Service-level Prediction and Anomaly Detection Towards System Failure Root Cause Explainability in Microservice Applications

Raphael Rouf

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ARTS

Graduate Program in Information Systems and Technology

York University
Toronto, Ontario

April 2025

Abstract

Microservice and cloud computing operations are increasingly adopting automation. The importance of models in fostering resilient and efficient adaptive architectures is central to ensuring that services operate with expected behavior and performance. To effectively predict, detect, and explain system failures, it is essential to develop a comprehensive understanding of the affected application. This means examining its unexpected behaviors from multiple perspectives, including logs, metrics, and dependencies, to uncover the underlying root causes.

This thesis presents a novel approach to system failure prediction, root cause analysis and explainable failure type analysis by leveraging a three-fold modality of IT observability data: logs, metrics, and traces. The proposed methodology integrates Graph Neural Networks (GNN) to capture spatial information and Gated Recurrent Units (GRU) to encapsulate the temporal aspects within the data. A key emphasis lies in utilizing a stitched representation derived from logs, microservice events and resource metrics to predict system failures proactively. The traces are aggregated to construct a comprehensive service call flow graph and represented as a dynamic graph. Furthermore, permutation testing is applied to harness node scores, aiding in the identification of root causes behind these failures.

We evaluate our approach on open source datasets: MicroSS, QOTD and Train Ticket dataset that captures various types of system faults such as resource overload and wrong manipulation faults. Our findings on real world cases demonstrates that the proposed three-fold modality of observability data based on enhanced preprocessing and applying GNN-GRU and gradient distance explanation model captures failure type predictions well and explains them effectively for engineers to help debug and diagnose the issue.

Keywords: Microservice, System Failure, Multimodal data, Graph Neural Networks, Gated Recurrent Units, Temporal dynamics, Spatial features, Fault Injection, Anomaly Detection, System Failure Prediction, System Failure Localization, System Failure Root Cause Analysis

Co-Authorship

This thesis is based on my published works, detailed below, where I served as the first author. My primary contributions to these publications encompassed proposing research ideas and questions, conducting literature reviews, orchestrating experimental setups, designing experiments, data preprocessing, and data generation. Furthermore, I played a direct role in shaping the data modeling process and contributed towards the predictive analytics and root cause analysis work for service-level issues within the team. This thesis will be based on these published works.

The publications are listed as follows:

1. **Raphael Rouf**, Farhoud Kaleibar, Seema Nagar, Mohammadreza Rasolroveicy, Marin Litoiu, Prateeti Mohapatra, Pranjal Gupta, Ian Watts, *Identifying Failure Root Causes for Cloud-Native Microservice Applications*, Planning for Submission to Main Track (ACSOS 2025)

2. **Raphael Rouf**, Mohammadreza Rasolroveicy, Marin Litoiu, Seema Nagar, Prateeti Mohapatra, Pranjal Gupta, Ian Watts, *InstantOps: A Joint Approach to System Failure Prediction and Root Cause Identification in Microservice Cloud-Native Applications*, Accepted to 15th ACM/SPEC International Conference on Performance Engineering (ICPE 2024).

Acknowledgments

I would like to take this moment to thank my supervisor, Professor Marin Litoiu, for his support and guidance throughout my research and graduate studies journey. His expertise in Machine Learning, Performance Engineering, and Information Technology provided me with extensive knowledge and insights that deeply contributed to my research and development of this thesis. I am immensely grateful for his efforts to build my confidence and his strong commitment to help me acquire the skills necessary to succeed in publishing into well established Machine Learning & Performance Engineering conferences. I am deeply thankful to Dr. Manar Jammal and Dr. Marios Fokaefs for generously dedicating their time and expertise to serve as Committee Members for my Thesis Defence.

I also extend my sincere thanks to Professor Joydeep Mukherjee and Alexandru Bălu̧ă whose guidance and support during the early stages of my research were instrumental in shaping and initiating the foundation of my work. Their encouragement and collaboration were truly appreciated. Additionally, I am thankful to the CERAS Labs team members I had the privilege to work alongside. Thank you to Farhoud Kaleibar, Sarah Fehresti, Hamza Hussain, Harit Ahuja, and Mohammad Reza for their collaboration and knowledge sharing, which enriched my learning experience and contributed significantly to my progress.

Finally, I dedicate this thesis to my brother, Yar Rouf, and my parents, whose unwavering support, confidence, and encouragement helped me persevere during the challenging moments of my graduate studies. Their love and understanding have been a constant source of strength and motivation throughout this journey.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

With the growing use of cloud computing, enhanced system stability and reliability have become critical due to increased dependence on cloud servers for data storage and processing. Anomalies– unusual deviations from expected system behavior, can signify potential security breaches or malfunctions, posing risks to system availability and integrity [17, 4]. Anomaly detection and prediction are crucial in server and cloud environments for the immediate identification of abnormal activities, thereby preventing these issues from escalating into serious problems [10, 42]. For instance, in a microservice-based application, a sudden spike in latency or memory consumption in a critical service could indicate an imminent failure, potentially causing downstream effects that disrupt dependent services.

Anomalies can originate from the infrastructure, platform and application levels. Chaos Engineering– is the practice of running experiments to deliberately inject faults, known as chaos, into applications to cause anomalies. Injecting faults into the microservice applications provides how services reacts to failures based on the anomalous behavior and responses to faults. For example, infrastructure-level faults like network outages cause connection errors and increased latency, platform-level issues like server overloads lead to slower response times, while application-level bugs, such as memory leaks, result in excessive resource consumption and crashes. Injecting faults at different levels of the application to simulate anomalous behavior will reflect this unexpected behavior in trace, metric and log data sources. The anomalous data within these sources helps predict and detect the type of failure, providing Site Reliability Engineers (SREs) a strong call to action about potential issues that could affect the performance of their microservice application.

The automatic detection and prediction of anomalies and failures in microservice applications have garnered significant interest among researchers in recent years. Historically, anomaly detection has been a focus of extensive study. Techniques such as statistical meth-

ods, clustering, and rule-based systems have been employed for this purpose [12, 29, 16, 35, 14, 30, 8, 2]. However, as modern systems have become more interconnected and intricate, there has been a palpable shift towards leveraging machine learning models to achieve enhanced efficiency and accuracy in anomaly detection [10, 31, 9, 22].

To gain a holistic perspective of the system's health, research has particularly emphasized various single-modal data sources:

Traces in [45, 20, 39]: Traces offer insights into the flow of requests and the interactions among different microservices. Anomalies in traces, such as unexpected latencies or failed requests, can be revealing. For instance, a deviation from a usual 10-millisecond response time to a full second indicates potential concerns.

Logs in [11, 36, 28, 7]: Logs, the textual records generated by applications and infrastructure components, provide a comprehensive context about events, errors, and warnings. A surge in error messages or emergent warning patterns that deviate from the standard can pinpoint anomalies.

Resource and Performance Metrics in [33, 40, 23, 26]: Monitoring various metrics, including CPU usage, memory allocation, network activity, response time, and throughput, offers pivotal insights into the system's performance and health. Unforeseen spikes, drops, or inconsistencies in these metrics might be indicative of performance bottlenecks, inefficiencies, or other health-related concerns within a service.

However, recent studies [43, 19, 18] suggest that relying solely on single-modal data for failure localization may not be adequately efficient. One primary reason is that a single failure can have cascading effects on multiple facets of microservices. Such a failure might manifest itself in various modalities, leading to multiple anomaly patterns. For instance, a database slowdown might result in both extended response times (observable in traces) and a surge in error logs. Furthermore, there exist certain failures that might not be evident in specific modalities. If detection methods rely solely on one modality, they risk missing out on these anomalies. A classic example might be an internal logic error in a service that doesn't necessarily result in increased resource usage or evident trace anomalies but could still produce erroneous outputs.

Given these complexities, there's a growing consensus in the research community about the necessity of a multi-modal approach, combining data from various sources to create a holistic and more accurate picture of the application's health. The multi-modal approach would provide detailed data and insight for metric transformation– transforming numerical metrics into binary values suggesting anomalies occurring and log parsing– parsing log data using a log transformation tool to detect in real-time the system failures and fuse the insights gathered to provide explanations for the root cause of the system failure.

1.2 Research Questions

In this thesis, we focus on detecting resources' over-utilization (also known as overload) and wrong manipulation system fault from the anomalous behavior within logs, metrics, traces and events. The assumption is that in real-world applications, resource over-utilization and improper manipulations, simulated in our study, are often caused by external factors beyond the applications being monitored. Within this particular use case, our experiments are organized around several research questions, which are discussed below:

- **RQ1:** How efficient is multi-modal approach for system failure prediction compared to other neural network algorithms?
- **RQ2:** How does the efficiency of multi-modal approach for failure prediction compare to state of-the-art methods?
- **RQ3:** How does multi-modal approach perform root cause analysis as compared to other neural network algorithms?
- **RQ4:** How does multi-modal approach perform in terms of root cause node localization compared to existing methods?
- **RQ5:** How effective is our approach in classifying the types of predicted failures?

These research questions fall into two main areas: (1) prediction accuracy and (2) failure classification. The first four questions (RQ1, RQ2, RQ3 and RQ4) focus on evaluating the predictive efficiency of the multi-modal approach against existing neural network and state-of-the-art methods. The latter question (RQ5) assess the approach's effectiveness in classifying failure types and identifying the affected service within the microservice application. Together, these questions support the broader research goal of improving both the accuracy and reliability of system failure prediction and classification to assess and the basis to initiate remediation actions.

1.3 Contributions

In our approach, the workflow is divided into sequential phases to effectively process and analyze multi-modal data for system failure prediction and classification. The phases in this approach are designed to build upon one another. The first phase sets the stage for accurate system failure prediction by ensuring that both spatial and temporal aspects of the data are effectively modeled. The second phase extends this process by enhancing the quality of input data through anomaly detection and feature extraction, feeding refined signals into the GNN-GRU model, and ensuring that the insights gained from the model are both actionable and transparently explained. This sets the stage for leveraging these insights to identify failure patterns and provide actionable recommendations for SRE's to assist in mitigation planning.

The contributions of this thesis can be summarized as follows:

- **Prediction of System Failures:** We propose an enhanced multi-modal data preprocessing framework that effectively encodes signals from logs, metrics, and traces, improving the accuracy of failure prediction.
- **Classification of System Failure Types:** We develop a novel GNN-GRU model for multi-class classification, capable of predicting specific types of failures by leveraging the spatial and temporal dependencies present in multi-modal data.
- **Localization of System Failures:** We design an advanced node localization mechanism that leverages multi-modal data and graph-based structures to accurately identify the root cause node within a system when a failure occurs. By analyzing both spatial and temporal dependencies, this method enhances the precision of fault localization compared to existing techniques.
- **Root Cause Explanation of System Failure Types:** We introduce a method to produce explanations based on the gradient distance from observability data and its influence on predictions made by the GNN-GRU model to provide human-readable justifications, increase transparency and trust in the model's decision-making process.

Chapter 2

Background and Related Works

2.1 Overview

Recent works have significantly leaned into exploring methods for anomaly and failure detection within microservices and cloud applications, leveraging various data-oriented and machine-learning approaches [10, 31, 9, 22]. The exploration extends to real-time anomaly detection and the prediction of future system failures, prompting a dedicated focus on identifying the root causes that influence system failures and pinpointing the faulty services responsible. Chaos engineering plays a crucial role in generating multi-modal datasets to reveal a system's response to anomalies, especially when root causes are unexplainable or difficult to localize, anomalies go undetected, or predictions are unreliable, ultimately leading to service failures in the application.

The works surveyed [41, 32] also explored failure type classification and methods to grasp the reasons and causes behind failures and revealing their inherent challenges and limitations. These methods help aid in the understanding of existing system failure prediction, localization and anomaly detection with ML techniques to explain root causes of system failures, exploring avenues such as log analysis, dependency analysis from traces, and metric causality.

To the best of our knowledge, Diagfusion [41] is the only known work that includes failure type prediction but lacks explanations for the causes. In this thesis, we present a GNN-GRU-based method that predicts failure types and offers explanations, filling a key gap in multimodal AIOps research.

2.2 Background

In this section, we will go through the key concepts to define terms that are the background and foundation behind our research in service level prediction and anomaly detection towards system failure root cause Explainability in Microservice Application.

2.2.1 Micro-services & Cloud-Native Applications

There are a variety of architectural patterns and technologies that simplify the deployment and management of large-scale web applications. The microservice architecture is one of those architecture patterns that organizes applications as loosely coupled and independent services that communicate with each other's application programming interfaces (APIs) over lightweight Internet protocols, such as the Hypertext Transfer Protocol (HTTP) or Remote Procedure Calls (RPCs). Applications, composed of multiple distributed micro-services, are typically containerized and deployed to production environments leveraging Docker or kubernetes technologies [34].

