InstantOps: A Joint Approach to System Failure Prediction and Root Cause Identification in Microservices Cloud-Native Applications

Raphael Rouf
raphaelr@my.yorku.ca
York University
Toronto, Ontario, Canada

Mohammadreza Rasolroveicy
roveicy@ibm.com
IBM Canada Lab
Markham, Ontario, Canada

Marin Litoiu
mlitoiu@yorku.ca
York University
Toronto, Ontario, Canada

Seema Nagar
senagar3@in.ibm.com
IBM Research
Bangalore, India

Prateeti Mohapatra
pramoh01@in.ibm.com
IBM Research
Bangalore, India

Pranjal Gupta
Pranjal.Gupta2@ibm.com
IBM Research
Bangalore, India

Ian Watts
ifwatts@ca.ibm.com
IBM Canada Lab
Markham, Ontario, Canada

ABSTRACT

As microservice and cloud computing operations increasingly adopt automation, the importance of models for fostering resilient and efficient adaptive architectures becomes paramount. This paper presents `InstantOps`, a novel approach to system failure prediction and root cause analysis 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, microservices events(e.g. Image Pull Back Off, PVC Pending), 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.

To evaluate the efficiency of `InstantOps`, we utilized in-house data from the open-source application Quote of the Day (QoTD) as well as two publicly available datasets, MicroSS and Train Ticket. The F1 scores obtained in predicting the system failures from these data sets were 0.96, 0.98, and 0.97, respectively, beating the state-of-the-art. Additionally, we further evaluated the efficiency of root cause analysis using MAR and MFR. These results also outperform the state of the art.

CCS CONCEPTS

• **Computing methodologies** → **Distributed computing methodologies**; **Feature selection**; *Regularization*; • **Information systems** → **Clustering**; • **Networks** → Network performance analysis.

KEYWORDS

Cloud Computing, Anomaly Prediction, Resource Overload, Machine Learning, Root Cause Analysis

ACM Reference Format:
Raphael Rouf, Mohammadreza Rasolroveicy, Marin Litoiu, Seema Nagar, Prateeti Mohapatra, Pranjal Gupta, and Ian Watts. 2024. InstantOps: A Joint Approach to System Failure Prediction and Root Cause Identification in Microservices Cloud-Native Applications. In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3629526.3645047

1 INTRODUCTION

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 [2, 13]. 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 [7, 35].

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 methods, clustering, and rule-based systems have been employed for this purpose [1, 5, 9, 11, 12, 24, 25, 28]. 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 [6, 7, 18, 26].

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

Traces in [16, 32, 38]: 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 [4, 8, 23, 29]: 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 [19, 21, 27, 33]: 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 [14, 15, 36] 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.

In this paper, through the introduction of `InstantOps`, we augment the existing state-of-the-art research [14–16, 34, 36] by fusing multimodal data source including logs, traces, resource metrics, and microservices events (e.g. Image Pull Back Off, PVC pending) in a time series format to predict system failures at the service (node) level. In this paper, we characterize a 'node' as an individual service within the micro-service architecture. Specifically, our objective is to predict, identify, and localize the nodes in the system that might be responsible for imminent system failures. By pinpointing and localizing the problematic node within the microservice architecture, we can strategize appropriate remediation measures, such as node scaling or resource configuration.

We stitch the multi-modal data for a time window in a novel fashion, where traces act as a thread. We serialize logs, traces, and metrics for each time window. In this process, we intertwine traces as the stitching thread, serializing logs, and metrics for each time window. This serialized data aids in constructing a dynamic dependency graph that delineates the interconnections among services—wherein nodes represent services and their interactions are depicted as edges.

Moreover, we enrich this dependency graph by merging both logs and resource metrics as attributes assigned to its nodes. This augmentation enhances the representation of the system's spatial features at specific points in time. To analyze and comprehend these spatial features, we employ Graph Neural Networks (GNN), recognizing that a system's failure isn't a single-point occurrence but a lifecycle that gradually progresses toward a system crash. Understanding the temporal evolution of these spatial features is imperative. To address this, we utilize Gated Recurrent Unit (GRU) models, acknowledging the importance of capturing the system's changing dynamics over time.

