/en/latest/
[3]https://dashboard.hpc.unimelb.edu.au/

Table 6.2: RCL accuracy reported in the MicroCERCL paper

Application	Acc@1	Acc@2	Acc@3	Acc@5	Acc@10
HH	0.632	0.756	0.796	0.833	0.896
SH	0.607	0.732	0.792	0.849	0.907

accuracy compared to the results reported in the MicroCERCL paper. These results confirm that our centralized baseline is competitive and suitable for a fair comparison with the proposed cascaded approach.

Table 6.3: RCL-only performance of our centralized GNN baseline

Method	Acc@1	Acc@3	Acc@5	MAR	MRR
Centralized (RCL only)	0.9203	0.9783	0.9928	1.1594	0.9534

We also evaluate the centralized baseline on RCA as an independent task, formulated as a fault type classification problem. The results are shown in Table 6.4.

Table 6.4: RCA-only performance of our centralized GNN baseline

Method	Accuracy	Precision	Recall	F1-score
Centralized (RCA only)	0.8478	0.8667	0.8478	0.8512

We then train the baseline to jointly perform RCL and RCA using a shared representation and a multi-task objective. The joint learning results are summarized in Table 6.5. Compared to training separate models for RCL and RCA, joint optimization has increased in both localization performance and fault classification accuracy. This indicates that shared representations allow complementary information to be exploited across tasks, leading to more effective end-to-end root cause diagnosis.

We next evaluate our proposed cascaded GNN architecture for joint RCL and RCA and compare it against the centralized joint model. The results are summarized in Table 6.6. Overall, the cascaded approach achieves localization and diagnosis performance that is comparable to the centralized baseline. This demonstrates that decomposing the global graph into communication-driven clusters and processing them via a cascaded network does not degrade diagnostic accuracy.

To assess inference efficiency, we measure the end-to-end diagnosis time for both joint models. For the centralized joint GNN, the average and median diagnosis times are 17.779 ms and 17.707 ms, respectively. In comparison, the cascaded joint GNN incurs

Table 6.5: Joint RCL and RCA performance of our centralized baseline

	RCL					**RCA**			
Method	**Acc@1**	**Acc@3**	**Acc@5**	**MAR**	**MRR**	**Accuracy**	**Precision**	**Recall**	**F1-score**
Joint RCL & RCA	0.9275	0.9855	1.0000	1.1377	0.9550	0.8696	0.8877	0.8696	0.8715

Table 6.6: Joint RCL and RCA performance of the cascaded GNN

	RCL					**RCA**			
Method	**Acc@1**	**Acc@3**	**Acc@5**	**MAR**	**MRR**	**Accuracy**	**Precision**	**Recall**	**F1-score**
Cascaded RCL & RCA	0.9130	0.9420	0.9493	1.9058	0.9316	0.8623	0.8703	0.8623	0.8640

an average diagnosis time of 20.237 ms and a median time of 20.273 ms. These results indicate that, under the scale of the MicroCERCL dataset (approximately 50 nodes), the cascaded architecture does not yet exhibit clear latency improvements over the centralized baseline.

This observation is expected, as the MicroCERCL dataset is a medium-scale dataset in terms of efficiency evaluations and does not sufficiently stress the scalability limits of centralized graph processing. In such settings, the overhead introduced by clustering and multi-stage inference can offset potential computational savings. Consequently, while the cascaded GNN maintains comparable diagnostic accuracy, its efficiency benefits are not fully realized at this scale.

To further examine scalability, we conduct additional experiments on the iAnomaly dataset, which contains significantly larger service graphs. These experiments are designed to stress-test the centralized model and assess how the cascaded architecture behaves as graph size increases. We discuss these results next.

6.5.3 Scalability analysis on the iAnomaly dataset

Using the iAnomaly dataset, we vary the service graph size from 50 to 10,000 nodes to enable a systematic comparison between the centralized joint GNN and the cascaded joint GNN under increasing graph sizes.

We measure the average end-to-end diagnosis time for three configurations: (i) the centralized joint RCL/RCA model, (ii) the cascaded joint model with a fixed number of clusters (set to 10), and (iii) the cascaded joint model with an adaptive number of

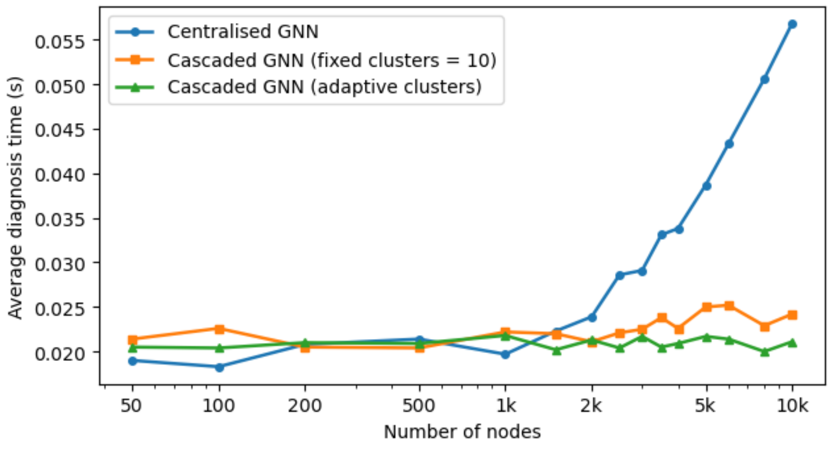


Figure 6.3: Scalability comparison on the iAnomaly dataset

clusters, where the number of clusters is selected proportionally to the graph size (approximately one cluster per 20 nodes).

Figure 6.3 summarizes the results. As the number of nodes increases, the centralized model exhibits a clear upward trend in diagnosis time, indicating limited scalability due to global message passing over an increasingly large graph. The cascaded model with a fixed number of clusters reduces this growth but still shows a mild increase in latency as cluster sizes grow with graph scale. In contrast, when the number of clusters is adaptively controlled based on the graph size, the cascaded GNN maintains an approximately constant diagnosis time across all evaluated scales.

These results confirm that the proposed cascaded GNN architecture is capable of preserving diagnostic accuracy while enabling scale-invariant inference, making it well suited for large-scale microservice deployments in edge and cloud environments.

6.5.4 Ablation study

To analyze the contribution of individual design choices in the proposed cascaded GNN architecture, we conduct a series of ablation experiments focusing on joint learning, clustering strategy, and GNN layer type.

Table 6.7 presents the results of the ablation study. The first row corresponds to the proposed full cascaded model with joint learning, communication-driven clustering,

Table 6.7: Ablation study on the proposed cascaded GNN architecture

Category	Variant	RCL					RCA			
		Acc@1	**Acc@3**	**Acc@5**	**MAR**	**MRR**	**Acc.**	**Prec.**	**Rec.**	**F1**
Proposed (Full Model)		**0.9130**	**0.9420**	**0.9493**	**1.9058**	**0.9316**	**0.8623**	**0.8703**	**0.8623**	**0.8640**
Joint Learning	RCL only	0.8696	0.9420	0.9638	1.8116	0.9123	–	–	–	–
	RCA only	–	–	–	–	–	0.6739	0.6760	0.6739	0.6692
Clustering	Random	0.7971	0.8623	0.8841	2.6884	0.8386	0.8478	0.8667	0.8478	0.8512
GNN Layer	GAT	0.8478	0.9493	0.9855	1.3333	0.9067	0.8333	0.8422	0.8333	0.8346

and GCN layers.