To automate the life-cycle of containers, kubernetes have become the de facto standard in container orchestration in the industry. Kubernetes clusters comprise management or master nodes, worker nodes, and pods. A pod comprises one or more few containers grouped together that runs on worker nodes and is a single instance of an application or microservice. A pod can be scaled up or down depending on demand. Depending on application architecture, the number of pods and nodes required varies from application to application [15].

Container technology was first popularized by Docker by packaging application code, its dependencies, and a runtime into a container image that can then be deployed to any operating system that supported Docker. Doing so makes applications running as containers independent of the underlying operating system and eliminates code library conflicts with other applications running on the same system. The main advantage of micro-services architecture and containerization is that each individual microservice can be developed using frameworks and languages that best serve its particular purpose without having to standardize the framework and language across the entire application stack. Other benefits include the ability to distribute the micro-services across multiple hosts, increasing server efficiencies, automation of installation, scalability, manageability, and, importantly, fault isolation [34]. These micro-services can be grouped into n-tier architectures for improved manageability and scalability.

2.2.2 Workload Generation

To add to the complexity, large scale distributed applications will need to handle a significant amount of user demand and manage the application under varying loads to maintain system performance. Workload is the demand imposed on an application and can be measured in a variety of ways. One example is the rate of incoming requests to the web server. As increasing load imposes greater demands on underlying system resources, such as CPU and memory, with poor application response times can have a dramatic impact on user engagement. To put in perspective, an increase of 500 ms in response time would lead to a 20% reduction in user traffic for Google [6]. To ensure the microservice application is accurately modeled in our chaos engineering experiments, we generate synthetic workloads using JMeter, enabling an accurate representation of production applications.

2.2.3 Chaos Engineering & Failure Types

Chaos engineering is a study and practice of using fault injection tools to perform experiments in which faults are synthetically introduced into an application to create, analyze and detect anomalous behavior on microservice applications within a safe and self-contained environment. The fault injection tool we use in our work is called "Chaos Mesh", which is crucial for generating synthetic data for various realistic fault scenarios. For our research, these fault scenarios encompass types of resource utilization faults using a fault injection tool Table 3.1 and wrong-manipulation system faults (Figure 2.1) . The data generated with anomalies from the chaos engineering experiments is used as features and trained on our Anomaly Detection and System Failure Prediction Model for detecting anomalies of system failures, predicting service-level faults and ultimately explaining the failures root cause on Microservice Application.

2.2.4 Multi-modal Observability Data

Multi-modal Observability Data is the integration of various types of data collected from different sources within a microservice architecture. In our work, these sources include logs, metrics, traces, and events from microservice applications. Logs are textual records generated by the microservice application that provide context about events, errors, and warnings. Metrics are quantitative data points such as CPU usage, memory allocation, network activity, response time, and throughput that offer insights into the system's performance and health. Traces capture the flow of requests and interactions among different microservices, helping to identify anomalies like unexpected latencies or failed requests. Microservices events are specific events related to microservices, such as "Image Pull Back Off" or "PVC Pending," which provide additional context about the state and behavior of the system. By leveraging

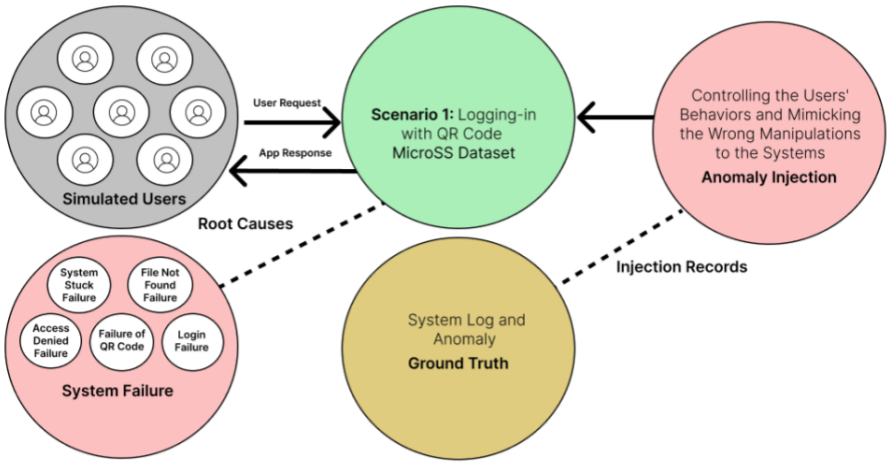

Figure 2.1: Chaos Engineering Process

multi-modal observability data and fusing these data-sources, the approach augments ecisting state-of-the-art research [43, 19, 18, 20, 41] and aims to predict system failures proactively and identify root causes more effectively.

2.2.5 Data Enhancement techniques

Log Semantic Parsing & Data Preprocessing is a technique to extract and classify golden signals from log lines by recognizing and interpreting patterns gathered from analyzing large amounts of textual log data. For our work, we use an log semantic parsing tool called BERT-Ops developed by IBM to extract features within a one-minute window. This provides deeper insights into the health of the application from the original the textual log data source. In our work, these golden signals are classified into traffic, saturation, latency, availability, and errors. By recording the occurrence of each golden signal in the application, we can use them as log features improve the GNN-GRU models ability to detect and predict system failures.

The Isolation Forest algorithm is designed to detect anomalies by isolating data points in a dataset [24]. It constructs a series of random binary trees by randomly selecting a feature and a split value between the minimum and maximum values of that feature. The key idea is that anomalies are easier to isolate because they are more likely to be separated earlier in the tree construction process. The anomaly score for a data point is computed based on the average path length of the point across all trees in the forest. The path length is defined as the number of edges traversed from the root node to reach the point. A higher anomaly score indicates that the data point is more likely to be an anomaly. These method enhances the model's ability to predict system failure types in microservice applications by improving the data quality and relevance to use for our use cases.

2.2.6 Nodes, Edges & Dynamic Dependency Graphs

We define nodes as individual services, each performing specific functions within the overall system. These nodes operate independently, communicating with other services by sending requests and receiving responses. Edges represent the interactions or communications between these services (nodes). These interactions are depicted as connections in a dynamic dependency graph, where edges indicate the flow of data or requests between services. In this work, the interactions between nodes are recorded as weights within the Graph Neural Network (GNN). These weights quantify the number of interactions between two services in the application.

Dynamic dependency graphs are used to represent the interactions and dependencies between microservices over time. These graphs are constructed using traces, which capture the flow of requests among services. By integrating logs and metrics as node attributes, the graphs provide a comprehensive view of the system's behavior at any given time. This enriched representation, which incorporates interactions and their weights within the GNN along with log semantic and metric as features enhances the GNN-GRU model's ability to predict specific types of system failures. The GNN-GRU model captures both the spatial and temporal dependencies within the microservice architecture to have a comprehensive overview of anomalies to recognize patterns from the series of anomalies and service interactions leading towards the system failures impacting the application.

2.2.7 Graph Neural Networks & Gated Recurrent Units (GNN-GRU)

The GNN captures the spatial relationships between micro-services, while the GRU captures the temporal dynamics. This integrated approach enables the prediction of system failures and the identification of root causes by analyzing the interactions and temporal changes within the microservice architecture. A Graph Neural Network (GNN) uses graph-structured data, capturing spatial relationships and interactions between nodes.

In the context of this paper, Graph Neural Network (GNN) are used to construct a dynamic dependency graph that represents the interconnections among microservices. Nodes in the graph represent individual services, and edges represent interactions between these services. The GNN captures the spatial features of the system at specific points in time. Gated Recurrent Unit (GRU) handles sequential data, capturing temporal dynamics. In this work, GRUs are used to understand how the spatial features captured by the GNN change over time. This helps in predicting system failures by considering the changes in the applications behavior over time.

2.2.8 Explainability

Explainability is the ability to uncover the underlying causes and interpret the model's reasoning behind its predictions of system failures in microservice applications. Effective explainability enables root cause analysis by identifying key anomalies across logs, metrics, and traces, providing SREs with a structured framework to quickly interpret and resolve issues.

For our gradient-based explanation, we identify which features (from logs, metrics, or traces) are most influential to identify specific failure types. By analyzing the gradients, the gradient-based explanation model can provide insights into the importance of each feature to pinpoint the key factors that contribute to system failures to identify the root causes for a clear call to take action by Site Reliability Engineers (SREs).

2.3 System Failure Prediction and Localization from Detected Anomalies for Microservice Applications

Zhao et al. [43] proposed a novel approach called AnoFusion for unsupervised failure detection through multimodal data for microservice systems. AnoFusion uses GTN to learn the correlation of the heterogeneous multimodal data and constructs a heterogeneous graph structure. Then, GAT is utilized to capture significant features and update the heterogeneous graph. Finally, GRU is used to predict the data pattern at the next step. However, while AnoFusion looks at the system level to predict the failures of the microservice, we focus our attention on the service level to localise which nodes are the leading cause in predicting the system failures and take that into account in our system failure prediction.

Li et al. [20] proposed TraceRCA, a root cause microservice localization approach designed for trace anomaly detection and flagging abnormal traces to predict the root cause.

Zhou et al. [45] presented MEPFL, a model designed to predict latent errors that possess the potential to precipitate failures, especially during the runtime in production environments of microservice applications. This approach is realized through the comprehensive analysis of system trace logs and the training of prediction models utilizing features distilled from these logs. In essence, MEPFL aims to empower developers by providing them with the capacity to identify and rectify faults before their manifestation as failures in a production setting. The model specifically addresses three predominant types of faults in microservice applications: system overload, memory leak, and sudden node crash. These fault types constitute nearly half of all microservice application faults, substantiated by existing empirical studies, underscoring the pivotal role and applicability of MEPFL in fortifying the reliability of microservice applications. The TraceRCA and MEPFL approaches are designed to take not only account traces but metrics, logs, and events using edge interactions of the nodes to localize the faults at the service level.

Lee et al. [19] introduced a novel approach named Hades, designed for detecting system anomalies in software systems. Hades seamlessly integrates heterogeneous data sources, including logs and metrics, to proficiently identify system anomalies. These anomalies, which encompass system failures, performance degradation, and other unanticipated behaviors, are detected promptly, thereby enabling system administrators to enact corrective actions swiftly. To facilitate the prediction of system anomalies, the authors employ a binary classification approach, wherein each data chunk is labeled as either 'normal' or 'anomalous.'

Lee et al. [18] proposed Eadro, an approach for anomaly detection within microservices. Eadro operates by modeling the standard behavior of microservices and identifying deviations from this established normalcy. Specifically, it employs a deep neural network to derive discriminative representations of microservice statuses through multi-modal learning, com-

pelling the model to apprehend fundamental features indicative of anomalies through multi-task learning. The model, which ingests multi-source monitoring data—including traces, logs, and Key Performance Indicators (KPIs)—generates a score for each microservice, reflecting the likelihood of an anomaly. A higher score indicates a greater probability of the microservice encountering an anomaly. Eadro's anomaly detection module is capable of identifying various types of anomalies, such as network-related issues, resource exhaustion, and software bugs. By pinpointing the root causes of anomalies, Eadro assists system administrators and developers in promptly troubleshooting and addressing issues.

2.4 Failure Type Classification and Explainable Root Cause Analysis from Detected Anomalies for Microservice Applications

Zhang et al. propose Diagfusion [41], a joint learning approach for root cause localization and failure prediction, though it lacks explanations for why failures occur. This approach leverages multimodal data to enhance fault detection by employing advanced embedding techniques fastText and data augmentation. It constructs a dependency graph and employs a graph neural network to pinpoint the root cause and identify the type of failure.

Ren et al. [32] alternatively focuses on detecting and explaining anomalies in microservice systems but does not go into classification of failure types. The framework uses a Spatial-Temporal Graph Convolutional Network (STGCN) to capture key information and deep SVDD to generate decision boundaries for detecting anomalies. A graph representation is employed to describe complex dependency relationships, with logs and metrics embedded into the node features. The framework aims to provide interpretable results that facilitate engineers' understanding and handling of incidents.

Garreau et al. [1] proposed an approach for a theoretical examination of the Local Interpretable Model-Agnostic Explanations (LIME) algorithm. It uncovers that LIME's explanations for linear models align with the gradients of the explained model and emphasizes the significant impact of parameter choices on feature importance determination. The research also quantifies the local surrogate error inherent in LIME, casting light on potential limitations in the reliability of its interpretations. These findings lead to actionable insights that can enhance the interpretability and trustworthiness of model explanations provided by LIME.

Chapter 3

Multi-Modal Binary Fault Prediction towards Failure Classification and Explainability

3.1 Multi-Modal Binary Fault Prediction Methodology

In this section, we propose InstantOps, a systematic approach designed to predict system failures and facilitate subsequent root cause analysis using GNNs in a Microservice application. We initiate with the standardization of features and then proceed to the systematic construction of a graph. In this representation, nodes symbolize system components, while edges correspond to the interactions between these components, quantified on a per-minute basis. To quantify the frequency of these interactions, we incorporate the concept of 'weights'.