Our focus lies not only in training a multi-modal GNN-GRU model for predicting system failures but also in utilizing the learned node scores to localize the root cause. This approach allows us to repurpose the trained model for the specific task of root cause analysis. By leveraging the insights gained from failure prediction, particularly the node scores, we aim to guide the root cause analysis process. We believe that the predictive learning captured in failure prediction can significantly aid in root cause localization, thereby aligning our methodologies for a more comprehensive understanding of system behavior and failure analysis.

The contributions of this paper can be summarized as follows:

- Novel System Failure Prediction: We introduce a unique approach to predict system failures by fusing multimodal observability data in a novel fashion wherein we overlay the logs and metrics data as node attributes in the graph constructed using the traces.
- We propose to capture both temporal and spatial aspects in effective failure prediction.
- Innovative Root Cause Analysis Method: We use the multi-modal system failure prediction model for pinpointing the root causes of failures at the node level.
- Comprehensive Experimental Study: We conduct an extensive experimental analysis to evaluate the efficiency of our model in terms of both prediction accuracy and root cause analysis. This assessment is based on two open-source and one proprietary dataset derived from open-source microservice systems.

The remainder of the paper is organized as follows: Section 2 describes the motivation behind our experiments and study. In Section 3, we discuss recent related works in failure predictions and anomaly detection in microservices, specifically focusing on the use of multimodal datasets. Section 4 details our methodology and elaborates on our GNN-GRU based algorithm for detecting failures in microservices. Section 5 delves into our approach to temporal failure prediction and root cause analysis including an explanation of the algorithms. In Section 6, we discuss the evaluation of our experimental study. Section 7 addresses the potential threats to validity. Finally, in Section 8, we conclude our findings and present potential avenues for future research.

2 MOTIVATION:

The analysis of system failures necessitates a thorough examination of diverse data modalities. Prior research indicates that relying solely on single-modal data is inadequate for capturing the intricacies of failure patterns, especially in microservice applications. In this paper, we utilize three distinct datasets: Anafusion's MicroSS [36], TraceRCA's Train Ticket [16], and an in-house dataset for the "Quote of the Day (QOTD)" open-source application. These datasets comprise different combinations of logs, traces, events, and metrics, depending on the specific experiment. For instance, the QOTD dataset incorporates logs, traces, events, and resource metrics, while the Train Ticket dataset emphasizes traces and metrics. To ensure a comprehensive evaluation, all datasets encompass records of every failure injection. Our primary objective is to determine the precision of system failure predictions when leveraging supervised learning. Distinctly from previous models, we incorporate edges into the system to systematically assess the impact on specific nodes. Furthermore, in this paper, the edges are constructed by the communications among the services in the microservice application.

The analysis of these interactions offers critical insights into which nodes require interventions, such as scaling, restarting, or remediation of various failures.

For system failure prediction, multiple neural network models are tested to ascertain the most efficient predictive methodology. The effectiveness of `InstantOps` is subsequently quantified using metrics including accuracy, precision, recall, and the F1 Score. To benchmark its performance, `InstantOps` is compared against two established methods, Anafusion and TraceRCA.

3 RELATED WORKS

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.

Zhao et al. [36] 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 moment. 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. In Anafusion, the authors model one node, in this paper, we model the entire microservice application. Furthermore, our model is built with multiple data sources such as events from each microservices.