Impact of Joint Learning. Removing joint learning and training task-specific models leads to consistent performance degradation. For RCL, Acc@1 decreases from 0.9130 to 0.8696. For RCA, the impact is more pronounced: the F1-score drops sharply from 0.8640 to 0.6692. This substantial reduction indicates that shared representations across localization and diagnosis tasks improve feature generalization and enhance discriminative capability.

Impact of Clustering Strategy. Replacing communication-driven clustering with random clustering significantly degrades RCL performance. Acc@1 drops from 0.9130 to 0.7971, and MAR increases from 1.9058 to 2.6884, indicating poorer ranking quality. Although RCA F1-score decreases more modestly (from 0.8640 to 0.8512), the degradation across both tasks confirms that preserving communication dependencies during graph decomposition is critical for maintaining localization accuracy.

Impact of GNN Layer Type. Substituting GCN layers with GAT layers yields mixed results. While Acc@5 slightly increases (0.9493 to 0.9855) and MAR improves (1.9058 to 1.3333), Acc@1 decreases (0.9130 to 0.8478) and RCA F1-score drops from 0.8640 to 0.8346. Given that GAT introduces additional attention computation overhead, these results suggest that the added complexity does not translate into consistent performance gains. Overall, GCN layers offer a better balance between effectiveness and computational efficiency in the cascaded architecture.

6.6 Summary

This chapter addressed the scalability limitations of existing GNN-based joint RCL and analysis methods in edge–cloud environments. We introduced a cascaded GNN architecture that decomposes the diagnosis task through communication-driven clustering and hierarchical graph reasoning. Our framework consists of a proposal network that operates on individual communication-aware clusters to generate cluster-level outputs, followed by an output network which considers each cluster as a node and operates over the graph of clusters to produce graph-level outputs by considering inter-cluster connectivity. Extensive experimental results demonstrated that the cascaded approach preserves diagnostic accuracy while maintaining near-constant inference latency as system size grows. This confirms the effectiveness of hierarchical graph learning for scalable anomaly diagnosis in large-scale edge systems.

Chapter 7
Conclusions and Future Directions

This chapter serves as the conclusion of the thesis, providing a comprehensive summary of the primary works and contributions presented. Additionally, it outlines essential future directions for the continued advancement of anomaly-aware management of microservice-based edge–cloud systems.

7.1 Summary of Contributions

The rapid growth of microservice-based IoT applications and the increasing adoption of edge computing have fundamentally transformed how distributed systems are designed and operated. By enabling computation closer to data sources, edge computing offers significant benefits in terms of latency reduction, bandwidth efficiency, and QoS. However, the heterogeneous, resource-constrained, and highly dynamic nature of edge environments introduces new challenges for performance monitoring and management. In particular, performance anomalies in microservice-based edge systems are difficult to detect and diagnose due to complex service dependencies, anomaly propagation, and the limitations of centralized and resource-intensive analysis. This thesis addresses these challenges by developing scalable and efficient techniques for performance anomaly detection and diagnosis stages of the anomaly management lifecycle in edge–cloud integrated environments.

Chapter 1 introduced the problem of performance anomaly detection and diagnosis in microservice-based IoT applications deployed in edge computing environments. The chapter outlined the key characteristics of edge–cloud systems, discussed the limitations of existing cloud-centric approaches, and formulated the research challenges arising from data scarcity, system heterogeneity, and scalability constraints. It then defined the research objectives and questions that guide the thesis and summarised the core contributions and structure of the work.

Chapter 2 established the research foundation through a comprehensive systematic review and taxonomy of performance anomaly detection and diagnosis techniques for edge–cloud systems. The chapter surveyed state-of-the-art approaches across anomaly detection, root cause localization (RCL), and root cause analysis (RCA), and classified existing methods based on diagnostic scope, observability sources, learning paradigms, modeling approaches, and system architectures, while also considering scalability constraints in edge environments. By critically analyzing current limitations and identifying open research problems, this chapter motivated the need for scalable, decentralized, and edge-aware anomaly management solutions.

Chapter 3 addressed the lack of realistic and reproducible evaluation data by introducing *iAnomaly*, a full-system emulation and automated dataset generation framework for edge computing environments. The chapter presented a comprehensive performance anomaly dataset derived from real executions of microservice-based IoT applications under diverse failure scenarios. In addition, a trace-driven dataset generation simulator was developed to support scalability experiments on large dependency graphs. Together, these contributions enable systematic and reproducible evaluation of anomaly detection and diagnosis techniques under realistic and large-scale conditions.

Chapter 4 focused on improving the scalability of anomaly detection model training in heterogeneous edge infrastructures. The chapter proposed two clustering-based training strategies: intra-cluster parameter transfer learning (ICPTL) and cluster-level model training (CM), that reduce training overhead while preserving detection accuracy. Extensive experimental evaluations demonstrated that these approaches effectively balance computational efficiency and diagnostic performance, addressing the challenges posed by resource constraints and workload diversity in edge environments.

Chapter 5 investigated the limitations of centralized RCL in distributed edge systems and introduced a decentralized localization framework based on communication- and colocation-aware clustering and Personalized PageRank (PPR). By performing diagnosis directly at the edge-device level, the proposed approach significantly reduces communication overhead and localization latency. The chapter further introduced an edge-specific anomaly scoring mechanism and a lightweight P2P approximation strategy to support multi-cluster anomaly propagation, demonstrating robust and scalable localization across varying system scales.

Chapter 6 addressed the scalability challenges of state-of-the-art supervised GNN–based diagnosis techniques. The chapter proposed a cascaded GNN architecture for joint RCL and fault type analysis, combining communication-driven clustering with hierarchical inference. By decomposing global dependency graphs into structured subgraphs and enabling multi-stage inference, the proposed approach achieved near-constant inference latency at scale while maintaining diagnostic accuracy comparable to centralized GNN baselines.

Together, these contributions form a cohesive and scalable framework for performance anomaly detection and diagnosis in microservice-based edge computing environments. By addressing challenges spanning data availability, learning efficiency, decentralized diagnosis, and scalable joint analysis, this thesis advances the state of the art in edge-aware AIOps and provides a solid foundation for future research toward autonomous, resilient, and intelligent edge–cloud systems.

7.2 Future Research Directions

Although research on anomaly detection and diagnosis for microservice-based edge–cloud systems is still in its early stages, the increasing adoption of edge computing and large-scale IoT deployments indicates that this area will become increasingly important in the coming years. The complexity, heterogeneity, and dynamic nature of edge environments introduce new challenges that are not fully addressed by existing cloud-centric approaches. While the techniques developed in this thesis contribute toward scalable and efficient anomaly detection and diagnosis in such systems, several open research challenges remain. The following subsections outline promising directions for future work that could further advance anomaly-aware management of microservice-based edge–cloud environments.