We define a service,as shown in figure 3.1, as a particular aspect of the given microservice application that handles a specific function and interacts with other services to make up the microservice application, which we call the node(s). The Quote of the Day (Qotd) Microservice Application is a sample open source microservice application we used to simulate faults and run experiments to gather data on how the system reacts to service failures and performance degradation. In the case of the Quote of the Day (Qotd) Microservice application, each node performs a different service, such as: qotd-author, qotd-db, qotd-web etc. We extracted the source and target of the service information from the original MicroSS and Train Ticket datasets by preprocessing the logs and trace dataset of each microservice application. Our primary intention of the GNN-GRU Hybrid model is to identify and localize which node is predicted to fail in the next time window in a given mircroservice application.

When an interaction between two nodes (or services) is observed for the first time within a specified duration (from start-time to end-time), an edge is established between them with an initial weight of 1. If an edge between these nodes already exists within this timeframe, its weight is incremented by 1, marking an additional recorded interaction. Essentially, this

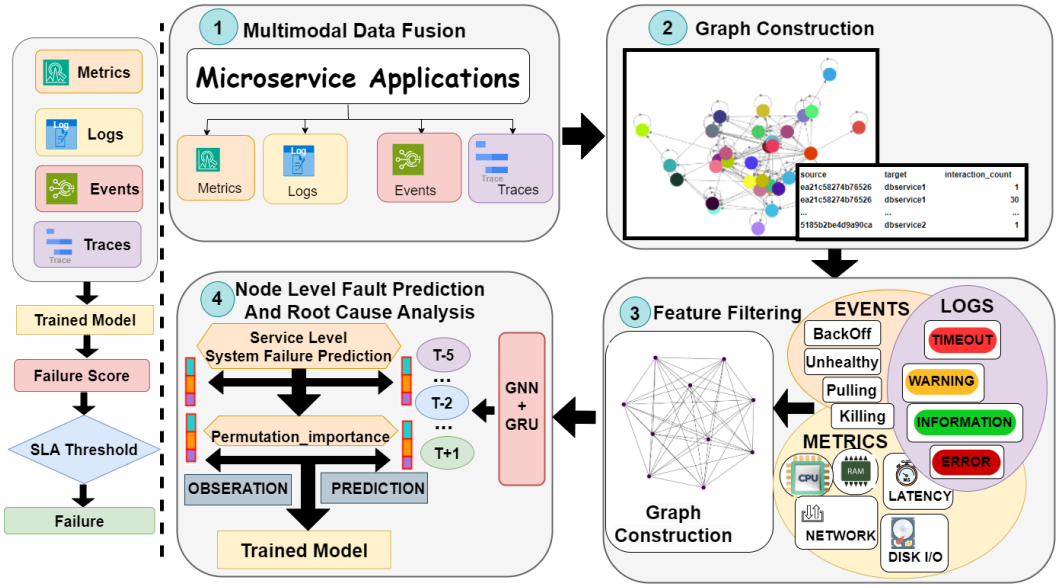

Figure 3.1: Multi-Modal Offline Training Model

weight serves as a metric, representing the number of interactions between two nodes within the designated interval. For example, if the weight of an edge between the direction of node A and node B is 10, it indicates that Service A interacted with Service B 10 times within the specified observation period.

Further, a GNN model is developed, integrating node feature information and graph topology to generate node embeddings. The mathematical framework defines the forward propagation and convolution operations within the GNN and Graph Convolutional Network (GCN) layers. The GNN was used for its resource efficiency from RNN while capturing node importance by incorporating edge features and enhancing the representation of interactions. Temporal aspects and dependencies of the system are incorporated by using a Gated Recurrent Unit (GRU) model. During the training phase, the cross-entropy loss function is employed to optimize model parameters to minimize the difference between predicted outputs and actual labels. We predict system failure when the entire application crashes or if the application receives 500 errors more than 99% of the time in the t + 1 time windows. This threshold was determined based on the strictest service level agreements and prior research, but it can be adjusted to meet the specific needs of the microservice application. Finally, the model's predictive accuracy, a quantifiable metric that measures the proportion of correct predictions relative to the total, is used to evaluate its ability to predict system failures and enable root cause analysis.

Here we discuss about the workflow of the progress in detail:

First, we preprocess and serialize the multi-modal data-set within a controlled time-frame.

To capture the heterogeneity and correlation multi-modal data including logs, traces, resource metrics and events (if they exist), we employ GNNs to construct a graph among the nodes. In our model, the layers are defined using GCNConv, which stands for Graph Convolutional Networks. Next, we construct graphs using edges, which are typically available in traces. These layers aggregate information from neighbors in a graph. We then filter features by selecting important keywords from logs and events that serve as features and correlating them with resource metrics. The first layer takes the features, while the second layer accepts the output from the first and produces additional graph neural network features.

After processing the node features with the GNN layers, the node embeddings are further refined through a GRU cell. For failure prediction, we consider both the features and temporal aspects across different timestamps to predict system failures in the subsequent time moment, which in our case is t + 1. This layer can capture temporal dependencies in the node embeddings produced by the GNN layers. While GRU models are traditionally used for sequence data, in our approach, we use it as an additional transformation for node embeddings.

3.1.1 Data Preprocessing and Graph Construction

Data normalization is conducted to maintain consistency among the dataset features, expressed mathematically as:

$$x_{\text{std}} = \frac{x - \mu}{\sigma}$$

where x is the raw feature, μ is the mean, and σ is the standard deviation of the feature across all data points. A graph G is then defined as $G = (V, E)$, where V is a set of nodes representing system components, and E is a set of edges representing interactions between nodes from the traces. The adjacency matrix $A \in \mathbb{R}^{n \times n}$ encapsulates the connections, where $A_{ij} = 1$ if a connection exists between nodes v_i and v_j, and 0 otherwise. Edges $e_{ij} \in E$ can also be defined based on dynamic count metrics from logs between nodes:

$$e_{ij} = f(\text{logs}_{ij}, \text{metrics}_{ij})$$

where f represents a function determining interactions between nodes based on logs and count metrics. In the logs, we would identify the interactions from each nodes by knowing when a node was accessed. For example, when it mentions that the db was accessed.

Node and Edge Definition

To understand the complex dependencies among various system components and their metrics, nodes V and edges E serve as critical entities in our graph G as they help in understand-

ing the complex dependencies among various system components and their metrics. These can be represented as:

$$V = \{v_1, v_2, \ldots, v_n\}$$

where n signifies the total number of system components or nodes. The edges, e_{ij}, symbolize the interactions or correlations among nodes v_i and v_j and can be mathematically defined by establishing a predefined similarity threshold θ as:

$$e_{ij} = \begin{cases} 1, & \text{if } \text{sim}(v_i, v_j) > \theta \\ 0, & \text{otherwise} \end{cases}$$

where $\text{sim}(v_i, v_j)$ conveys the degree of cosine similarity or interaction strength between nodes i and j.

3.1.2 Log Parsing

To predict system failure and their respective causes, serialization of the logs is an integral part of our work. The architectural design of this project is inspired by the previous work, BERTOps [11], which utilizes the approach from Drain [13] to extract structured information from raw log data by clustering log lines into templates based on their structural similarity and BERT [5] for building an encoder based Large Language Model (LLMs) for the log data. By fine-tuning the pre-trained BERTOps model on labeled data from downstream tasks such as log classification and fault category prediction, BERTOps can learn to accurately represent log data and perform various log analysis tasks with high accuracy. While this research paper aims to classify a log line, we classify each node and extract vital features from the logs, such as the number of errors a node receives within a specific time window. For instance, given the log structure "[2023-08-22T17:20:12.083] [Error] default - [418241] Quote request timeout", we extract the following components: timestamp, node id, number of errors.

3.1.3 GNN Model Formulation

Leveraging the ability of GNNs to capture localized graph structures and enable accurate predictions, the formulation and forward propagation within a GNN layer include the transformation and aggregation of node features across successive layers, adhering to:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

where:

- $H^{(l)}$ is the matrix of node features at layer l,
- $\tilde{A} = A + I$ includes the adjacency matrix A fortified with self-loops I,
- $\tilde{D}$ represents the degree matrix of $\tilde{A}$,
- $W^{(l)}$ denotes the weight matrix at layer l,
- σ embodies a non-linear sigmoid activation function.

This structural formulation of GNN simultaneously ensures the preservation of spatial relations between nodes and facilitates an iterative enhancement of node representations through the aggregation of neighboring information. This mechanism is instrumental in decoding intricate patterns, which are crucial for the predictive analysis of system failures, and provides a robust foundation for subsequent root cause analysis.

3.1.4 Graph Neural Network Model Development

The GNN model leverages both node feature information and topological structure. The forward propagation of a GNN model is often expressed as:

$$h_{v_i}^{(l+1)} = \sigma\left(\sum_{v_j \in N(v_i)} W^{(l)} \cdot h_{v_j}^{(l)}\right)$$

where $h_{v_i}^{(l)}$ represents the feature vector of node v_i at layer l and $N(v_i)$ is the set of neighbors of node v_i.

The operation in a Graph Convolutional Network (GCN) layer can be expressed as:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

where $\tilde{A} = A + I$ and $\tilde{D}$ is the degree matrix of $\tilde{A}$.

Enhancing the GCN model to include edge features, the node representation becomes:

$$h_{v_i}^{(l+1)} = \sigma\left(\sum_{v_j \in N(v_i)} W^{(l)} \cdot h_{v_j}^{(l)} + U^{(l)} \cdot e_{ij}\right)$$

where $U^{(l)}$ is a trainable weight matrix for edge representations at layer l.

3.1.5 Temporal Failure Prediction

In our approach, we integrate the power of GNN with GRU to predict temporal failures. The GNN captures the spatial structure of the data by operating on a graph, encapsulating local neighborhood information of each node through iterative feature aggregation from its neighbors. The propagation in GNN is steered by the adjacency matrix $\tilde{\mathbf{A}}$ and its diagonal degree matrix $\tilde{\mathbf{D}}$, formalized as:

$$\mathbf{X}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}^{(l)}\mathbf{W}^{(l)}\right)$$

The GRU captures the temporal dynamics across sequences. The GRU discerns sequential patterns using its intrinsic update and reset gates. The vital computations within the GRU include:

$$\begin{aligned}
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) && \text{(Reset gate)} \\
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) && \text{(Update gate)} \\
\hat{h}_t &= \tanh(W x_t + U(r_t \odot h_{t-1}) + b) && \text{(New potential hidden state)} \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t && \text{(Actual hidden state update)}
\end{aligned}$$

For each sequence and its respective time step, the GNN first takes in spatial information derived from the graph. The conjoined features from the GNN for every time step are subsequently funneled into the GRU, which modifies its hidden state according to its preceding state and the contemporary input. The culminating output of the GRU forms the basis of the prediction. The associated loss is computed relative to the true labels, followed by a backward pass to refine the model parameters. Upon training, this model can be employed to prognosticate failures for imminent time steps.

Figure 3.2 depicts the model architecture of a system designed for predicting system failure at the node level in a microservice application. This system employs a hybrid approach, integrating a GNN with a GRU model. The GNN component is responsible for capturing the interactions among microservice nodes, effectively learning from the topological structure of the microservices network. Each node in the GNN represents a microservice, and the edges reflect the interactions between these services.

The GRU part of the model handles the temporal aspects of the system's features, such as resource utilization and error rates, which are critical for understanding the state of the system over time. The GRU's ability to maintain information across time steps makes it particularly suitable for this task, as it can recognize patterns that precede a system failure.

The figure illustrates nodes representing different layers and operations within the neural network, such as convolutional layers (conv1.lin.weight, conv2.lin.weight), which are used in

Algorithm 1 Temporal Failure Prediction using GNN with GRU

procedure GNN_TemporalFailurePrediction(nodes, edges, y)
    Standardize *nodes*
    Map node names to integers in *edges* and *nodes*
    Convert *nodes*, *edges*, *y* to *data*
    Initialize GNN-GRU model with 2 GCN layers and a GRU layer
    for each epoch **do**
        for each batch in train_loader **do**
            Get temporal sequence of *nodes* for the current batch
            for each time-step t **do**
                $\mathbf{X}^{(l+1)} \leftarrow \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}^{(l)}\mathbf{W}^{(l)}\right)$
                $r_t \leftarrow \sigma(W_r x_t + U_r h_{t-1} + b_r)$
                $z_t \leftarrow \sigma(W_z x_t + U_z h_{t-1} + b_z)$
                $\hat{h}_t \leftarrow \tanh(W x_t + U(r_t \odot h_{t-1}) + b)$
                $h_t \leftarrow (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$
            Use h_t for prediction
            Compute loss with CrossEntropy
            Backward pass
            Update model parameters
        Evaluate on the validation set
        Compute metrics (Accuracy, Precision, Recall, F1 Score)
        Predict failure for $t + 1$ using current model state
    return Trained GNN-GRU model

Table 3.1: Conjoined features from the GNN

Features	**Traces**	**Metrics**	**Logs**	**Events**
# of Node Interactions	✓			
CPU Usage		✓		
Memory Usage		✓		
Disk I/O		✓		
Network I/O		✓		
5XX Errors			✓	
2XX Requests			✓	
4XX Errors			✓	
API Latency			✓	
CrashLoopBackOff				✓
ImagePullBackOff				✓
NodeNotReady				✓
PodScheduled				✓
NodeReady				✓
Unhealthy				✓
VolumeMount				✓
Failed (Image Pull)				✓
Resource Constraints				✓

processing nodes interactions, and GRU components (gru_cell.weight_ih, gru_cell.weight_hh), which are adept at handling temporal aspects. The AccumulateGrad nodes suggest the accumulation of gradient values for each parameter across multiple batches or time steps.