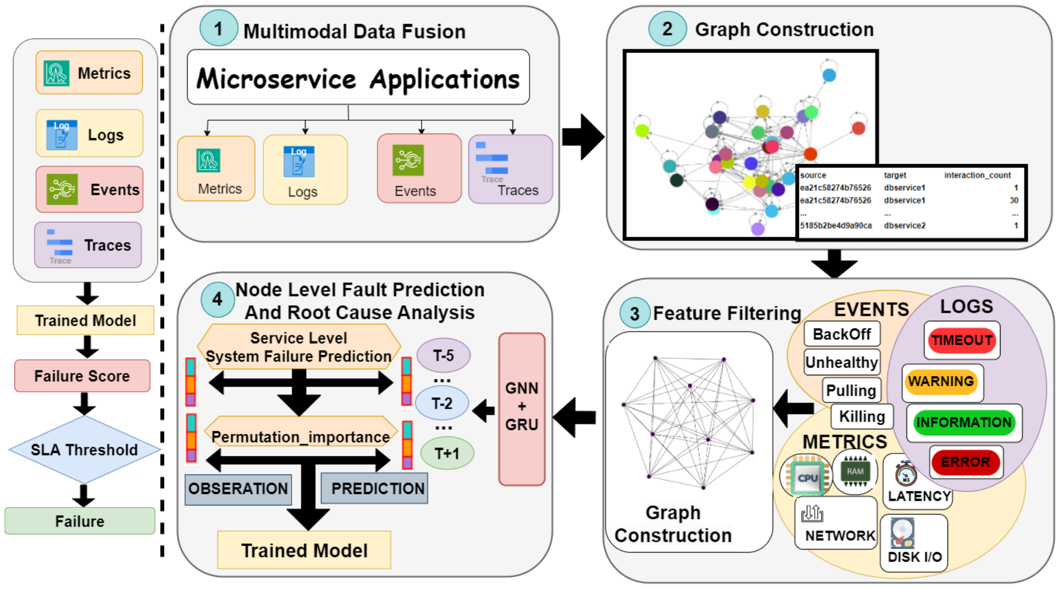
Zhang et al. [34] proposed an alternative method known as DiagFusion. This approach leverages multimodal data to enhance fault detection by employing advanced embedding techniques fastText and data augmentation. It constructs a dependency graph and employs a graph neural network to pinpoint the root cause and identify the type of failure.

Li et al. [16] 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. [38] 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. [15] 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. [14] 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, compelling 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. Research works similar to Hades and Eardo aim to detect system anomalies and failure, our intention is not to detect system anomalies but to predict the anomalies at the next minute based on the last 5 minute time window.

4 METHODOLOGY

In this section, we provide a systematic approach designed to predict system failures and facilitate subsequent root cause analysis using Graph Neural Networks (GNNs) in a Microservice application. We initiate with the standardization of features and then proceed



source	target	interaction_count
ea21c58274b76526	dbservice1	1
ea21c58274b76526	dbservice1	30
...	...	...
5185b2be4d9a90ca	dbservice2	1

Figure 1: InstantOps Offline Training Model

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'.

When an interaction between or services is observed for the first time within a specified duration, we create an edge between the two corresponding nodes in GNN, and the weight of the edges is 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 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 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. Additionally, the model is extended to incorporate edge features, 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. 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.

As shown in Figure 1, the workflow of `InstantOps` is divided into multiple stages:

First, we preprocess and serialize the multi-modal data-set within a controlled time-frame. For each data source, we define time windows with the same size. The sequence of the time windows defines the stream of graphs for our model. Next, we construct the Graph Neural Networks using edges and nodes. For each time window, we add the features to the nodes in the graph. The layers of the GNN 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.

We further elaborate on our steps below:

4.1 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 microservices, and E is a set of edges representing interactions between services. The adjacency matrix $A \in \mathbb{R}^{n\times n}$ encapsulates the number of interactions, where $A_{ij} = 1$ if an interaction exists between services represented by nodes v_i and v_j, and 0 otherwise. In other paper, authors define Edges $e_{ij} \in E$ can also be defined based on metrics between nodes:

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

where f represents a function determining interactions between nodes based on logs and metrics.