7.2.1 Advanced Evaluation Environments and Benchmarking

Progress in performance anomaly detection and diagnosis research is often constrained by the limited availability of realistic datasets and large-scale evaluation environments for edge–cloud systems. Although the iAnomaly framework introduced in this thesis provides a reproducible platform for generating performance anomaly datasets from microservice-based IoT applications, further extensions could significantly enhance its capabilities.

For example, the current framework primarily collects system and application metrics. Extending the toolkit to capture distributed tracing data alongside metrics would enable the creation of richer multi-source datasets suitable for evaluating advanced RCL and RCA techniques. Since Pixie is already integrated within the iAnomaly observability stack, incorporating trace collection can be achieved with relatively minor extensions to the existing framework.

Another promising direction is expanding the dataset generation capabilities using data-driven synthetic data generation techniques. Approaches such as Generative Adversarial Networks (GANs) could be used to augment collected datasets with additional anomaly patterns, enabling the training and evaluation of learning-based diagnosis models under a wider variety of scenarios.

Beyond dataset generation, the framework could also be enhanced to support real-time experimentation on anomaly-aware resource management strategies. By enabling controlled deployment, monitoring, anomaly injection, and mitigation within the same experimental environment, iAnomaly could serve as a practical platform for evaluating closed-loop anomaly management techniques that dynamically adapt resource allocation or service configurations in response to detected anomalies.

Finally, future research could focus on developing scalable simulation environments capable of emulating heterogeneous edge infrastructures and large microservice dependency graphs. High-fidelity simulators and digital testbeds would allow researchers to systematically evaluate the scalability, efficiency, and robustness of anomaly management techniques under controlled yet realistic conditions.

7.2.2 Multi-Modal Observability and Cross-Layer Diagnosis

Effective diagnosis of performance anomalies in microservice-based systems requires comprehensive observability across multiple telemetry sources. In practice, modern distributed systems generate diverse monitoring data including metrics, logs, and traces. However, many existing anomaly detection techniques for edge environments primarily rely on metric-based signals, while logs and traces are often analyzed independently or ignored due to resource constraints.

Future research should therefore explore efficient techniques for jointly analyzing multi-modal observability data within edge–cloud environments. Integrating metrics, logs, and distributed traces can provide complementary perspectives on system behavior, enabling more accurate detection of anomalous patterns and improved root cause identification. For example, traces capture service-level execution paths and latency propagation patterns, while logs often contain contextual information about failures or abnormal system states.

Developing lightweight multi-modal data fusion techniques that can operate under the computational and memory limitations of edge nodes represents an important research challenge. Such approaches could leverage hierarchical processing strategies in which initial anomaly detection is performed locally at the edge using lightweight models, while more complex multi-source analysis is performed collaboratively across edge clusters or in the cloud. Advancing multi-modal observability analysis will therefore be essential for enabling more accurate and robust anomaly diagnosis in distributed edge systems.

7.2.3 Learning Paradigms for Data-Scarce Edge Environments

Many learning-based anomaly detection and diagnosis techniques rely on large labelled datasets for training. However, obtaining labelled anomaly data in real-world edge environments is often difficult due to the rarity of failures, the diversity of application workloads, and the operational complexity of distributed deployments. Consequently, future research should explore learning paradigms that can operate effectively under limited labelled data.

In particular, semi-supervised, self-supervised, and weakly supervised learning approaches offer promising directions for improving anomaly detection and diagnosis under such conditions. While these paradigms have gained increasing attention in cloud-based anomaly detection research, their adoption in edge environments remains relatively limited. Developing lightweight variants of these methods that can operate efficiently on resource-constrained edge nodes represents an important opportunity for advancing edge-aware AIOps.

In addition, recent advances in transformer-based models and large foundation models have demonstrated strong capabilities for modeling complex temporal patterns and system dependencies. Exploring practical strategies for adapting such models to edge environments, for example through model distillation, hierarchical inference, or edge–cloud collaborative architectures, could further enhance diagnostic performance.

7.2.4 Efficiency-Aware Diagnostic Architectures for Edge Systems

Edge computing environments impose strict constraints on computational resources, memory capacity, and network bandwidth. Consequently, anomaly detection and diagnosis techniques designed for cloud environments cannot always be directly applied to edge systems without modification. While this thesis addressed several scalability challenges through clustering-based learning strategies and hierarchical diagnostic architectures, further research is needed to develop diagnostic algorithms explicitly designed for edge environments.

One important direction involves designing resource-adaptive models capable of dynamically adjusting their computational complexity based on available system resources or data characteristics. Adaptive methods that adjust model parameters, feature representations, or inference strategies based on runtime feedback could improve both efficiency and robustness in dynamic environments. Similarly, lightweight diagnostic architectures capable of operating with limited compute and memory resources remain an important area of investigation.

Another promising direction is the integration of complementary efficiency mechanisms into unified diagnostic frameworks. Techniques such as dimensionality reduction, lightweight model architectures, distributed inference, graph pruning, and hierarchical reasoning are often studied independently. Developing approaches that systematically combine these mechanisms could significantly improve scalability and efficiency while maintaining acceptable diagnostic accuracy. Achieving an effective balance between diagnostic effectiveness and computational efficiency will therefore remain a central challenge in the design of next-generation edge-aware anomaly management systems.

7.2.5 Adaptation to Dynamic Edge Deployment and Placement Strategies

The techniques proposed in this thesis exploit specific microservice placement characteristics to improve the scalability and efficiency of anomaly detection and diagnosis in edge environments. However, real-world edge–cloud systems may adopt a variety of service placement strategies depending on operational objectives such as latency minimization, QoS guarantees, resource utilization, or adaptive workload balancing. Consequently, an important direction for future work is to investigate how the proposed approaches can be generalized to operate effectively under diverse and dynamically evolving placement policies.

For anomaly detection, the clustering-based model training strategies proposed in Chapter 4 assume a QoS-aware placement of microservices when forming service groups for efficient model training. Future research could explore how these training strategies can be adapted to alternative placement strategies, including purely resource-driven, latency-aware, or dynamically adaptive scheduling policies, while preserving the efficiency benefits of clustered training.

Similarly, the decentralized RCL approach introduced in Chapter 5 and the cascaded GNN-based joint RCL/RCA framework presented in Chapter 6 rely on communication- and colocation-aware clustering to improve diagnostic scalability. In practice, however, microservice deployments may not always follow such placement assumptions. Extending these approaches to support a wider range of placement strategies, such as QoS-driven or dynamically adaptive orchestration, while maintaining diagnostic accuracy and efficiency represents an important research opportunity. In addition, future work could investigate integrating communication-aware clustering with federated learning or hierarchical inference mechanisms to enable scalable deployment of learning-based diagnostic models directly at edge nodes.

7.2.6 Towards End-to-End Autonomous Anomaly Management

The contributions of this thesis focus primarily on the detection and diagnosis stages of the anomaly management lifecycle. While accurate anomaly detection and root cause localization are essential prerequisites, the ultimate goal of anomaly-aware management systems is to enable automated mitigation and recovery. Future research can therefore extend the techniques developed in this thesis toward end-to-end autonomous anomaly management frameworks capable of detecting anomalies, diagnosing their causes, and triggering appropriate mitigation actions without human intervention.