In the computational graph shown, we see the backward propagation flow, which is part of the training phase where the model's parameters are adjusted. The backward nodes represent the derivatives of the loss function with respect to the model's parameters, and the arrows indicate the direction of the gradients' flow.

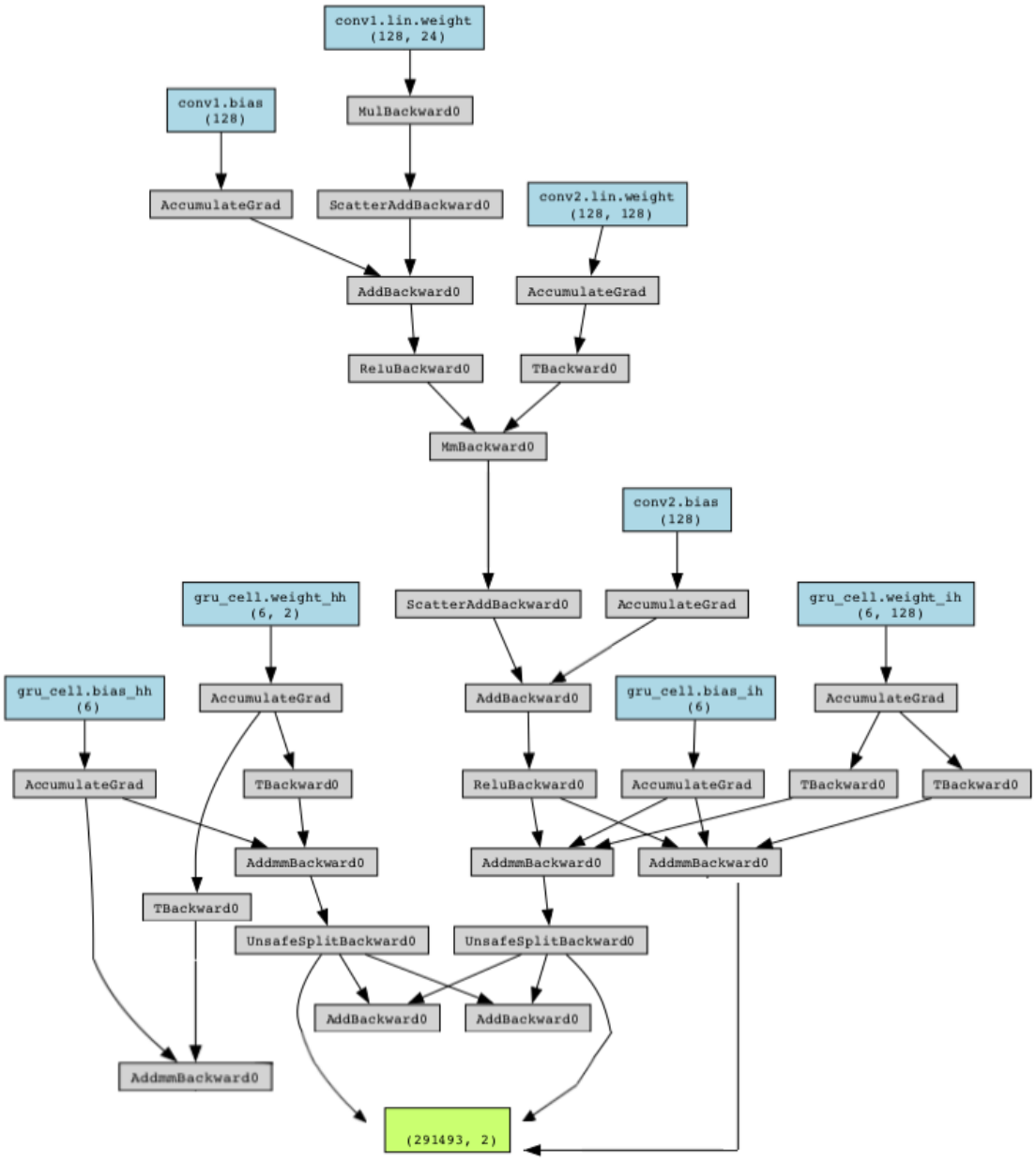

Figure 3.2: The Diagram for GNN+GRU Neural Network Model

3.1.6 Root Cause Localization Analysis of the Predicted Failures

In complex microservice applications, identifying the root causes of faults is critical for maintaining operational efficiency and reliability. To systematically analyze and diagnose faults in our system, we employ a two-fold metric-based approach. This approach focuses on evaluating the effectiveness of Multi-Modal root cause localization processes at the node level. Specifically, we utilize Mean First Rank (MFR) and Mean Average Rank (MAR) as our primary metrics.

MFR focuses on the position where the first correct item (fault) appears in a ranked list. In the context of fault analysis, this means identifying the most likely root cause of a problem as quickly as possible. The quicker a primary fault is identified, the faster remedial actions can be taken. This is crucial in systems where prompt fault resolution is essential to minimize downtime or prevent cascading issues. In practice, MFR is calculated by averaging the ranks at which the first true fault appears across all instances in a dataset. A lower MFR value indicates higher efficiency in pinpointing the primary fault quickly.

MAR extends the analysis to consider the average rank of all relevant items in the ranked list. This is particularly important in scenarios where multiple potential faults might contribute to a problem. MAR provides a broader view of the system's diagnostic accuracy. It is essential for comprehensive fault identification, especially in complex systems where multiple issues can coexist or be interrelated. MAR is calculated by averaging the ranks of all relevant faults across each instance in the dataset. It involves more intricate computations as it takes into account the position of each relevant item, not just the first one.

To implement these metrics, we utilized three different datasets, each comprising a ranked list of potential faults for each node generated upon the detection of a fault in the system. For instance, in the case of the QoTD datasets, we encountered three distinct faults: CPU, Memory, and Domain Name Service (DNS) errors. The ranked lists from these datasets are then scrutinized using the MFR and MAR metrics, allowing us to quantify the accuracy and efficiency of our fault identification process. This methodology ensures a robust analysis of the diagnostic capabilities at the node level within our system.

3.2 Multi-Model Failure Classification Root Cause Explainability Methodology

This section outlines our approach for detecting anomalous behavior in metric data to identify root causes of system failures. We employ log comprehension from raw logs to gain insights into system failures and pinpoint the faulty services from which anomalous behavior

originates, using trace data. The integration of insights from multiple data sources facilitates a comprehensive explanation of the root causes of system failures in the Microservices application.

Building on the InstantOps approach for binary classification of failure prediction, we implement multi-class classification to identify specific failure types of the Microservice Application. The workflow involves the following steps as shown in Figure 3.3: We begin by enhancing the preprocessing framework through the use of metric anomaly detection techniques to extract key metrics. Additionally, we employ golden signal classification to derive relevant features from logs, ensuring that crucial signals embedded in the raw data are effectively captured. Next, we utilized out developed GNN-GRU Model for multi-class classification with capabilities of predicting specific types of failures. This is achieved by constructing dynamic dependency graphs of microservice interactions, utilizing traces to enable the model to learn the complex patterns and relationships between microservices. Finally we employ a gradient based method to highlight the features most influential in the decision-making process to quantify the influence of each feature on the model's output, offering clear insights into the model's decision-making process. Lastly, we provide human-readable explanations of the model's decisions using a gradient-based approach, we highlight the influence of different features on the predictions.

3.2.1 Enhanced Data Preprocessing

This component utilizes metric anomaly detection techniques to extract meaningful features from metrics, and employs LLM-based golden signal classification [11] to derive relevant features from logs, ensuring that crucial signals embedded in the raw data are effectively captured for subsequent analysis (Table 4.1). We introduce two distinct preprocessing techniques tailored for metrics and logs to effectively capture relevant signals for failure type prediction.

Using Isolation Forest for Metric Anomaly Scores

The Isolation Forest algorithm is designed to detect anomalies by isolating data points in a dataset. Given a set of input features $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$, where each X_i represents a feature, the Isolation Forest constructs a series of random binary trees. Each tree is built by randomly selecting a feature X_i and then randomly selecting a split value between the minimum and maximum values of that feature. The key idea is that anomalies are easier to isolate because they are more likely to be separated earlier in the tree construction process.

The anomaly score $s(x)$ for a data point $x \in \mathbf{X}$ is computed based on the average path length $h(x)$ of the point across all trees in the forest. The path length $h(x)$ is defined as

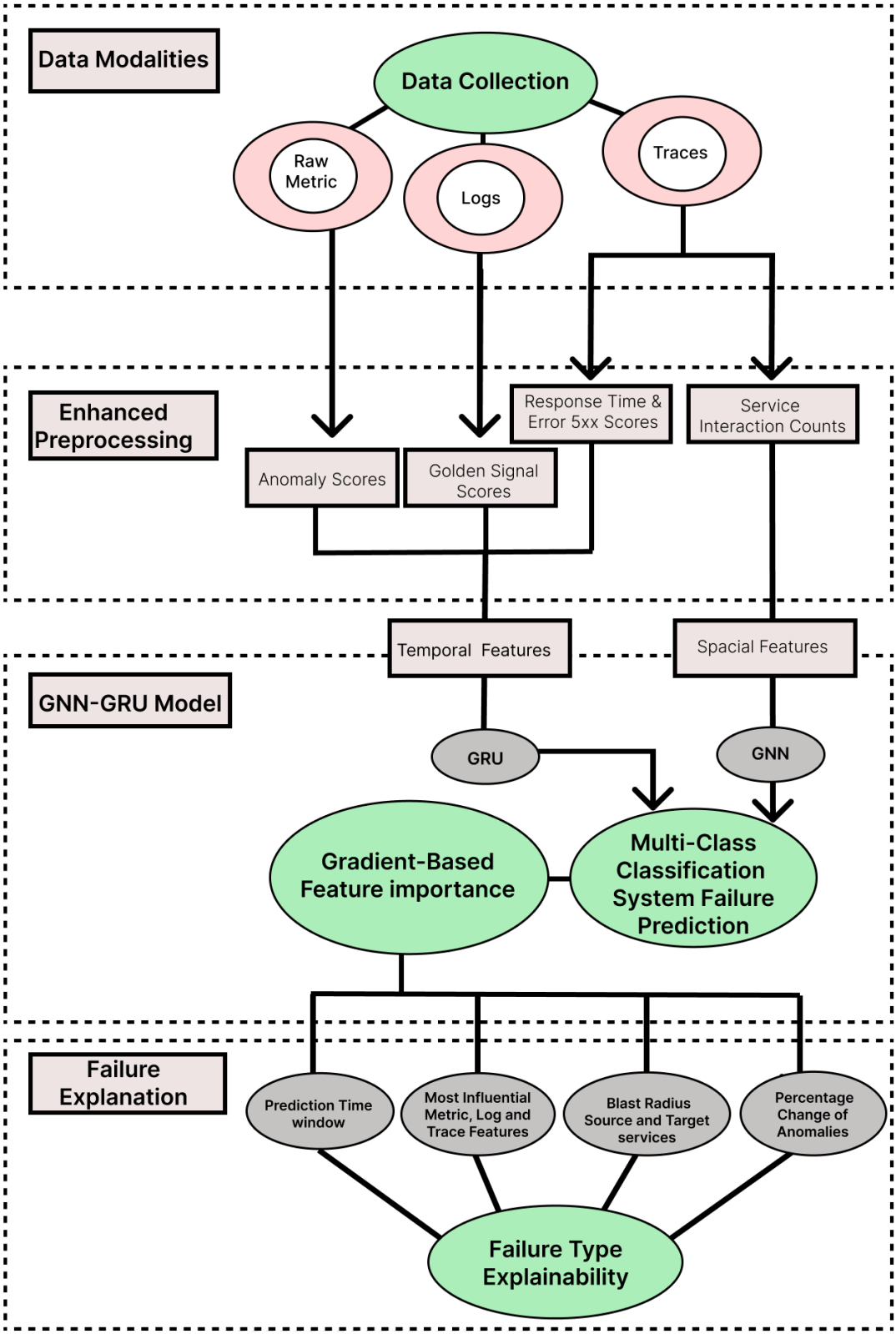

Figure 3.3: Multi-Class Classification Approach

the number of edges traversed from the root node to reach the point x. Mathematically, the anomaly score is given by:

$$s(x) = 2^{-\frac{h(x)}{c(n)}}$$

where $c(n)$ is the average path length of an unsuccessful search in a binary search tree, defined as:

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n}$$

and $H(i)$ is the harmonic number, $H(i) = \sum_{k=1}^{i} \frac{1}{k}$.