4.2 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 our previous work, BERTOps [8], which utilizes the Drain [10] to extract structured information from raw log data by clustering log lines into templates based on their structural similarity and BERT [3] 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 BertOps aims to classify a log line, in this paper, we classify each node and extract vital features from the logs, such as the number of errors a node receives within the 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 features: timestamp, node id, number of errors. These features will be added to the nodes for a corresponding time window.

4.3 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 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.

4.4 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.

5 TEMPORAL FAILURE PREDICTION AND ROOT CAUSE ANALYSIS

In this section, we discuss the approaches for failure prediction and root cause analysis.

5.1 Temporal Failure Prediction

In our approach, we integrate the power of Graph Neural Networks (GNN) with Gated Recurrent Units (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 Gated Recurrent Unit (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:

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \quad \text{(Reset gate)}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \quad \text{(Update gate)}$$

$$\hat{h}_t = \tanh(W x_t + U(r_t \odot h_{t-1}) + b) \quad \text{(New potential hidden state)}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t \quad \text{(Actual hidden state update)}$$

For each sequence and its respective time step, the GNN first imbibes 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.

Algorithm 1 Temporal Failure Prediction using GNN with GRU

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

Table 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				✓

Figure 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 Graph Neural Network (GNN) with a Gated Recurrent Unit (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 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.

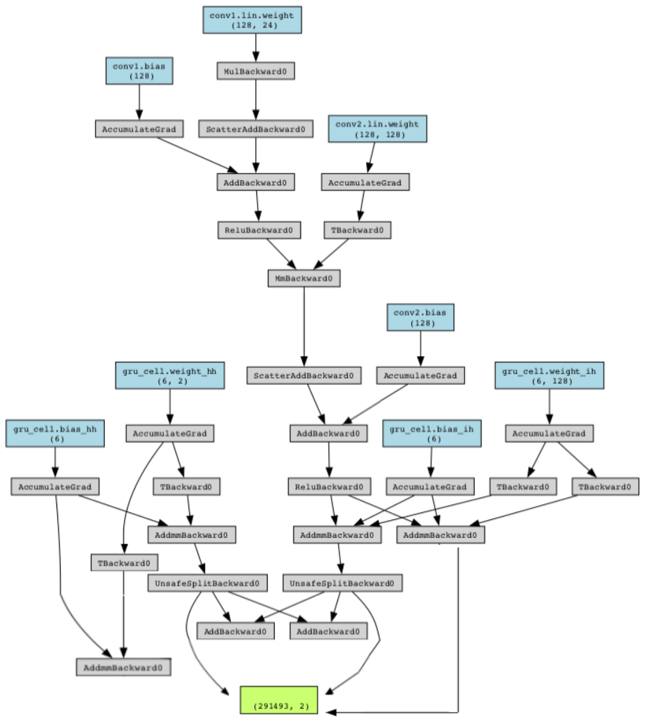


Figure 2: The Diagram for GNN+GRU Neural Network Model

5.2 Root Cause Analysis

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 InstantOps root cause localization processes at the node level. Specifically, we utilize Mean First Rank (MFR) and Mean Average Rank (MAR) as our primary metrics.

Mean First Rank (MFR): 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.

Mean Average Rank (MAR): 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 DNS. 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.

6 EVALUATION

Here, we present the data for evaluation, evaluate our methods for failure prediction and root cause analysis, and present the performance of the models on various datasets and baselines.

6.1 Datasets

In this paper, 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 [37] 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., [16] to enable comparative analysis with their findings. Three types of faults were introduced [16]: 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 [36] and [34].

The QoTD open-source application [3] consists of eight distinct microservices. We deployed QoTD on IBM OpenShift clusters and introduced a variety of faults using Chaos-Mesh [4]. These faults include disruptions in CPU, memory, and DNS.

For monitoring purposes, we employed Instana [5], which provided data points such as API response times, error codes, and various resource utilization metrics: CPU, memory, disk, and network. We utilized LogDNA [6] 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>'.