One promising direction involves integrating diagnosis outputs with automated mitigation policies based on reinforcement learning (RL) or deep reinforcement learning (DRL). Such approaches could learn optimal resource allocation or service reconfiguration strategies that minimize performance degradation under varying anomaly conditions. In addition, recent advances in large language models (LLMs) provide opportunities for developing AIOps assistants that support SREs by interpreting diagnostic results and recommending mitigation actions. Beyond human-in-the-loop support, emerging agentic frameworks could enable direct interaction between diagnosis modules and automated remediation systems.

Another important research avenue is the use of *digital twins* for edge–cloud systems. Digital twins can provide realistic virtual environments that replicate the behavior of distributed deployments, enabling candidate mitigation strategies to be evaluated safely before applying them to real infrastructures. Integrating diagnosis pipelines with digital twin environments could enable proactive experimentation and validation of mitigation actions, thereby improving the reliability and safety of automated anomaly management systems in edge environments.

7.3 Final Remarks

Edge computing has emerged as a key paradigm for supporting latency-sensitive, data-intensive IoT applications by enabling computation closer to data sources. At the same time, the adoption of microservice architectures has increased the flexibility and scalability of application deployments in edge–cloud environments. However, the heterogeneous, resource-constrained, and highly dynamic nature of these systems introduces significant challenges for performance monitoring and management. In particular, performance anomalies in microservice-based edge systems are difficult to detect and diagnose due to complex service dependencies, anomaly propagation effects, and the limitations of centralized and resource-intensive analysis approaches.

In this thesis, we investigated the fundamental challenges of performance anomaly detection and diagnosis in microservice-based IoT applications deployed across edge–cloud infrastructures. By addressing the diagnosis phases of the anomaly management lifecycle—from realistic data generation and scalable model training to decentralized RCL and joint diagnosis—this work developed a set of integrated techniques that enable accurate and efficient diagnosis under realistic and large-scale conditions. The dataset generation framework introduced in this thesis provide reproducible and scalable evaluation foundations, while the proposed learning and diagnostic approaches exploit structural and operational characteristics of edge environments to overcome scalability and efficiency limitations.

The decentralized RCL framework and the cascaded GNN architecture proposed in this thesis demonstrate that effective diagnosis in edge environments does not require fully centralized analysis. Instead, locality-aware clustering, hierarchical reasoning, and lightweight coordination mechanisms can significantly reduce communication overhead and inference latency while preserving diagnostic accuracy. Together, these contributions advance the state of the art in edge-aware AIOps and provide practical design principles for building scalable, anomaly-aware management systems in distributed edge–cloud deployments.

Research on performance anomaly detection and diagnosis, such as presented in this thesis, is essential for enabling reliable, autonomous, and resilient operation of future

IoT and edge computing systems. The methods and insights developed in this work can support edge platform providers and application developers in managing increasingly complex microservice-based deployments at scale. Moreover, these research outcomes lay a strong foundation for future advancements toward self-healing, adaptive, and intelligent edge–cloud infrastructures capable of meeting the demands of next-generation IoT applications.

Bibliography

[1] IoT Analytics. (2025) State of iot 2025: Number of connected iot devices growing 14% to 21.1 billion globally. Accessed: 2026-02-06. [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/

[2] F. Golpayegani, N. Chen, N. Afraz, E. Gyamfi, A. Malekjafarian, D. Schäfer, and C. Krupitzer, "Adaptation in edge computing: A review on design principles and research challenges," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 19, pp. 1162–1182, 2024.

[3] S. Pallewatta, V. Kostakos, and R. Buyya, "Placement of microservices-based iot applications in fog computing: A taxonomy and future directions," *ACM Computing Surveys*, vol. 55, p. 43, 2023.

[4] J. Lee and J. Lee, "Hierarchical mobile edge computing architecture based on context awareness," *Applied Sciences*, vol. 8, no. 7, 2018.

[5] Grand View Research. (2025) Edge computing market size, share & trends, 2025–2033. Accessed: 2026-02-06. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/edge-computing-market

[6] D. C. Li, C.-T. Huang, C.-W. Tseng, and L.-D. Chou, "Fuzzy-based microservice resource management platform for edge computing in the internet of things," *Sensors*, vol. 21, no. 11, 2021.

[7] F. Al-Doghman, N. Moustafa, I. Khalil, N. Sohrabi, Z. Tari, and A. Y. Zomaya, "Ai-enabled secure microservices in edge computing: Opportunities and challenges," *IEEE Transactions on Services Computing*, vol. 16, pp. 1485–1504, 2023.

[8] C. Wu, Q. Peng, Y. Xia, Y. Jin, and Z. Hu, "Towards cost-effective and robust ai microservice deployment in edge computing environments," *Future Generation Computer Systems*, vol. 141, pp. 129–142, 2023.

[9] S. Pallewatta, V. Kostakos, and R. Buyya, "Qos-aware placement of microservices-based iot applications in fog computing environments," *Future Generation Computer Systems*, vol. 131, p. 121–136, 2022.

[10] M. Hosseini Shirvani and Y. Ramzanpoor, "Multi-objective qos-aware optimization for deployment of iot applications on cloud and fog computing infrastructure," *Neural Computing and Applications*, vol. 35, p. 19581–19626, 2023.

[11] S. Becker, F. Schmidt, A. Gulenko, A. Acker, and O. Kao, "Towards aiops in edge computing environments," in *Proceedings of the 2020 IEEE International Conference on Big Data (Big Data)*. Atlanta, GA, USA: IEEE, 2020, pp. 3470–3475.

[12] M. Soualhia, C. Fu, and F. Khomh, "Infrastructure fault detection and prediction in edge cloud environments," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, ser. SEC '19. Arlington, Virginia: Association for Computing Machinery, 2019, p. 222–235.

[13] J. Hunter, "Deep learning-based anomaly detection for edge-layer devices," Master's thesis, University of Tennessee at Chattanooga, 2022. [Online]. Available: https://scholar.utc.edu/theses/740/

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

[15] W. Tian, H. Zhang, N. Yang, and Y. Zhang, "Graph-based root cause localization

in microservice systems with protection mechanisms," *International Journal of Software Engineering and Knowledge Engineering*, vol. 33, no. 08, pp. 1211–1238, 2023.

[16] S. Tuli, S. Tuli, G. Casale, and N. R. Jennings, "Generative optimization networks for memory efficient data generation," 2021. [Online]. Available: https://arxiv.org/abs/2110.02912

[17] S. Tuli, F. Mirhakimi, S. Pallewatta, S. Zawad, G. Casale, B. Javadi, F. Yan, R. Buyya, and N. R. Jennings, "Ai augmented edge and fog computing: Trends and challenges," *Journal of Network and Computer Applications*, vol. 216, p. 103648, 2023.

[18] S. Skaperas, G. Koukis, I. A. Kapetanidou, V. Tsaoussidis, and L. Mamatas, "A pragmatical approach to anomaly detection evaluation in edge cloud systems," in *International Workshop on Intelligent Cloud Computing and Networking*, ser. ICCN '24. Vancouver, Canada: IEEE, 2024, pp. 1–6.