A higher anomaly score indicates that the data point x is more likely to be an anomaly. In our GNN-GRU model, these anomalies are utilized to enhance the model's ability to predict system failure type in the microservice application.

Log Semantic Parsing & Transformation

Logs are semi-structured text generated during service execution, often containing critical signals for predicting system failures. Given the potentially vast volume of logs, an effective approach to manage and analyze them is crucial. To address this, we first apply a templatization technique called Drain [13], a popular log parser tool that clusters logs into homogeneous templates (groups). This process enables us to represent a window of logs as a count vector of these templates.

Let $L = \{l_1, l_2, \ldots, l_n\}$ be a set of log lines generated within a specific time window. Using the Drain technique, we cluster these log lines into $T = \{t_1, t_2, \ldots, t_m\}$ templates. The log lines in the window are then represented by a count vector $\mathbf{c} = [c_1, c_2, \ldots, c_m]$, where each c_i denotes the count of occurrences of the template t_i in the window:

$$c_i = \sum_{j=1}^{n} \mathbb{I}(l_j \in t_i), \quad i = 1, 2, \ldots, m,$$

where $\mathbb{I}(\cdot)$ is the indicator function, which is 1 if the log line l_j belongs to template t_i, and 0 otherwise.

To further extract crucial insights from the logs, particularly the "golden signals" [3] embedded in log lines, we leverage the BERTOps[11] technique. The five key signals—*traffic*, *saturation*, *latency*, *availability*, and *error*—provide a clear picture of system health. BERTOps is a custom large language model (LLM) based on the BERT architecture, specifically designed for operational data analysis. Large language models like BERTOps work by learning patterns from vast datasets, enabling them to understand and interpret complex, unstruc-

tured text data, such as logs. Through contextual understanding, BERTOps identifies critical patterns and classifies log lines according to the relevant golden signals. For each log window, we aggregate the signals into a five-dimensional vector $\mathbf{g} = [g_1, g_2, g_3, g_4, g_5]$, where each g_k represents the count of log lines associated with the k-th signal. This integration of advanced natural language processing allows us to monitor, diagnose, and predict system failures with greater accuracy.

$$g_k = \sum_{j=1}^{n} \mathbb{I}(l_j \text{ contains signal } k), \quad k = 1, 2, \ldots, 5.$$

Finally, a window of log lines for a given period is represented by concatenating the template count vector $\mathbf{c}$ with the golden signal count vector $\mathbf{g}$, resulting in a comprehensive and informative feature representation vector $\mathbf{f}$:

$$\mathbf{f} = [\mathbf{c} \mid \mathbf{g}],$$

where $[\cdot \mid \cdot]$ denotes the concatenation operation. This feature vector $\mathbf{f}$ effectively captures both the structural and semantic information of the log data.

3.2.2 Construction of Dynamic Dependency Graphs and Failure Type Prediction using GNN-GRU Modelling

This component focuses on constructing time-evolving microservice dependency graphs using traces and GNN-GRU modelling. These graphs dynamically represent the interactions and dependencies between microservices over time. Features inferred from other modalities, such as logs and metrics, are integrated as node attributes, enriching the representation of each microservice's state and interactions. The GNN-GRU modelling captures the data's temporal and spatial aspects. By leveraging the constructed dependency graphs, the GNN-GRU model performs multi-class classification to predict the specific type of failure, effectively learning complex patterns and dependencies indicative of various failure types. We propose a systematic approach utilizing GNNs combined with GRUs. This approach begins with the standardization of features, followed by the construction of a graph where nodes v_i represent system components (services), and edges e_{ij} represent the interactions between these components, quantified per minute. The frequency of these interactions is encoded as edge weights w_{ij}, which are initialized to 1 upon the first interaction within a given time frame and incremented by one for each subsequent interaction within the same period.

The graph $G = (V, E)$ is constructed, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes representing the microservices, and $E = \{e_{ij}\}$ is the set of edges representing the interactions between nodes. The edge weights w_{ij} serve as metrics for the interaction frequency between

nodes v_i and v_j during the observed interval.

To model the interactions and predict potential failures, we develop a GNN that integrates node feature information $\mathbf{x}_v$ and the graph topology. The propagation of information through the GNN is governed by a series of convolution operations. For each node v_i, the embedding $\mathbf{h}_i^{(l+1)}$ at layer $l+1$ is updated based on the embeddings of its neighbors $\mathcal{N}(v_i)$ at layer l and the edge features w_{ij} as follows:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(v_i)} w_{ij} \mathbf{W}^{(l)} \mathbf{h}_j^{(l)} + \mathbf{b}^{(l)}\right)$$

where $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight matrix and bias vector for layer l, and σ is the activation function.

Following the GNN layers, the node embeddings $\mathbf{h}_i^{(L)}$ from the final layer L are further processed through a GRU to capture temporal dependencies across different time steps. The GRU updates the hidden state $\mathbf{h}_t$ at time t as follows:

$$\mathbf{h}_t = \text{GRU}(\mathbf{x}_t, \mathbf{h}_{t-1}) = (1 - \mathbf{z}_t) \circ \mathbf{h}_{t-1} + \mathbf{z}_t \circ \tilde{\mathbf{h}}_t$$

where $\mathbf{z}_t$ is the update gate, and $\tilde{\mathbf{h}}_t$ is the candidate activation. The GRU allows the model to consider both the current node embedding $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$, thereby capturing temporal dynamics in the data.

3.2.3 Explanations for Failure Type Prediction

This component explains the predictions made by the GNN-GRU model that is achieved by employing a gradient-based method to highlight the features most influential in the decision-making process, offering human-readable justifications that enhance the transparency and trustworthiness of the model's outputs. This approach allows us to quantify the influence of each feature on the model's output, offering clear insights into the model's decision-making process.

First, gradients for input features are calculated to understand which features most influence the model's predictions. This is achieved by setting the input features to require gradients, running a forward pass through the model, and back-propagating the loss to obtain the gradients. These gradients are then used to compute the L2 norm, giving a measure of the gradient's magnitude for each feature, which indicates their influence on the prediction. We can identify the critical interactions between nodes (services) in the graph by focusing on the most influential features and their associated gradient distances. The edge interaction of interest is pinpointed by mapping the nodes in the model's edge index back to their

original names or IDs. The blast radius (the extent of the impact) is assessed by examining the gradient changes associated with the interaction between these nodes, allowing us to understand which features and interactions contributed most to the failure, and how they propagated across the network. This information is then used to generate detailed explanations for each prediction, capturing the key features, the extent of their impact, and the specific node interactions involved.

Chapter 4

Experimental Evaluation

In this section, we present the data for evaluation and then evaluation metrics for our approach against baselines for each phase of our workflow. This assessment would help understand the underlying factors contributing to system failure prediction, classification and value of the explanations to SREs derived from our approach.

4.1 Phase 1: Multi-Modal Binary Fault Prediction

4.1.1 Datasets & Tools

In this phase, we utilized two open-source microservice datasets: Train-Ticket and MicroSS, and Quote of the Day (QoTD) application that was deployed in-house on IBM OpenShift Clusters V4.12.36 with 16 CPU cores, and 32 GB Ram.

Train-Ticket [44] which comprises 41 microservices, is frequently used by researchers for root cause identification and localization. We accessed this application made available online by Li et al., [20] to enable comparative analysis with their findings. Three types of faults were introduced: application bugs, CPU exhaustion, and network congestion. While these faults were introduced at various system levels, we specifically focused on those injected into the microservices, aligning them for comparison with the other two datasets.

MicroSS, also known as the Genetic AIOps Atlas (GAIA) dataset [1], contains 10 microservices, two databases (MYSQL and Redis), and is supported by five host machines. It is designed to cater to both mobile and PC users. The GAIA dataset encompasses five distinct faults: system hang-ups, process crashes, system failures like login issues, missing files, and access denials. A record detailing the injection of these failures is provided alongside the data. This dataset has been widely used for predicting system failure and root cause localization in [43] and [41].

The QoTD open-source application [2] consists of eight distinct microservices. We deployed

[1] https://github.com/CloudWise-OpenSource/GAIA-DataSet
[2] https://gitlab.com/quote-of-the-day/quote-of-the-day/-/tree/master

QoTD on IBM OpenShift clusters and introduced a variety of faults using Chaos-Mesh [3]. These faults include disruptions in CPU, memory, and DNS.

For monitoring purposes, we employed Instana [4], which provided data points such as API response times, error codes, and various resource utilization metrics: CPU, memory, disk, and network. We utilized LogDNA [5] to extract application logs. Furthermore, we devised a custom script to capture events per minute at the node level. This script monitored events like "Scheduled", "Unscheduled", "Pulled", "Failed", "Started", etc. These events were retrieved by querying `'oc describe <resource> <resource-name>'`.

4.1.2 Evaluation Metrics

Failure Prediction:

The model is trained using CrossEntropy loss, formulated as:

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

where y_i and $\hat{y}_i$ are the true label and predicted probability for sample i, respectively.

To assess the model, accuracy and F1 Score are computed. Accuracy is given as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

F1 Score is calculated using precision and recall, which consider the model's performance regarding false positives and false negatives. F1 Score is expressed as:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

with

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

and

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Root Cause Analysis:

To validate and understand the impact of crucial nodes for the root cause, we employed a function for evaluating the accuracy without k nodes. This function deliberately nullifies the top k nodes (based on their importance) and evaluates how the model's accuracy is affected without their presence:

[3] https://chaos-mesh.org/
[4] https://www.ibm.com/products/instana
[5] https://www.ibm.com/case-studies/logdna-cloud

- **Ranking Nodes:** Nodes are ranked in descending order based on their computed importance.
- **Nullification of Top Nodes:** Features of the top k nodes are set to zero, effectively removing their influence from the network.
- **Model Evaluation:** With these nodes nullified, the model's accuracy is gauged again, highlighting the impact of these crucial nodes on the network's overall performance. A pronounced drop in these accuracies compared to the baseline underscores the critical nature of these nodes within the network.

Top-k accuracy is a metric commonly used in retrieval and recommendation tasks. It measures how often the true item (or one of the true items) appears in the top k items of the ranked list and is defined as:

$$A@k = \frac{1}{|A|} \sum_{a \in A} \begin{cases} 1 & \text{if } RC_i^a \in RC_s^a[k] \\ 0 & \text{otherwise} \end{cases}$$

where,

- A is the set of test samples.
- RC_i^a is the true root cause instance for sample a.
- $RC_s^a[k]$ is the set of top-k predicted instances for sample a.

MFR evaluates the average rank at which the first correct item is found in the ranked list and is defined as:

$$\text{MFR} = \frac{1}{N} \sum_{i=1}^{N} \text{rank}_i$$

where,

- N is the total number of ranked lists.
- rank_i is the rank of the first true item in the i^{th} ranked list.

MAR measures the average rank of all relevant items in the ranked list. It's a s metric when there are multiple relevant items per query, and you want to assess how well the system ranks all of them on average and is defined as:

$$\text{MAR} = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{1}{|R_i|} \sum_{r \in R_i} \text{rank}_{i,r} \right)$$

where,

- N is the number of ranked lists.
- R_i is the set of relevant items for the i^{th} ranked list.
- $\text{rank}_{i,r}$ is the rank of relevant item r in the i^{th} ranked list.

4.2 Phase 2: Multi-Model Failure Classification Root Cause Explainability

4.2.1 Datasets

For this phase, we solely use the widely used open-source MicroSS dataset[6] also known as the Genetic AIOps Atlas (GAIA) for system failure type prediction that contains 10 microservices, two databases (MYSQL and Redis), and is supported by five host machines. This dataset encompasses four faults: system stuck, login issues, missing files, and access denials and consists of Metrics, Logs and Trace data. A record detailing the injection of these failures is provided alongside the data to evaluate the type of failure prediction approach. Table 4.1 summarizes the features constructed from all three data modalities.

Table 4.1: Features Categorized by Metric, Log, and Trace

Data Type Category	**Features**
Metric	CPU_total_pct_Score Memory_usage_total_Score Diskio_summary_Score Diskio_write_rate_Score Network_out_packets_Score Network_in_packets_Score
Log Counts	Availability_Count Latency_Count Exception_Count Saturation_Count Error Counts_Count Template_Count
Trace Anomaly Scores	Errors_minute_Score Response_time_Score

Our evaluation leverages diverse dataset collected from metrics, logs and traces to obtain a comprehensive view of the application crucial for system performance monitoring and identifying the types of failures that may occur in the application. The metrics include CPU utilization, memory allocation, diskio summary and write rate, network out and in packets response times. The trace features include error rate and response time. We leveraged the log data to obtain semantic insights using the BERTOps method which consists of exception, availability and latency counts. We preprocess the raw values of metrics into anomaly scores. Table 4.1 summarizes the features available from all three data modalities. We enhance the metric features by corresponding metric anomaly scores of each feature and obtained a comprehensive dataset for assessing the robustness and accuracy of our system in detecting

[6] https://github.com/CloudWise-OpenSource/GAIA-DataSet/tree/main/MicroSS

and explaining system failures. Therefore the failure type counts represent the failure types on each service interaction per minute which we use to make predictions to identify the interactions with the most impact for causing the fault.

