A GRAPH-BASED SLA BREACH PREDICTION FRAMEWORK AT THE SERVICE LEVEL IN NEURAL INFERENCE

SARA FEHRESTI

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

MAY, 2025

Abstract

Cloud computing, as the backbone of modern adaptive software architectures, has revolutionized data storage and processing, driven by the power and flexibility of microservices. Despite their advantages in fault isolation and flexible deployment, microservices often experience unpredictable latency spikes, leading to costly Service Level Agreement (SLA) violations.

This thesis introduces a multiscale time-spectrum framework, called GRASP (Graph-based SLA Breach Prediction). It leverages time-series data, sequence processing, and graph-based modeling to proactively detect performance anomalies and predict SLA breaches in microservice-based systems within upcoming time windows. In our framework, raw data is transformed into graph representations and fed into deep learning models to capture both topological and temporal characteristics. By combining graph analysis with sequential modeling, our dual approach not only identifies critical service dependencies but also pinpoints potential end-to-end bottlenecks.

Evaluations on microservice datasets demonstrate its superiority over baseline methods in early warnings, forecasting breaches, and localizing root causes at the service level, underscoring its potential to enhance the reliability and efficiency of microservice-based applications in cloud environments.

Acknowledgment

I would like to express my deepest gratitude to my esteemed supervisor, Dr. Marin Litoiu, whose generosity and kindness granted me the life-changing opportunity to pursue my second master's degree at York University. I will never forget his unwavering support, compassion, and invaluable guidance, which have been instrumental in my academic progress and personal growth. Without his help, achieving my goals would have been impossible, and I am certain that the knowledge and experience I have gained will continue to enrich my life for years to come.

I am also profoundly thankful to my parents for their unceasing support and encouragement, which have been the backbone of my journey. My heartfelt appreciation goes out to my sister and my two brothers, whose warm presence and steadfast belief in me have offered immense motivation throughout this challenging path.

Last but not least, the completion of this journey would not have been possible without my anchor of life, pillar of support, best friend, and soulmate, Sina. His love, patience, and unwavering faith in my potential have been the driving force that kept me moving forward every step of the way.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Today, microservices architecture is not merely a new trend in software engineering but rather an intelligent response to the complexity and dynamism of large-scale systems. In this approach, instead of integrating all functionalities into a centralized structure, each small and autonomous service focuses solely on a specific task and communicates with other services through lightweight interfaces [1]. This level of independence enables rapid feature deployment, heterogeneous scalability, and a significant reduction in the risk of errors [2, 3, 4]. At the same time, microservices introduce new challenges in monitoring and ensuring efficiency, as tracking the performance of dozens or even hundreds of distributed services in cloud-based environments requires specialized tools and techniques to promptly identify performance bottlenecks and potential points of failure [5, 6]. Recent research efforts aim to improve the microservices experience further by introducing new patterns for monitoring, data-driven troubleshooting, and dynamic resource allocation [7, 8, 9]. Latency distribution contains enough distinguishing features to differentiate between normal and anomalous traces to aid in downstream tasks such as anomaly detection and root cause localization [2].

The Challenge of Service Level Agreements (SLAs). Adhering to Service Level Agreements (SLAs) in microservice architectures is crucial [10, 11]. These agreements establish performance and availability guarantees that directly impact system reliability and user experience. Violating SLA also cause financial penalties. [12]. SLAs define performance and availability expectations, often measured through metrics such as latency and resource utilization. In microservice-based systems, even minor issues in one service can cascade across the entire application, leading to SLA violations. For instance, a bottleneck in one microservice can cause delays that propagate through downstream services, potentially breaching latency thresholds and impacting user experience [13, 14].