[19] M. M. John, H. Holmström Olsson, and J. Bosch, "Ai on the edge: Architectural alternatives," in *Proceedings of the 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. Portoroz, Slovenia: IEEE, 2020, pp. 21–28.

[20] E. Raj, D. Buffoni, M. Westerlund, and K. Ahola, "Edge mlops: An automation framework for aiot applications," in *Proceedings of the 2021 IEEE International Conference on Cloud Engineering (IC2E)*. San Francisco, CA, USA: IEEE, 2021, pp. 191–200.

[21] E. Peltonen and S. Dias, "Linkedge: Open-sourced mlops integration with iot edge," in *Proceedings of the 3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum*, ser. ESAAM '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 67–76.

[22] N. Psaromanolakis, V. Theodorou, D. Laskaratos, I. Kalogeropoulos, M.-E. Vlontzou, E. Zarogianni, and G. Samaras, "Mlops meets edge computing: an edge platform with embedded intelligence towards 6g systems," in *2023 Joint European Con-*

ference on Networks and Communications & 6G Summit, ser. EuCNC/6G Summit. Gothenburg, Sweden: IEEE, 2023, pp. 496–501.

[23] J. Soldani and A. Brogi, “Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey,” *ACM Computing Surveys*, vol. 55, no. 3, 2022.

[24] N. Fu, G. Cheng, Y. Teng, G. Dai, S. Yu, and Z. Chen, “Intelligent root cause localization in microservice systems: A survey and new perspectives,” *ACM Computing Surveys*, vol. 57, no. 12, Jul. 2025.

[25] T. Wang and G. Qi, “A comprehensive survey on root cause analysis in (micro) services: Methodologies, challenges, and trends,” 2024. [Online]. Available: https://arxiv.org/abs/2408.00803

[26] C. Bulla and M. N. Birje, “Improved data-driven root cause analysis in fog computing environment,” *Journal of Reliable Intelligent Environments*, vol. 8, p. 359–377, 2021.

[27] O. Kalinagac, W. Soussi, Y. Anser, C. Gaber, and G. Gür, “Root cause and liability analysis in the microservices architecture for edge iot services,” in *Proceedings of the ICC 2023 - IEEE International Conference on Communications*, 2023, pp. 3277–3283.

[28] Z. Li, N. Zhao, M. Li, X. Lu, L. Wang, D. Chang, X. Nie, L. Cao, W. Zhang, K. Sui, Y. Wang, X. Du, G. Duan, and D. Pei, “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*, ser. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 996–1008.

[29] Y. Zhu, J. Wang, B. Li, Y. Zhao, Z. Zhang, Y. Xiong, and S. Chen, “Microirc: Instance-level root cause localization for microservice systems,” *Journal of Systems and Software*, vol. 216, no. C, Oct. 2024.

[30] S. Zhang, P. Jin, Z. Lin, Y. Sun, B. Zhang, S. Xia, Z. Li, Z. Zhong, M. Ma, W. Jin, D. Zhang, Z. Zhu, and D. Pei, “Robust failure diagnosis of microservice system

through multimodal data," *IEEE Transactions on Services Computing*, vol. 16, no. 6, pp. 3851–3864, 2023.

[31] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128610001568

[32] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[33] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[34] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[35] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 13–16.

[36] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and opportunities in edge computing," in *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, 2016, pp. 20–26.

[37] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[38] K. Dolui and S. K. Datta, "Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing," in *2017 Global Internet of Things Summit (GIoTS)*, 2017, pp. 1–6.

[39] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2021.

[40] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, *Microservices: Yesterday, Today, and Tomorrow*, M. Mazzara and B. Meyer, Eds. Cham: Springer International Publishing, 2017.

[41] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud container technologies: A state-of-the-art review," *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 677–692, 2019.

[42] R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar, and J. Ott, "Consolidate iot edge computing with lightweight virtualization," *IEEE Network*, vol. 32, no. 1, pp. 102–111, 2018.

[43] A. Brogi and S. Forti, "Qos-aware deployment of iot applications through the fog," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1185–1192, 2017.

[44] I. Lera, C. Guerrero, and C. Juiz, "Availability-aware service placement policy in fog computing based on graph partitions," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3641–3651, 2019.

[45] C. Guerrero, I. Lera, and C. Juiz, "A lightweight decentralized service placement policy for performance optimization in fog computing," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, pp. 2435–2452, 2019.

[46] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource provisioning for iot services in the fog," in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, 2016, pp. 32–39.

[47] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic computing: A new paradigm for edge/cloud integration," *IEEE Cloud Computing*, vol. 3, no. 06, pp. 76–83, nov 2016.

[48] L. Ismail and R. Buyya, "Artificial intelligence applications and self-learning 6g networks for smart cities digital ecosystems: Taxonomy, challenges, and future directions," *Sensors*, vol. 22, no. 15, 2022.

[49] M. Tsukada, M. Kondo, and H. Matsutani, “A neural network-based on-device learning anomaly detector for edge devices,” *IEEE Transactions on Computers*, vol. 69, no. 7, pp. 1027–1044, 2020.

[50] D. R. Patrikar and M. R. Parate, “Anomaly detection using edge computing in video surveillance system: review,” *International Journal of Multimedia Information Retrieval*, vol. 11, pp. 85–110, 2022.

[51] S. Samarakoon, S. Bandara, N. Jayasanka, and C. Hettiarachchi, “Self-healing and self-adaptive management for iot-edge computing infrastructure,” in *2023 Moratuwa Engineering Research Conference (MERCon)*, 2023, pp. 473–478.

[52] Y. Zhu, J. Wang, B. Li, X. Tang, H. Li, N. Zhang, and Y. Zhao, “Root cause localization for microservice systems in cloud-edge collaborative environments,” 2024. [Online]. Available: https://arxiv.org/abs/2406.13604

[53] D. Liu, C. He, X. Peng, F. Lin, C. Zhang, S. Gong, Z. Li, J. Ou, and Z. Wu, “Microhecl: high-efficient root cause localization in large-scale microservice systems,” in *Proceedings of the 43rd International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP ’21. IEEE Press, 2021, p. 338–347.

[54] Google Cloud. (2018) Kubernetes best practices: Resource requests and limits. Accessed: 2026-02-22. [Online]. Available: https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-best-practices-resource-requests-and-limits

[55] V. Prasad, M. Bhavsar, and S. Tanwar, “Influence of monitoring: Fog and edge computing,” *Scalable Computing*, vol. 20, pp. 365–376, 2019.

[56] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, “All one needs to know about fog computing and related edge computing paradigms: A complete survey,” *Journal of Systems Architecture*, vol. 98, pp. 289–330, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762118306349

[57] I. Korontanis, A. Makris, and K. Tserpes, "Edgecloud mon: A lightweight monitoring stack for k3s clusters," *SoftwareX*, vol. 26, p. 101675, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711024000463

[58] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag, "Dapper, a large-scale distributed systems tracing infrastructure," Google, Inc., Tech. Rep., 2010. [Online]. Available: http://research.google.com/archive/papers/dapper-2010-1.pdf

[59] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Communications of the ACM*, vol. 59, no. 5, p. 50–57, Apr. 2016.