We define the window size as one minute, aggregating the features derived from metrics, logs, and traces, as well as the interactions between services within each one-minute interval. Ensuring that all interactions within the same minute are kept together preserves the integrity and relevance of the data, preventing disruption in the analysis. Splitting interactions within the same window would fragment the data, reducing its effectiveness for meaningful predictions.

The distribution of data points across different failure types is shown in Table 4.2. To maintain proportionality and ensure that the training and testing sets are representative of all failure types, we employ a stratified split with a 70-30 distribution ratio of the different fault types in the training and test sets. Since multiple services operate concurrently each minute, we select the service that has the greatest impact, identified through gradient distance analysis, to represent the fault type for that minute. This 'lead service' represents this fault type.

This approach maintains the relative frequencies of each failure type in the dataset, ensuring that the training set, comprising 70% of the data, is optimally prepared to learn from varied instances of failures. Meanwhile, the remaining 30% allocated to the test set impartially assesses the model's ability to generalize across these critical failure scenarios without bias introduced by skewed distributions.

After making predictions, we identify the service originating the fault with the max impact by determining the highest gradient distance. We then focus on this primary service interaction as the lead indicator of the failure, disregarding other service interactions from that minute, and generate explanations based on this analysis. The number of failure types drastically decrease after making the prediction for the explanation due to the fact that we firstly disregard scenarios where we don't have all data sources and secondly, we narrow down the failure type to the max abnormalities. Therefore, the model is trained on each service interaction occurring per minute to identify the failure type and then the service interaction with the greatest gradient distance impacting the is chosen as to identify the source service of the fault type and target to investigate the blast radius.

Table 4.2: System Failure Type Counts per minute

Fault Type	**Per Service**	**Per Lead Service**
File Not Found	2,621	39
System Stuck	2,116	12
Access Denied	1,266	15
Login Failure	53,838	660

4.2.2 Evaluation Metrics

Evaluation Metrics for Failure Prediction:

The model is trained using CrossEntropy loss, formulated as:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

where y_i and $\hat{y}_i$ are the true label and predicted probability for sample i, respectively. To assess the model, accuracy and F1 score are computed.

The model leverages the applications topology to analyze the service interactions with features to train on service and infrastructure performance and frequency of log semantic abnormalities. Training the model on a variety of abnormality types on each service interaction provides extensive data on each minute for classifying the type of failure.

Accuracy is given as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

F1 Score is calculated using precision and recall, which consider the model's performance regarding false positives and false negatives. F1 Score is expressed as:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

with

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

and

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

System Failure Type Explanation

To validate and understand the impact of specific metric, log, and trace anomalies on the type of system failure, we evaluated on four failure types for each service interaction, identifying the service most influencing the prediction:

- **Service Interaction Data:** Performance metrics, logs, and traces were collected per-minute for each service interaction, providing a detailed view of the impact of each interaction.
- **Model Evaluation:** The model predicts failure types using input features and the graph structure of the services, emphasizing the importance of critical service interactions on overall network performance.

- **Service-Level Failure Type Impact:** Gradient distance was used to determine the service most significantly influencing the prediction during each interaction. This approach highlights the leading indicator and source of the failure for further investigation. It enables analysis of abnormalities in both source and target services and identifies the blast radius of the failure.

We identified the most impactful service interaction by analyzing gradient distances, marking it as the key failure indicator. Explanations were generated based on this analysis, focusing on the localized source and target services and their associated abnormalities. These explanations were formalized into an **Explanation Template**, which provides insights on the failure's occurrence time, the evaluated prediction, contributing metrics, and contextual details about the failure's origin.

The explanations were derived from the model's fault classification and the features contributing to these failure predictions. The explanation is intended to provide a clear, data-driven summary of the failure type, its timing and location, and the most influential metrics, logs, and traces contributing to the model's decision. To evaluate the model's output, we used the template in Table 4.3 to generate explanations based on gradient distances of anomaly features. Qualitative analysis was conducted by manually comparing these explanations with application metrics, logs, and trace graphs across randomly chosen instances to observe correlations.

Table 4.3: Failure Type Explanation Template and Sample Explanation

Failure Type Explanation Template	**Sample Explanation for Failure Type Prediction**
Timestamp: [timestamp]	**Timestamp**: 2021-07-18 16:00:00
Prediction: [Failure Type] Failure	**Prediction**: Access Denied Failure
Contributing Metrics: 1. **[Metric 1]**: [Type of anomaly with XYX% increase/decrease] 2. **[Metric 2]**: [Type of anomaly with XYX% increase/decrease] 3. **[Metric 3]**: [Type of anomaly with XYX% increase/decrease]	**Contributing Metrics**: 1. **CPU total pct**: Gradient distance increase of 0.04% 2. **Exception count**: Gradient distance increase of 0.03% 3. **5xx errors per minute count**: Gradient distance increase of 0.27%
Context: These anomalies were detected during communication between [Service 1] and [Service 2].	**Context**: These anomalies were detected during communication between `redisservice2` and `dbservice1`.

4.3 Generalizability of InstantOps

Our approach was developed and tested on 3 datasets representing small to medium-sized microservice applications. We prioritized computationally efficient models to ensure that training time remained under five minutes. Given this, our approach is generalizable to larger microservice systems, provided it is retrained on datasets that reflect the increased complexity and scale. This retraining would allow both the GNN and GRU models to effectively adapt to more extensive application environments.

The approach leverages observability data from all available monitoring sources—logs, metrics, and traces—without depending on any single predefined feature. However, to achieve consistent performance when generalizing to new microservice applications, similar types of observability data must be available, as the model relies on this comprehensive data to maintain its predictive performance.

Chapter 5

Results and Discussions

5.1 Multi-Modal Binary Fault Prediction: Localization of System Failures Root Causes on the Service-Level

In this section, we focus on the resources' over-utilization (also known as overload) use case as the cause of anomalies. The assumption is that resource utilization is driven by factors external to the applications being monitored. Within this particular use case, our experiments are organized around several research questions, which are discussed below.

- **RQ1:** How efficient is the Multi-Modal approach for system failure prediction compared to other neural network algorithms?
- **RQ2:** How does the efficiency of the Multi-Modal approach for failure prediction compare to state-of-the-art methods?
- **RQ3:** How does the Multi-Modal approach perform root cause analysis as compared to other neural network algorithms?
- **RQ4:** How does the Multi-Modal approach perform in terms of root cause node localization compared to existing methods?

RQ1: How efficient is the Multi-Modal approach for system failure prediction compared to other neural network algorithms?

To assess the efficiency of our Multi-Modal approach, three distinct datasets were utilized, as shown in Table 5.1. The first, MicroSS, is an open-source dataset comprising 419,959 data. This data was divided into training and testing sets with 335,933 and 84,026, respectively. The Train Ticket open-source dataset encompasses 24,492 data, which are further divided into 5,085 for testing and 19,337 for training. Lastly, our generated dataset QOTD includes 450,000 data, with a distribution of 90,000 for testing and 360,000 for training.

Table 5.1: Data Overview for Applications

Application	Total Samples	Test_data	train_data
MicroSS	419959	84026	335933
Train Ticket	24492	5085	19337
QoTD	450000	90000	360000

For each dataset, several application metrics were considered which we defined as features: application resource utilization, logs detailing errors for each node within specific time frames (e.g., 1 minute), edge constructions that link nodes with traces within certain time windows, and the frequency of interactions between nodes. These metrics assist in identifying anomalies by observing deviations in standard node interactions. Specifically for the QoTD dataset, we also incorporated a subset of trace data called events, capturing the count values of events such as out of memory, scheduled activities, pull events, failed creation events, and image pull back-offs on a per-node basis. Using this data, we aimed to predict systemic failures.

We adhere to a standard Service Level Agreement (SLA) that designates a system as "failed" if 99.9% of the total requests received on the server resulted in errors, such as a 503 error, within a specific time frame (e.g., 1 minute). The labeled dataset was employed to validate the accuracy of our predictive efforts.

Further validation was sought by leveraging multiple neural network models, as delineated in Table 5.2. These models, previously utilized by other researchers [43, 41], were examined to compare the effectiveness of our Multi-Modal approach. Notably, our Multi-Modal approach employs a Graph Neural Network (GNN) to structure the relationships between features and nodes. Additionally, a Gated Recurrent Unit (GRU) captures temporal aspects of each node in the system, such as time lags of t-2 and t-5.

The performance metrics revealed that our Multi-modal approach achieved the highest F1 score for MicroSS at 0.98, with a recall of 0.98 and a precision of 0.97. Similar scores were observed for the Train Ticket dataset. For the QoTD dataset, the F1 score was 0.96, with a recall and precision both measuring 0.96. When comparing other neural network algorithms using the MicroSS dataset, the combination of GTN with GRU and LSTM yielded an F2 score of 0.94. The Train Ticket dataset registered a score of 0.92 for the integration of GTN and LSTM, while GNN and LSTM achieved 0.94.

Table 5.2: Comparison of different algorithms for MicroSS, Train Ticket, and QoTD datasets

	MicroSS				Train Ticket				QoTD			
Algorithm	Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1
InstantOps	**0.97**	**0.97**	**0.98**	**0.98**	**0.98**	**0.97**	**0.98**	**0.98**	**0.97**	**0.96**	**0.96**	**0.96**
GNN+LSTM	0.93	0.93	0.93	0.93	0.91	0.87	0.91	0.89	0.91	0.94	0.94	0.94
GNN	0.90	0.93	0.90	0.91	0.87	0.83	0.87	0.85	0.92	0.93	0.92	0.93
GTN	0.73	0.93	0.73	0.81	0.92	0.86	0.92	0.89	0.88	0.89	0.88	0.88
GTN+LSTM	0.95	0.93	0.96	0.94	0.93	0.92	0.96	0.92	0.91	0.90	0.94	0.93
GTN+GRU	0.96	0.92	0.96	0.94	0.93	0.86	0.93	0.90	0.92	0.94	0.89	0.92

RQ1: We showed that by fusing traces, logs, and metrics and capturing temporal aspects using GRUs to identify anomalies by observing deviations in the interactions between services, our Multi-modal approach obtained higher precision, recall and F1 scores in comparison to other neural network algorithms on three data-sets: QoTD, MicroSS and Train Ticket.

RQ2: How does the efficiency of our Multi-modal approach for failure prediction compare to state-of-the-art methods?

Table 5.3 illustrates a comparison study between our Multi-modal approach, JLT and Anofusion to establish a multimodal baseline.

The JLT method aggregates the results from JumpStarter [27], LogAnomaly [28], and Traceanomaly [25]. It employs majority voting, marking a failure if two or more modalities fail simultaneously. A notable observation about JLT is that it disregards the correlation among different modalities. For the MicroSS dataset, which integrates metrics, logs, and traces, JLT achieves an F1 score of 0.61, a recall of 0.94, and a precision of 0.46.

To further improve JLT's performance for fault prediction, [43] proposed Anofusion which uses a graph neural network to learn the correlations in the heterogeneous multimodal dataset. For the same MicroSS dataset, Anofusion achieved an F1 score of 0.85, a recall of 0.94, and a precision of 0.79.

In our Multi-modal approach, we have further refined the Anofusion model by constructing a graph based on traces, which displays interactions among the nodes in real time. Our Multi-modal approach employs both GNN and GRU to account for the system's temporal aspects. Our experimental results indicate that our Multi-modal approach, when applied to the MicroSS dataset, achieves an F1 score of 0.98, a precision of 0.97, and a recall of 0.98.

Table 5.3: The Average Percentage Among Precision, Recall, and F1-Score of Different Approaches on MicroSS Dataset

Approach	Modality			MicroSS Dataset		
	Metric	Log	Trace	Prec.	Rec.	F1
InstantOps	✓	✓	✓	**0.970**	**0.980**	**0.980**
AnoFusion	✓	✓	✓	0.795	0.945	0.857
JLT	✓	✓	✓	0.461	0.940	0.618

RQ2: We showed that by constructing a dependency graph based on traces to depict the service interactions of the faulty service using GNN and GRU to incorporate temporal aspects, our Multi-modal approach achieves higher precision, recall and F1 score than the state-of-the-art methods.

RQ3: How does our Multi-modal approach perform root cause analysis as compared to other neural network algorithms?

To assess the efficiency of `InstantOps` in root cause localization at the node level, we used two metrics: MFR (Mean First Rank) and MAR (Mean Average Rank) described earlier on three datasets: MicroSS, Train Ticket, and QoTD as shown in Table 5.4.