6.2 Evaluation Metrics

6.2.1 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

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

6.2.2 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:

[1] [2]

[3] https://gitlab.com/quote-of-the-day/quote-of-the-day/-/tree/master

[4] https://chaos-mesh.org/

[5] https://www.ibm.com/products/instana

[6] 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.

Mean First Rank (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.

Mean Average Rank (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.

6.3 Results and Discussions

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

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

6.3.1 RQ1: How efficient is `InstantOps` *for system failure prediction compared to other neural network algorithms?* To assess the efficiency of `InstantOps`, three distinct datasets were utilized, as shown in Table 2. The first, MicroSS, is an open-source dataset comprising 419,959 data. This data was divided into training and testing sets with 84,026 and 335,933, respectively. The Train Ticket open-source dataset encompasses 24,492 data, which are further divided into 19,337 for testing and 5,085 for training. Lastly, our in-house dataset includes 450,000 data, with a distribution of 360,000 for testing and 90,000 for training.

Table 2: Data Overview for Applications

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

For each dataset, several applications metrics were considered which we defined as features: application resource utilization, logs detailing errors for each node within specific timeframes (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 event metrics, capturing 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 adhered 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 timeframe (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 3. These models, previously utilized by other researchers [34, 36], were examined to compare the effectiveness of `InstantOps`. Notably, InstantOps 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 `InstantOps` 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 3: 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, `InstantOps` obtained higher precision, recall and F1 scores in comparison to other neural network algorithms on three datasets: QoTD, MicroSS and Train Ticket.

6.3.2 *RQ2: How does the efficiency of* `InstantOps` *for failure prediction compare to state-of-the-art methods?*

Table 4 illustrates a comparison study between `InstantOps`, JLT and Anafusion to establish a multimodal baseline.

The JLT method aggregates the results from JumpStarter [22], LogAnomaly [23], and Traceanomaly [20]. 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, Zhao et [36]. proposed Anafusion, an unsupervised failure detection technique that integrates multimodal data for microservices. It uses a graph neural network to learn the correlations in the heterogeneous multimodal dataset. For the same MicroSS dataset, Anafusion achieved an F1 score of 0.85, a recall of 0.94, and a precision of 0.79.

In `InstantOps`, we have further refined the Anafusion model by constructing a graph based on traces, which displays interactions among the nodes in real time. `InstantOps` employs both GNN and GRU to account for the system's temporal aspects. Our experimental results indicate that `InstantOps`, 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 4: The Average Percentage Among Precision, Recall, and F1-Score of Different Approaches on MicroSS Dataset

	Modality			MicroSS Dataset		
Approach	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, `InstantOps` achieves higher precision, recall and F1 score than the state-of-the-art methods.

6.3.3 *RQ3: How does* `InstantOps` *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.

As can be seen from the table, `InstantOps` achieves an MFR score of 1.49 and a MAR score of 1.51 on the QoTD dataset. On the MicroSS dataset, `InstantOps` achieves a score of 1.51 for both MFR and MAR. On the Train Ticket dataset, `InstantOps` 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: 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 `InstantOps` approach performs better in terms of MAR and MFR in comparison to other neural network algorithms in its effectiveness of determining failure type at the node level on three datasets: QoTD, MicroSS and Train Ticket.

6.3.4 *RQ4: How does* `InstantOps` *perform in terms of root cause node localization compared to existing methods?*

In this experiment, we compare `InstantOps` 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. `InstantOps` is designed to fuse features such as resource utilization, events and logs and construct graph neural network based on the interations among the nodes in microservice. and it localize the faulty node that corresponds to system failure.

Table 6 provides a quantitative comparison of `InstantOps` with seven algorithms such as TraceRCA [16], MicroScope [17], MEPFL (RF) [38], TraceAnomaly[20], Random-Walk [31], and RCSF [30] benchmarked against other methods sourced from [16]. The metrics $A@1$, $A@2$, and $A@3$ are utilized as evaluative standards.