[60] OpenTelemetry. (2026) Opentelemetry logging. Accessed: 2026-02-23. [Online]. Available: https://opentelemetry.io/docs/specs/otel/logs/

[61] S. Forti, M. Gaglianese, and A. Brogi, "Lightweight self-organising distributed monitoring of fog infrastructures," *Future Generation Computer Systems*, vol. 114, pp. 605–618, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X19334582

[62] 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)*, 2016, pp. 207–218.

[63] A. Souza, N. Cacho, A. Noor, P. P. Jayaraman, A. Romanovsky, and R. Ranjan, "Osmotic monitoring of microservices between the edge and cloud," in *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2018, pp. 758–765.

[64] L. Carnevale, A. Celesti, A. Galletta, S. Dustdar, and M. Villari, "From the cloud to edge and iot: a smart orchestration architecture for enabling osmotic computing," in *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 2018, pp. 419–424.

[65] S. Ilager, J. Fahringer, S. C. d. L. Dias, and I. Brandic, “Demon: Decentralized monitoring for highly volatile edge environments,” in *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*, 2022, pp. 145–150.

[66] S. Ilager, J. Fahringer, A. Tundo, and I. Brandić, “A decentralized and self-adaptive approach for monitoring volatile edge environments,” *ACM Trans. Auton. Adapt. Syst.*, Jun. 2025, just Accepted.

[67] P. Wang, J. Xu, M. Ma, W. Lin, D. Pan, Y. Wang, and P. Chen, “Cloudranger: Root cause identification for cloud native systems,” in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2018, pp. 492–502.

[68] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, “Drain: An online log parsing approach with fixed depth tree,” in *2017 IEEE International Conference on Web Services (ICWS)*, 2017, pp. 33–40.

[69] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J.-G. Lou, M. Chintalapati, F. Shen, and D. Zhang, “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*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 807–817.

[70] Y. Wang, G. Qin, and Y. Liang, “A reliability anomaly detection method based on enhanced gru-autoencoder for vehicular fog computing services,” *Comput. Secur.*, vol. 150, no. C, Mar. 2025.

[71] S. Taherizadeh, A. C. Jones, I. Taylor, Z. Zhao, and V. Stankovski, “Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review,” *Journal of Systems and Software*, vol. 136, pp. 19–38, 2018.

[72] 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*

and Data Mining, ser. KDD'19. Anchorage, AK, USA: Association for Computing Machinery, 2019, p. 2828–2837.

[73] 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*, ser. KDD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 387–395.

[74] T. Su, R. Mo, Y. Gong, and H. Wang, "St-graphrca: A root cause analysis model for spatio-temporal graph propagation in iot edge computing," *Sensors*, vol. 26, no. 5, 2026. [Online]. Available: https://www.mdpi.com/1424-8220/26/5/1474

[75] Z. Heeb, O. Kalinagac, W. Soussi, and G. Gür, "Iomirca: Root cause analysis in iot-extended 5g microservice environments," in *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, ser. SAC '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 106–108.

[76] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 4393–4402. [Online]. Available: https://proceedings.mlr.press/v80/ruff18a.html

[77] S. Tuli, G. Casale, and N. R. Jennings, "Tranad: deep transformer networks for anomaly detection in multivariate time series data," *Proceedings of the VLDB Endowment*, vol. 15, no. 6, p. 1201–1214, Feb. 2022.

[78] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "Do deep neural networks contribute to multivariate time series anomaly detection?" *Pattern Recognition*, vol. 132, 2022.

[79] V. Christodoulou and Y. Bi, "A combination of cusum-ewma for anomaly detection in time series data," in *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2015, pp. 1–8.

[80] G. Box, G. Jenkins, and G. Reinsel, *Time Series Analysis: Forecasting and Control*. Wiley, 2015.

[81] Prometheus. (2026) Alerting rules. Accessed: 2026-02-23. [Online]. Available: https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/

[82] 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*, vol. 13, no. 7, pp. 1443–1471, 2001.

[83] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422.

[84] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, ser. MLSDA'14. New York, NY, USA: Association for Computing Machinery, 2014, p. 4–11.

[85] 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 and Data Mining*, ser. KDD '20. Virtual Event, CA, USA: Association for Computing Machinery, 2020, p. 3395–3404.

[86] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. M. Shroff, "Lstm-based encoder-decoder for multi-sensor anomaly detection," *ArXiv*, vol. abs/1607.00148, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:9286983

[87] X. Jiang, H. Luo, Y. Sun, and S. K. Das, "Circa: A framework for collaborative identification of root cause analysis in iot microservices," *IEEE Transactions on Services Computing*, vol. 19, no. 1, pp. 209–224, 2026.

[88] J. Xu, H. Wu, J. Wang, and M. Long, "Anomaly transformer: Time series anomaly detection with association discrepancy," in *International Conference on Learning*

Representations, 2022. [Online]. Available: https://openreview.net/forum?id=LzQQ89U1qm_

[89] C. Lee, T. Yang, Z. Chen, Y. Su, and M. R. Lyu, "Eadro: An end-to-end troubleshooting framework for microservices on multi-source data," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23. IEEE Press, 2023, p. 1750–1762.

[90] S. Corli, L. Moro, D. Dragoni, M. Dispenza, and E. Prati, "Quantum machine learning algorithms for anomaly detection: A review," *Future Generation Computer Systems*, vol. 166, p. 107632, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X2400596X

[91] R. Frehner and K. Stockinger, "Applying quantum autoencoders for time series anomaly detection," *Quantum Machine Intelligence*, vol. 7, 2025.

[92] 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*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 3009–3017.

[93] Skyline. (2025) What is skyline? Accessed: 2026-03-05. [Online]. Available: https://earthgecko-skyline.readthedocs.io/en/latest/overview.html

[94] Z. Zhuang, Y. Zhang, K. Zhao, C. Guo, B. Yang, Q. Wen, and L. Fan, "Noise matters: Cross contrastive learning for flink anomaly detection," *Proceedings of the VLDB Endowment*, vol. 18, no. 4, p. 1159–1168, Dec. 2024.

[95] R. Das and T. Luo, "Lightesd: Fully-automated and lightweight anomaly detection framework for edge computing," in *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*, 2023, pp. 150–158.

[96] Q. Zhang, R. Han, G. Xin, C. H. Liu, G. Wang, and L. Y. Chen, "Lightweight and accurate dnn-based anomaly detection at edge," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2927–2942, 2022.

[97] L. Chen, Y. Xu, M. Li, B. Hu, H. Guo, and Z. Liu, "Privacy-preserving lightweight time-series anomaly detection for resource-limited industrial iot edge devices," *IEEE Transactions on Industrial Informatics*, vol. 21, no. 6, pp. 4435–4446, 2025.

[98] K. Fu, W. Zhang, Q. Chen, D. Zeng, X. Peng, W. Zheng, and M. Guo, "Qos-aware and resource efficient microservice deployment in cloud-edge continuum," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Virtual event: IEEE, 2021, pp. 932–941.