As can be seen from the table, our Multi-modal approach achieves an MFR score of 1.49 and a MAR score of 1.51 on the QoTD dataset. On the MicroSS dataset, our Multi-modal approach achieves a score of 1.51 for both MFR and MAR. On the Train Ticket dataset, our Multi-modal approach achieves a score of 1.06 for both MAR and MFR. We also observed that the combination of GNN and LSTM exhibited similar strong performance. For the QoTD dataset, GNN+LSTM achieved a score of 1.51 for both MAR and MFR. On the MicroSS dataset, GNN+LSTM achieved a score of 1.06 for MAR and a score of 1.07 for MFR. On the Train Ticket dataset, GNN+LSTM achieved a score of 1.6 for both MAR and MFR.

Table 5.4: Effectiveness of failure type determination at the node level

Algorithm	QoTD		MicroSS		Train Ticket	
	MAR	MFR	MAR	MFR	MAR	MFR
InstantOps	**1.51**	**1.49**	**1.06**	**1.06**	**1.06**	**1.06**
GNN+LSTM	1.51	1.51	1.06	1.07	1.06	1.06
GNN	1.51	1.51	1.59	1.59	1.06	1.19
GTN	1.67	1.62	1.15	1.24	1.47	1.47
GTN+LSTM	1.89	1.89	1.18	1.11	1.37	1.47
GTN+GRU	1.89	1.56	1.06	1.14	1.90	1.95

RQ3: We showed by localizing the root causes at the node level, the Multi-modal approach performs better on average in terms of MAR and MFR in comparison to other neural network algorithms in its effectiveness of determining failure type at the node level across three datasets: QoTD, MicroSS and Train Ticket.

RQ4: How does our Multi-modal approach perform in terms of root cause node localization compared to existing methods?

In this experiment, we compare our Multi-modal approach with seven algorithms. Microscope and MEPFL are microservice anomaly detection approaches where Microscope collects network and SLO metrics to infer root causes during SLO violations while MEPFL predicts latent errors and faulty microservices by integrating trace logs and injecting faults. TraceAnomaly, an unsupervised approach, learns trace patterns to detect abnormal traces and localise root causes. MonitorRank employs Random Walk, which combines historical and real-time metrics for root cause ranking in service-oriented web architectures. RCSF, designed for enterprise systems, analyses performance logs and dependency models to identify fault propagation sequences. Our Multi-modal approach is designed to fuse features such as resource utilization, events and logs and construct graph neural network based on the interactions among the nodes in microservice and it localize the faulty node that corresponds to system failure.

Table 5.5 provides a quantitative comparison of our Multi-modal approach with seven algorithms such as TraceRCA [20], MicroScope [21], MEPFL (RF) [45], TraceAnomaly[25], Random-Walk [38], and RCSF [37] benchmarked against other methods sourced from [20]. The metrics $A@1$, $A@2$, and $A@3$ are utilized as evaluative standards.

As is evident from the table, our Multi-modal approach demonstrates a high degree of effectiveness. With an $A@1$ score of 0.92, it surpasses the majority of the algorithms in the list and maintains consistent performance across $A@2$ and $A@3$. This consistent high performance across metrics suggests the reliability and robustness of the our Multi-modal approach algorithm.

Algorithms such as `Random Walk` and `RCSF`, although displaying commendable values in $A@2$ and $A@3$, have relatively lower $A@1$ values. This difference could indicate potential variability in their performance across different stages or conditions.

Conversely, `TraceAnomaly` and `MicroScope` consistently perform worse, further delineating the performance gap between these methods and our Multi-modal approach.

Table 5.5: Comparison of root cause localization on faults of different levels on **A**

Algorithm	A@1	A@2	A@3
InstantOps	**0.92**	**0.95**	**0.98**
TraceRCA	0.83	0.93	0.97
MicroScope	0.56	0.62	0.7
MEPFL (RF)	0.94	0.97	0.97
Random Walk	0.51	0.86	0.94
RCSF	0.52	0.86	0.93
TraceAnomaly	0.49	0.59	0.63

RQ4: We showed by localizing the root causes at the node level, our Multi-modal approach outperforms in terms of the evaluative standards A@1, A@2 and A@3 in comparison to other existing methods in root cause localization on faults of different levels on A, demonstrating higher effectiveness than existing methods.

5.2 Multi-Model Fault Classification: Anomaly Detection & Explainable Root Cause Analysis

In this section, we focus on system failures caused by the over-utilization (overload) of resources, incorrect user behaviours and improper system manipulations. We assume that when users initiate login requests via the QR-code login service, abnormalities can arise due to code-related issues, excessive resource consumption, and the system's inability to manage the influx of requests. This can cause the service to malfunction, eventually leading to cascading failures in other dependent services. These failures jeopardize the system's ability to meet both its functional and non-functional requirements. Our experiments are organized around the following research questions:

RQ5: How effective is our approach to predict the failure types?

To assess the effectiveness of our approach, we evaluated our model on a public dataset called MicroSS. The data comprised 419,959 records of metric, log, and trace data. The anomaly fault injection records were used as the ground truth to fairly evaluate our approach. This data was divided into 70% training and 30% testing. For the data, several application metrics were taken into account as features: application resource utilization metrics, log exceptions, availability and latency counts occurrence, response time and error 5xx count occurrence, edge construction that links nodes with traces, and frequency of interactions between nodes with minute-by-minute time windows. Using this in-depth data across data sources, we

aim to improve on our previous approach by going a step further from binary system failure prediction across various datasets by identifying systemic failure types on a particular dataset.

Table 5.6 shows the performance of our approach (GNN-GRU) in comparison with SOTA and other baselines and reveals that our approach achieves the highest F1 score of 90%, with a recall of 91% and a precision of 91%. Considering the complexity of identifying failure types, fusing traces, logs, and metrics, enhancing the prepossessing of the features, and capturing temporal aspects using GRUs, helps in not only identifying whether a system failure occurred but also what type of failure occurred. By fusing traces, logs, and metrics, by capturing temporal aspects using GRUs and by enhancing the preprocessing of the features to identify the type of failure by observing deviations within features in the interactions between services, our approach obtains comparable high precision, recall and F1 scores compared to other neural network algorithms on the MicroSS Dataset.

Table 5.6: Performance comparison of different models

Experiments	**Acc.**	**F1**	**Prec.**	**Rec.**
GNN-GRU	**90.73%**	**90.67%**	**91.23%**	**90.73%**
GNN-GRU w/o BERTOps	84.88%	83.97%	85.42%	84.88%
Diagfusion (SOTA) [41]	75%	83.90%	86.00%	82.90%
Xgboost	89.66%	89.52%	89.71%	89.66%
Xgboost w/o BERTOps	85.95%	85.61%	86.33%	85.95%

RQ5: We showed that classifying fault types of the initial predicted system fault by localizing the root causes at the node level, our Multi-modal approach outperforms in terms of the evaluative metrics with F1 score of 90.67%, with a recall of 90.73% and a precision of 91.23% in comparison to other existing methods in fault type classifications, demonstrating higher effectiveness.

Table 5.7: Sample Explanation for Failure Type Prediction

Failure Type	**Sample Explanation**
Access Denied Failure	At 2021-07-18 16:00:00, the model predicted access denied failure. The most impactful metrics feature was CPU total pct with a gradient distance increase of 0.04%. The most impactful logs feature was exception count with a gradient distance increase of 0.03%. The most impactful traces feature was 5xx errors per minute count with a gradient distance increase of 0.27%. This occurred between requests from redisservice2 to dbservice1.

The sample explanation of a fault, shown in Table 5.7, demonstrates how our approach highlights relevant anomalies and their potential causes, aiding in debugging, decision-making, and providing a strong call to action. Our approach leverages gradient-distance-based expla-

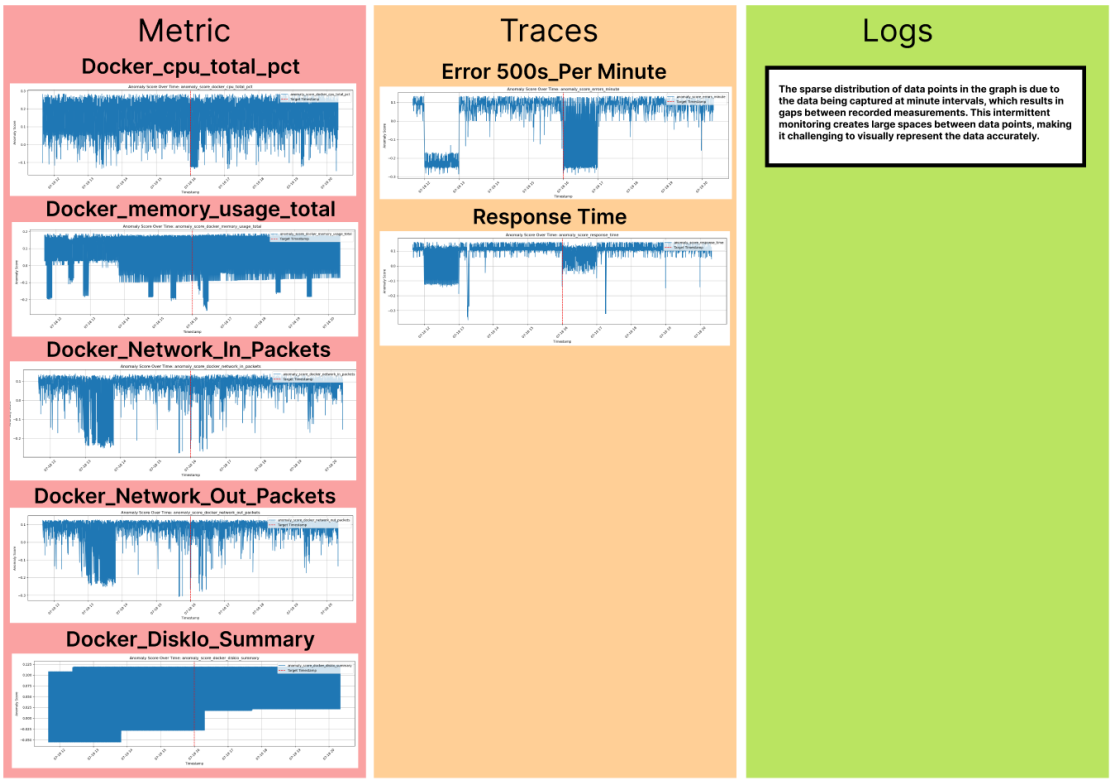

Figure 5.1: System Failure Explaination through Visualization

nations from our GNN-GRU algorithm through feature importance and visualization 5.1 as observed by the negative anomaly score. By analyzing how gradients shift when key features are altered, our method identifies anomalies across logs, metrics, and traces, highlighting the most impactful contributors to classify the failure type for the failure predictions. This allows us to pinpoint the affected services, the direction of the service request, the anomaly type, while the gradient distance quantifies the relative impact of each feature and provides a strong indicator to infer the root cause.

Initial feedback from SREs was positive, with one SRE noting, "The prediction instills confidence as it connects alarms to anomalies in metrics and error levels from logs, while clearly outlining the affected hosts, the type of anomaly, and the specific time-point that supports the hypothesis." This indicates that our method has promise in linking anomalies across three modalities to failure types while offering contextual insights into severity and impact. However, despite this encouraging feedback, we currently lack sufficient feedback to confidently evaluate the overall effectiveness of the explanations. Further validation is needed to determine whether these explanations consistently enhance failure diagnosis and decision-making across diverse failure scenarios. This presents future opportunity to re-evaluate the approach, ensuring it delivers reliable, actionable insights at scale.

5.3 Threats to Validity

In our research, we've identified several threats to validity that warrant careful consideration for Phase 1 and 2 of our study.

For Phase 1: Multi-Modal Binary Fault Prediction

The first threat pertains to the accuracy of failure labeling within our microservices study. Specifically, while examining the 'Quote of the Day' (QoTD) application, we established a performance baseline with JMeter and subsequently annotated failure events using Chaos Mesh for fault injection. This process was supplemented by observations of microservice crashes in OpenShift clusters. Nevertheless, any potential mislabeling or misinterpretation of these events could compromise the integrity of our findings.

Further complicating our validity is the diversity of our data sources. We've utilized three datasets, including two open-source ones, to support the generalizability of our results. However, the variance in size and scope between our experimental data and the real-world complexity of microservice operations could limit the applicability of our conclusions.

Another significant validity threat arises from the granularity of our data collection. By capturing a wide array of metrics—from resource usage to node interactions—on a one-minute interval, we assume that this level of granularity is sufficient for predicting imminent system failures. Yet, there is a risk that more nuanced or granular data could yield different insights, which means our current approach may overlook certain subtleties.

Lastly, the scope of our datasets, which are smaller in comparison to those used in extensive industrial microservice systems, could undermine the scalability of our algorithm. While we believe our algorithm should function effectively even with coarser-grained datasets, the true test of its applicability will come when it is applied to the larger and more complex datasets that we plan to obtain from our clients in future work. This step is crucial for us to validate the efficiency of our model and ensure that it can withstand the demands of a full-scale industrial environment.