As is evident from the table, `InstantOps` 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 `InstantOps` 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 `InstantOps`.

Table 6: 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, `InstantOps` 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.

7 THREATS TO VALIDITY

In our research, we've identified several threats to validity that warrant careful consideration. 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.

8 CONCLUSION

In this work, we proposed `InstantOps`, an approach that takes in multi-modal data to construct a graph using traces with logs and metric data as node attributes. Through this, we use our multimodal 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 have shown how `InstantOps` can identify and localize the faulty node in microservice which can facilitate the root cause analysis of the system failure. We believe that the use of a Graph Neural Network to construct topology and interaction among the nodes and incorporating temporal information using the GRU model improves the prediction of system failure. We aim to evolve `InstantOps` into an online learning system capable of updating its models in real time. This continuous learning approach would allow for immediate adjustments based on the latest system behavior, thereby enhancing predictive accuracy over time.

9 ACKNOWLEDGMENT

We would like to thank IBM Center for Advanced Studies and Ontario ORF-RE for their support.

REFERENCES

[1] Rafiul Ahad, Eric Chan, and Adriano Santos. 2015. Toward autonomic cloud: Automatic anomaly detection and resolution. In *2015 International Conference on Cloud and Autonomic Computing*. IEEE, 200–203.

[2] Vilde Christiansen, Moutaz Haddara, and Marius Langseth. 2022. Factors affecting cloud ERP adoption decisions in organizations. *Procedia Computer Science* 196 (2022), 255–262.

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

[4] Ying Fu, Meng Yan, Jian Xu, Jianguo Li, Zhongxin Liu, Xiaohong Zhang, and Dan Yang. 2022. 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*. 1566–1577.

[5] Sahil Garg, Kuljeet Kaur, Neeraj Kumar, Shalini Batra, and Mohammad S Obaidat. 2018. HyClass: Hybrid classification model for anomaly detection in cloud environment. (2018).

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

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

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

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

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

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

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

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

[14] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R Lyu. 2023. Eadro: An End-to-End Troubleshooting Framework for Microservices on Multi-Source Data. *arXiv preprint arXiv:2302.05092* (2023).

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

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

[17] JinJin Lin, Pengfei Chen, and Zibin Zheng. 2018. 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*, Vol. 16. Springer, 3–20.

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

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

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

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

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

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

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

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

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

[27] Y Rouf, J Mukherjee, and M Litoiu. 2023. 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)*. Melbourne, Australia, 68–78. https://doi.org/10.1109/SEAMS59076.2023.00018

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

[29] Arthur Vervaet. 2021. MoniLog: An Automated Log-Based Anomaly Detection System for Cloud Computing Infrastructures. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2739–2743.

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

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

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

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

[34] Shenglin Zhang, Pengxiang Jin, Zihan Lin, Yongqian Sun, Bicheng Zhang, Sibo Xia, Zhengdan Li, Zhenyu Zhong, Minghua Ma, Wa Jin, et al. 2023. Robust Failure Diagnosis of Microservice System through Multimodal Data. *arXiv preprint arXiv:2302.10512* (2023).

[35] Xu Zhang, Qingwei Lin, Yong Xu, Si Qin, Hongyu Zhang, Bo Qiao, Yingnong Dang, Xinsheng Yang, Qian Cheng, Murali Chintalapati, et al. 2019. Cross-dataset Time Series Anomaly Detection for Cloud Systems. In *USENIX Annual Technical Conference*. 1063–1076.

[36] Chenyu Zhao, Minghua Ma, Zhenyu Zhong, Shenglin Zhang, Zhiyuan Tan, Xiao Xiong, LuLu Yu, Jiayi Feng, Yongqian Sun, Yuzhi Zhang, et al. 2023. Robust Multi-Modal Failure Detection for Microservice Systems. In *arXiv preprint arXiv:2305.18985*.

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

[38] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. 2019. 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*. 683–694.