[99] A. Hrusto, E. Engström, and P. Runeson, "Optimization of anomaly detection in a microservice system through continuous feedback from development," in *Proceedings of the 10th IEEE/ACM International Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems*, ser. SESoS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 13–20.

[100] T. Amarbayasgalan, V. H. Pham, N. Theera-Umpon, and K. H. Ryu, "Unsupervised anomaly detection approach for time-series in multi-domains using deep reconstruction error," *Symmetry*, vol. 12, no. 8, 2020.

[101] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, "Microrca: Root cause localization of performance issues in microservices," in *Proceedings of the NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–9.

[102] Z. Zhang, B. Li, J. Wang, and Y. Liu, "Aamr: Automated anomalous microservice ranking in cloud-native environment," in *Proceedings of the SEKE'21*, 2021, pp. 86–91.

[103] L. Wu, J. Tordsson, J. Bogatinovski, E. Elmroth, and O. Kao, "Microdiag: Fine-grained performance diagnosis for microservice systems," in *Proceedings of the 2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*, 2021, pp. 31–36.

[104] L. Tang, E. Kou, W. Wang, and Q. Chen, "A root cause analysis framework for iot based on dynamic causal graphs assisted by llms," *IEEE Internet of Things Journal*, vol. 12, no. 16, pp. 34 563–34 581, 2025.

[105] R. Xin, P. Chen, and Z. Zhao, "Causalrca: Causal inference based precise fine-grained root cause localization for microservice applications," *Journal of Systems and Software*, vol. 203, no. C, Sep. 2023.

[106] R. Chen, J. Ren, L. Wang, Y. Pu, K. Yang, and W. Wu, "Microegrcl: An edge-attention-based graph neural network approach for root cause localization in microservice systems," in *Service-Oriented Computing: 20th International Conference, ICSOC 2022, Seville, Spain, November 29 – December 2, 2022, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 264–272.

[107] L. Wu, J. Bogatinovski, S. Nedelkoski, J. Tordsson, and O. Kao, "Performance diagnosis in cloud microservices using deep learning," in *Service-Oriented Computing – ICSOC 2020 Workshops: AIOps, CFTIC, STRAPS, AI-PA, AI-IOTS, and Satellite Events, Dubai, United Arab Emirates, December 14–17, 2020, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2020, p. 85–96.

[108] Z. Zhu, C. Lee, X. Tang, and P. He, "Hemirca: Fine-grained root cause analysis for microservices with heterogeneous data sources," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 8, Nov. 2024.

[109] C. Hou, T. Jia, Y. Wu, Y. Li, and J. Han, "Diagnosing performance issues in microservices with heterogeneous data source," in *Proceedings of the 2021 IEEE International Conference on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, 2021, pp. 493–500.

[110] R. Ren, Y. Wang, F. Liu, Z. Li, G. Tyson, T. Miao, and G. Xie, "Grace: Interpretable root cause analysis by graph convolutional network for microservices," in *Proceedings of the 2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*, 2023, pp. 1–4.

[111] Álvaro Brandón, M. Solé, A. Huélamo, D. Solans, M. S. Pérez, and V. Muntés-Mulero, "Graph-based root cause analysis for service-oriented and microservice architectures," *Journal of Systems and Software*, vol. 159, p. 110432, 2020.

[112] 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*, ser. ESEC/FSE 2023, San Francisco, CA, USA, 2023, p. 553–565.

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

[114] Moens, Pieter and Andriessen, Bavo and Sebrechts, Merlijn and Volckaert, Bruno and Van Hoecke, Sofie, "Edge anomaly detection framework for AIOps in Cloud and IoT," in *PROCEEDINGS OF THE 13TH INTERNATIONAL CONFERENCE ON CLOUD COMPUTING AND SERVICES SCIENCE, CLOSER 2023*. SCITEPRESS, 2023, pp. 204–211. [Online]. Available: http://doi.org/10.5220/0011838600003488

[115] S. B. Nath, S. Chattopadhyay, R. Karmakar, S. K. Addya, S. Chakraborty, and S. K. Ghosh, "Containerized deployment of micro-services in fog devices: a reinforcement learning-based approach," *The Journal of Supercomputing*, vol. 78, p. 6817–6845, 2022.

[116] E. Park, K. Baek, E. Cho, and I.-Y. Ko, "Fully decentralized horizontal autoscaling for burst of load in fog computing," *Journal of Web Engineering*, vol. 22, no. 6, pp. 849–870, 2023.

[117] T. K. Authors, "Kubernetes federation v2," https://kubernetes.io/docs/concepts/cluster-administration/, 2023.

[118] V. Colombo, A. Tundo, M. Ciavotta, and L. Mariani, "Towards self-adaptive peer-to-peer monitoring for fog environments," in *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 156–166.

[119] A.-D. Nguyen, "Oreca: Evaluating cadence and elasticity impacts on root

cause analysis in edge-cloud continuum," Master's thesis, Aalto University School of Science, 2025. [Online]. Available: https://urn.fi/URN:NBN:fi:aalto-202601221896

[120] J. Alonso, L. Orue-Echevarria, E. Osaba, J. López Lobo, I. Martinez, J. Diaz de Arcaya, and I. Etxaniz, "Optimization and prediction techniques for self-healing and self-learning applications in a trustworthy cloud continuum," *Information*, vol. 12, no. 8, 2021.

[121] N. Akbari, J. Grundy, A. Cheema, and A. N. Toosi, "Intentcontinuum: Using llms to support intent-based computing across the compute continuum," in *2025 IEEE International Conference on Web Services (ICWS)*, 2025, pp. 573–583.

[122] S. Nedelkoski, J. Bogatinovski, A. K. Mandapati, S. Becker, J. Cardoso, and O. Kao, "Multi-source distributed system data for ai-powered analytics," in *Proceedings of the 8th European Conference on Service-Oriented and Cloud Computing*, ser. ESOCC'20, Crete, Greece, 2020.

[123] C. R. Team, "Gaia: Generic aiops atlas dataset," https://github.com/CloudWise-OpenSource/GAIA-DataSet, 2022, dataset for AIOps research including anomaly detection, log analysis, and fault localization.

[124] ——, "Micross: A microservice simulation system for aiops research," https://github.com/CloudWise-OpenSource/GAIA-DataSet, 2022, microservice simulation environment used to generate the GAIA dataset.

[125] J. Goh, S. Adepu, K. N. Junejo, and A. Mathur, "A dataset to support research in the design of secure water treatment systems," in *Critical Information Infrastructures Security*. Cham: Springer International Publishing, 2017, pp. 88–99.

[126] C. M. Ahmed, V. R. Palleti, and A. P. Mathur, "Wadi: a water distribution testbed for research in the design of secure cyber physical systems," in *Proceedings of the 3rd International Workshop on Cyber-Physical Systems for Smart Water Networks*, ser. CySWATER '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 25–28.

[127] R. Mahmud, S. Pallewatta, M. Goudarzi, and R. Buyya, "ifogsim2: An extended ifogsim simulator for mobility, clustering, and microservice management in edge and fog computing environments," *Journal of Systems and Software*, vol. 190, p. 111351, 2022.