For Phase 2: Multi-Model Fault Classification

To cope with internal validity threats like selection bias, we ensured that the dataset was divided using a stratified approach, maintaining a proportional representation of failure types across training and test sets. To mitigate the impact of confounding variables, we plan to incorporate additional system-level features, like network conditions, in future iterations of our model. Regarding measurement error, we carefully preprocessed logs, metrics, and traces to extract meaningful features and used advanced anomaly detection techniques to minimize the influence of noisy or incomplete data. We used GNN-GRU to account for system evolution, ensuring the model can generalize across different system versions. For external validity, our study used only the publicly available MicroSS dataset, which may

limit the generalizability of our findings to other cloud-native microservice architectures. However, this was intentional, as our work extends our initial study on failure prediction, where our key aim this time was to focus on whether we could effectively classify failure types and generate explainable actions. To allow more time for this specific focus, we chose to stick to just one dataset.

5.4 Future Work

Future work will have a clear and continued focus on scenarios where graph-based representations are integral to build upon applications with micro-service architecture rather than broadening to unrelated architectures. Future research into testing the approach in more complex micro-service setups, such as those with high degrees of service-to-service communication or dynamic scaling behaviors. This will help demonstrate the method's adaptability to real-world challenges, such as evolving topologies and resource constraints. The prediction model can be evolved into an online learning system capable of updating its models in real time for a continuous learning approach which would allow for immediate adjustments based on the latest system behavior, thereby enhancing predictive accuracy over time.

The work can be expanded to incorporate more systematic and formal feedback from SREs. While the initial feedback was encouraging, conducting a comprehensive survey across different groups in the SRE team will put-forth a clear and strong foundation to evaluate the robustness and practical utility of our failure explanation system. This broader perspective will enable us to refine the explanation models, making them more user-friendly and effective for real-time operations. In addition, we aim to optimize a customized Large Language model for online failure type detection. This will provide SREs and administrators with faster identification of failure types and root causes, enhancing operational efficiency.

By integrating this optimized detection system into the work-flow, we hope to facilitate quicker responses to critical system issues. To further validate the generalizability of our approach, we plan to expand our evaluation across multiple datasets from varied environments and systems. This will help address our research questions and ensure that the system can reliably identify and explain failures across diverse scenarios. Future work will also involve evaluating the isolation forest anomaly score to enhance data preprocessing on traces, logs, and metrics, while fine-tuning the model's ability to capture temporal features using GRUs to further improve system failure detection and classification.

Chapter 6

Conclusion

In this thesis, we propose an approach which takes in multi-modal data to construct a graph using traces with logs and metric data as node attributes. Through this, we use our multi-modal system failure prediction approach and capture temporal and spatial aspects to precisely and effectively predict failures and determine the root causes of failures at the node level. In addition to using our in-house data set: QoTD, we used two open source datasets: MicroSS and Train-Ticket. In our experimental studies, we demonstrated how our approach can identify and localize the faulty node in microservice, facilitating the root cause analysis of the system failure.

We expand this fusing multi-modal data approach to construct graphs with node attributes, log semantic data and metric anomaly score/ label data to detect anomalies in real time. We continue using our in-house dataset and open source dataset and use of the GNN+GRU model. Through this, our multi-modal data system failure detection approach would explain the root causes of System Failure with relevant anomalous metrics, faulty nodes and trends. Our experimental studies show that our approach would explain the root causes of the system failure: localization of the faulty node, detection of relevant anomalies within the metrics & logs and its trends.

The contributions of this thesis include an improved framework for predicting system failures using logs, metrics, and traces; a novel GNN-GRU model for multi-class classification of failure types; an advanced node localization mechanism for precise fault identification; and a gradient-based explanation method that provides human-readable insights into the model's decisions. Together, these contributions address critical challenges in failure detection and response, demonstrating a robust solution to enhance system reliability and support effective mitigation planning. We show that the use of a GNN to construct topology and interaction among the nodes and incorporating temporal information using the GRU model improves the prediction of system failure and explanation of root causes for system failure to provide a call to action for SREs to act upon them.

In the first phase of our study, which focused on system failure prediction, we achieved F1 scores of 0.96, 0.98, and 0.97 across the datasets, surpassing the state-of-the-art benchmarks. Additionally, we assessed the efficiency of root cause analysis using MAR and MFR, obtaining results that outperformed existing approaches. In the second phase, which concentrated on multi-fault type classification and explainability, our proposed three-fold approach—leveraging observability data, enhanced preprocessing, GNN-GRU integration, and gradient distance—achieved an F1 score of 90%, along with a recall and precision of 91% each. These results highlight the effectiveness of our methodology in accurately classify the fault types of the system failure predictions and present them as of potentially valuable explanations to SREs for remediation actions.

Bibliography

[1] *Explaining the Explainer: A First Theoretical Analysis of LIME*, 2020.

[2] Rafiul Ahad, Eric Chan, and Adriano Santos. Toward autonomic cloud: Automatic anomaly detection and resolution. In *2015 International conference on cloud and autonomic computing*, pages 200–203. IEEE, 2015.

[3] David N Blank-Edelman. *Seeking SRE: conversations about running production systems at scale*. " O'Reilly Media, Inc.", 2018.

[4] Vilde Christiansen, Moutaz Haddara, and Marius Langseth. Factors affecting cloud erp adoption decisions in organizations. *Procedia Computer Science*, 196:255–262, 2022.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[6] S. Frank, G. Lin, X. Jin, R. Singla, A. Farthing, and J. Granderson. A performance evaluation framework for building fault detection and diagnosis algorithms. *Energy and Buildings*, 192:84–92, 2019.

[7] Ying Fu, Meng Yan, Jian Xu, Jianguo Li, Zhongxin Liu, Xiaohong Zhang, and Dan Yang. Investigating and improving log parsing in practice. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1566–1577, 2022.

[8] Sahil Garg, Kuljeet Kaur, Neeraj Kumar, Shalini Batra, and Mohammad S Obaidat. Hyclass: Hybrid classification model for anomaly detection in cloud environment. pages 1–7, 2018.

[9] Sahil Garg, Kuljeet Kaur, Neeraj Kumar, Georges Kaddoum, Albert Y Zomaya, and Rajiv Ranjan. A hybrid deep learning-based model for anomaly detection in cloud datacenter networks. *IEEE Transactions on Network and Service Management*, 16(3):924–935, 2019.

[10] L Girish and Sridhar KN Rao. Anomaly detection in cloud environment using artificial intelligence techniques. *Computing*, pages 1–14, 2021.

[11] Pranjal Gupta, Harshit Kumar, Debanjana Kar, Karan Bhukar, Pooja Aggarwal, and Prateeti Mohapatra. Learning representations on logs for aiops. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*, pages 155–166. IEEE, 2023.

[12] Tanja Hagemann and Katerina Katsarou. A systematic review on anomaly detection for cloud computing environments. In *2020 3rd Artificial Intelligence and Cloud Computing Conference*, pages 83–96, 2020.

[13] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*, pages 33–40. IEEE, 2017.

[14] Jordan Hochenbaum, Owen S Vallis, and Arun Kejariwal. Automatic anomaly detection in the cloud via statistical learning. *arXiv preprint arXiv:1704.07706*, 2017.

[15] IBM. What is kubernetes? `https://www.ibm.com/cloud/learn/kubernetes`, 2022.

[16] Mohammad S Islam, William Pourmajidi, Lei Zhang, John Steinbacher, Tony Erwin, and Andriy Miranskyy. Anomaly detection in a large-scale cloud platform. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 150–159. IEEE, 2021.

[17] Abul Khayer, Nusrat Jahan, Md Nahin Hossain, and Md Yahin Hossain. The adoption of cloud computing in small and medium enterprises: a developing country perspective. *VINE Journal of Information and Knowledge Management Systems*, 51(1):64–91, 2021.

[18] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R Lyu. Eadro: An end-to-end troubleshooting framework for microservices on multi-source data. *arXiv preprint arXiv:2302.05092*, 2023.

[19] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, Yongqiang Yang, and Michael R Lyu. Heterogeneous anomaly detection for software systems via semi-supervised cross-modal attention. *arXiv preprint arXiv:2302.06914*, 2023.

[20] Zeyan Li, Junjie Chen, Rui Jiao, Nengwen Zhao, Zhijun Wang, Shuwei Zhang, Yanjun Wu, Long Jiang, Leiqin Yan, Zikai Wang, et al. Practical root cause localization for microservice systems via trace analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pages 1–10. IEEE, 2021.

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

[22] Marin Litoiu, Ian Watts, and Joe Wigglesworth. The 13th cascon workshop on cloud computing: engineering aiops. In *Proceedings of the 31st Annual International Conference on Computer Science and Software Engineering*, pages 280–281, 2021.

[23] Dapeng Liu, Youjian Zhao, Haowen Xu, Yongqian Sun, Dan Pei, Jiao Luo, Xiaowei Jing, and Mei Feng. Opprentice: Towards practical and automatic anomaly detection through machine learning. In *Proceedings of the 2015 internet measurement conference*, pages 211–224, 2015.

[24] F.T. Liu, K.M. Ting, and Z.H. Zhou. Isolation forest. In *2008 8th IEEE International Conference on Data Mining*, pages 413–422, Pisa, Italy, Dec 2008.

[25] Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, et al. Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 48–58. IEEE, 2020.

[26] Meng Ma, Weilan Lin, Disheng Pan, and Ping Wang. Self-adaptive root cause diagnosis for large-scale microservice architecture. *IEEE Transactions on Services Computing*, 15(3):1399–1410, 2020.

[27] Minghua Ma, Shenglin Zhang, Junjie Chen, Jim Xu, Haozhe Li, Yongliang Lin, Xiaohui Nie, Bo Zhou, Yong Wang, and Dan Pei. {Jump-Starting} multivariate time series anomaly detection for online service systems. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 413–426, 2021.

[28] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *IJCAI*, volume 19, pages 4739–4745, 2019.

[29] Joydeep Mukherjee, Alexandru Baluta, Marin Litoiu, and Diwakar Krishnamurthy. Rad: Detecting performance anomalies in cloud-based web services. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, pages 493–501. IEEE, 2020.

[30] N Pandeeswari and Ganesh Kumar. Anomaly detection system in cloud environment using fuzzy clustering based ann. *Mobile Networks and Applications*, 21:494–505, 2016.

[31] Daniel Pop. Machine learning and cloud computing: Survey of distributed and saas solutions. *arXiv preprint arXiv:1603.08767*, 2016.

[32] Rui Ren, Yang Wang, Fengrui Liu, Zhenyu Li, and Gaogang Xie. Triple: the interpretable deep learning anomaly detection framework based on trace-metric-log of microservice. In *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2023.

[33] Y. Rouf, J. Mukherjee, and M. Litoiu. Towards a robust on-line performance model identification for change impact prediction. In *2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 68–78, Melbourne, Australia, 2023.

[34] J. Simonsson, L. Zhang, B. Morin, B. Baudry, and M. Monperrus. Observability and chaos engineering on system calls for containerized applications in docker. *Future Generation Computer Systems*, 122:117–129, 2021.

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

[36] Arthur Vervaet. Monilog: An automated log-based anomaly detection system for cloud computing infrastructures. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2739–2743. IEEE, 2021.

[37] Kui Wang, Carol Fung, Chao Ding, Polo Pei, Shaohan Huang, Zhongzhi Luan, and Depei Qian. A methodology for root-cause analysis in component based systems. In *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*, pages 243–248. IEEE, 2015.

[38] Ping Wang, Jingmin Xu, Meng Ma, Weilan Lin, Disheng Pan, Yuan Wang, and Pengfei Chen. Cloudranger: Root cause identification for cloud native systems. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 492–502. IEEE, 2018.

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

[40] Hongwei Zhang, Yuanqing Xia, Tijin Yan, and Guiyang Liu. Unsupervised anomaly detection in multivariate time series through transformer-based variational autoencoder. In *2021 33rd Chinese Control and Decision Conference (CCDC)*, pages 281–286. IEEE, 2021.

[41] Shenglin Zhang, Pengxiang Jin, Zihan Lin, Yongqian Sun, Bicheng Zhang, Sibo Xia, Zhengdan Li, Zhenyu Zhong, Minghua Ma, Wa Jin, et al. Robust failure diagnosis of microservice system through multimodal data. *arXiv preprint arXiv:2302.10512*, 2023.

[42] Xu Zhang, Qingwei Lin, Yong Xu, Si Qin, Hongyu Zhang, Bo Qiao, Yingnong Dang, Xinsheng Yang, Qian Cheng, Murali Chintalapati, et al. Cross-dataset time series anomaly detection for cloud systems. In *USENIX Annual Technical Conference*, pages 1063–1076, 2019.

[43] Chenyu Zhao, Minghua Ma, Zhenyu Zhong, Shenglin Zhang, Zhiyuan Tan, Xiao Xiong, LuLu Yu, Jiayi Feng, Yongqian Sun, Yuzhi Zhang, et al. Robust multimodal failure detection for microservice systems. *arXiv preprint arXiv:2305.18985*, 2023.

[44] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Transactions on Software Engineering*, 47(2):243–260, 2018.

[45] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 683–694, 2019.