[128] I. Lera, C. Guerrero, and C. Juiz, "Yafs: A simulator for iot scenarios in fog computing," *IEEE Access*, vol. 7, pp. 91 745–91 758, 2019.

[129] A. Coutinho, F. Greve, C. Prazeres, and J. Cardoso, "Fogbed: A rapid-prototyping emulation environment for fog computing," in *2018 IEEE International Conference on Communications (ICC)*, Kansas City, MO, USA, 2018.

[130] R. Mayer, L. Graser, H. Gupta, E. Saurez, and U. Ramachandran, "Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures," in *2017 IEEE Fog World Congress (FWC)*, Santa Clara, CA, USA, 2017.

[131] N. Akbari, A. N. Toosi, J. Grundy, H. Khalajzadeh, M. S. Aslanpour, and S. Ilager, "icontinuum: an emulation toolkit for intent-based computing across the edge-to-cloud continuum," in *2024 IEEE 17th International Conference on Cloud Computing*, ser. CLOUD '24. Shenzhen, China: IEEE, 2024, pp. 468–474.

[132] J. Hasenburg, M. Grambow, E. Grünewald, S. Huk, and D. Bermbach, "Mockfog: Emulating fog computing infrastructure in the cloud," in *2019 IEEE International Conference on Fog Computing (ICFC)*, Prague, Czech Republic, 2019.

[133] Y. Zeng, M. Chao, and R. Stoleru, "Emuedge: A hybrid emulator for reproducible and realistic edge computing experiments," in *2019 IEEE International Conference on Fog Computing (ICFC)*, Prague, Czech Republic, 2019.

[134] M. Symeonides, Z. Georgiou, D. Trihinas, G. Pallis, and M. D. Dikaiakos, "Fogify: A fog computing emulation framework," in *2020 IEEE/ACM Symposium on Edge Computing*, San Jose, CA, USA, 2020.

[135] J. Levin and T. A. Benson, "Viperprobe: Rethinking microservice observability with ebpf," in *Proceedings of the IEEE 9th International Conference on Cloud Networking*, ser. CloudNet'20, 2020.

[136] S. Pallewatta, V. Kostakos, and R. Buyya, “Microservices-based iot application placement within heterogeneous and resource constrained fog computing environments,” in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, ser. UCC’19. Auckland, New Zealand: Association for Computing Machinery, 2019, p. 71–81.

[137] 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 and Data Mining*, ser. KDD ’21. Virtual Event, Singapore: Association for Computing Machinery, 2021, p. 3220–3230.

[138] R. Wu and E. J. Keogh, “Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress (extended abstract),” in *Proceedings of the 2022 IEEE 38th International Conference on Data Engineering*, ser. ICDE’22, 2022.

[139] S. Kardani-Moghaddam, R. Buyya, and K. Ramamohanarao, “Performance anomaly detection using isolation-trees in heterogeneous workloads of web applications in computing clouds,” *Concurrency and Computation: Practice and Experience*, vol. 31, no. 20, 2019.

[140] P. P. Naikade, “Automated anomaly detection and localization system for a microservices based cloud system,” Master’s thesis, The University of Western Ontario, 2020. [Online]. Available: https://ir.lib.uwo.ca/etd/7109

[141] A. Samir and C. Pahl, “Dla: Detecting and localizing anomalies in containerized microservice architectures using markov models,” in *2019 7th International Conference on Future Internet of Things and Cloud (FiCloud)*. Istanbul, Turkey: IEEE, 2019, pp. 205–213.

[142] V. Ramamoorthi, “Machine learning models for anomaly detection in microservices,” *Quarterly Journal of Emerging Technologies and Innovations*, vol. 5, no. 1, p. 41–56, 2020. [Online]. Available: https://vectoral.org/index.php/QJETI/article/view/145

[143] J. Schneible and A. Lu, "Anomaly detection on the edge," in *Proceedings of the 2017 IEEE Military Communications Conference (MILCOM)*. Baltimore, MD, USA: IEEE, 2017, pp. 678–682.

[144] K. Yang, H. Ma, and S. Dou, "Fog intelligence for network anomaly detection," *IEEE Network*, vol. 34, no. 2, pp. 78–82, 2020.

[145] S. Kullback and R. A. Leibler, "On information and sufficiency," *Annals of Mathematical Statistics*, vol. 22, pp. 79–86, 1951. [Online]. Available: https://api.semanticscholar.org/CorpusID:120349231

[146] J. Kleinberg and E. Tardos, *Algorithm Design*. USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[147] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[148] T. Pitakrat, D. Okanović, A. van Hoorn, and L. Grunske, "Hora: Architecture-aware online failure prediction," *Journal of Systems and Software*, vol. 137, pp. 669–685, 2018.

[149] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. ki Cho, and H. Chen, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *International Conference on Learning Representations*, ser. ICLR '18, Vancouver, Canada, 2018.

[150] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, "A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems," *Engineering Applications of Artificial Intelligence*, vol. 85, pp. 634–644, 2019.

[151] 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)*. Madrid, Spain: IEEE, 2021, pp. 150–159.

[152] M. S. Islam, M. S. Rakha, W. Pourmajidi, J. Sivaloganathan, J. Steinbacher, and A. Miranskyy, “Anomaly detection in large-scale cloud systems: An industry case and dataset,” 2025. [Online]. Available: https://arxiv.org/abs/2411.09047

[153] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, “Algorithms for hyper-parameter optimization,” in *Proceedings of the 25th Annual Conference on Advances in Neural Information Processing Systems*, Granada, Spain, 2011.

[154] Y. Matsuo and D. Ikegami, “Performance analysis of anomaly detection methods for application system on kubernetes with auto-scaling and self-healing,” in *Proceedings of the 2021 17th International Conference on Network and Service Management (CNSM)*. Izmir, Turkey: IEEE, 2021, pp. 464–472.

[155] T. H. Haveliwala, “Topic-sensitive pagerank,” in *Proceedings of the 11th International Conference on World Wide Web*, ser. WWW ’02. New York, NY, USA: Association for Computing Machinery, 2002, p. 517–526.

[156] H. Wang, Z. Wei, J. Gan, Y. Yuan, X. Du, and J.-R. Wen, “Edge-based local push for personalized pagerank,” *Proceedings of the VLDB Endowment*, vol. 15, no. 7, p. 1376–1389, Mar. 2022.

[157] J. X. Parreira, D. Donato, S. Michel, and G. Weikum, “Efficient and decentralized pagerank approximation in a peer-to-peer web search network,” in *Proceedings of the 32nd International Conference on Very Large Data Bases*, ser. VLDB ’06. VLDB Endowment, 2006, p. 415–426.

[158] G. Yu, P. Chen, H. Chen, Z. Guan, Z. Huang, L. Jing, T. Weng, X. Sun, and X. Li, “Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments,” in *Proceedings of the Web Conference 2021*, ser. WWW ’21. Association for Computing Machinery, 2021, p. 3087–3098.

[159] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, “Fast unfolding of communities in large networks,” *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, oct 2008.

[160] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017. [Online]. Available: https://arxiv.org/abs/1412.6980