In multi-tenant cloud settings, where workloads fluctuate unpredictably, the need for proactive, service-specific prediction mechanisms becomes even more critical [15, 16]. Traditional monitoring, such as threshold-based alerts or static rules, fails to model the intricate, evolving interactions among microservices, often reacting only after violations occur [17, 18]. Among various performance metrics, latency stands out as the most critical factor contributing to SLA breaches because it directly affects user experience and service responsiveness [17, 18]. While metrics such as availability, error rate, throughput, and resource utilization are essential, SLAs are frequently structured around response time guarantees rather than just system uptime [17]. A service can be available 99.99% of the time, yet still fail its SLA if requests exceed the agreed latency threshold.

Advances in Predictive Techniques. Recent research has advanced performance monitoring through end-to-end tracing [5, 19, 20], root cause identification [2, 21, 22, 23], and dynamic resource allocation [24, 25, 26, 27]. Latency, as a key indicator of service responsiveness, has been a focal point of these efforts [28, 29, 30]. However, conventional latency-prediction models like queueing theory or simple regression, often falter outside their observed conditions, lacking the granularity to identify specific at risk services [9, 28]. Modern machine learning techniques have opened new avenues, blending topological and temporal modeling for robust anomaly detection [4, 14, 31, 32]. Graph Neural Networks (GNNs) excel at capturing structural dependencies, representing microservices as nodes and their interactions as edges to reveal bottlenecks and propagation effects [33, 28]. Temporal models, such as Temporal Convolutional Networks (TCNs) and Gated Recurrent Units (GRUs), complement this by tracking short-term spikes and long-term trends, respectively [34, 1, 35].

1.1 Our Contributions

- **Graph-Based Framework for SLA Violation Prediction.**By modeling microservice traces as directed graphs (where nodes represent services and edges carry information such as latency and resource consumption), GRASP captures the dynamic interactions and critical dependencies among microservices. This enables more accurate prediction of potential SLA violations.

- **Two-Stage Temporal Modeling.** This approach combines Temporal Convolutional Networks (TCN) for detecting short-term spikes and Bidirectional Gated Recurrent Units (Bi-GRU) for analyzing long-term trends. This hybrid technique covers a wide range of anoma-

lies and significantly improves prediction accuracy.

- **Service-Level SLA Violation Detection and Root Cause Localization.** Utilizing graph attention mechanisms and interpretable layers, the proposed framework can pinpoint which microservices are on the verge of violating SLAs. Consequently, resources can be allocated optimally, and problems can be addressed more efficiently.

- **Scalable Trace-Based Pipeline.** By independently processing microservice traces, this system is well-suited for deployment in large-scale cloud environments. Combining node-level features (CPU usage, memory, etc.) with edge-level features (data flow patterns, latency, etc.) provides a comprehensive view of the overall performance.

  Evaluations on the TrainTicket microservice dataset show that the TCN-BiGRU architecture outperforms purely sequence-based methods (e.g., GRU and LSTM) and graph-only approaches in both performance and stability when detecting SLA violations. Under various workload scenarios, it consistently yields reliable results.

1.2 Thesis Structure

The remainder of this document is structured as follows. Chapter 2 provides a comprehensive review of existing research on microservice performance monitoring, SLA breach prediction, and graph-based modeling techniques. It highlights the limitations of current approaches. Chapter 3 details the GRASP framework, describing the graph construction processes, sequential modeling architecture, and optimization strategies plus root cause localization in which the model predicts the breach services. Chapter 4 evaluates the proposed method on a microservice-based dataset, highlighting its ability to forecast SLA breaches. Finally, Chapter 5 summarizes the key findings, discusses limitations such as scalability and data imbalance, and outlines potential avenues for future research in graph-based and temporal modeling for microservice performance monitoring.

Chapter 2

Related Works

The shift to microservice architectures has heightened the complexity of ensuring performance and upholding Service Level Agreements (SLAs) due to their distributed, interdependent nature [3, 1, 6]. Modern cloud-native systems require sophisticated methods to predict SLA breaches and identify affected services proactively. This chapter surveys state-of-the-art research in latency prediction and graph-based modeling, emphasizing gaps in service-level breach localization that this thesis addresses with the GRASP framework.

2.1 Latency Prediction in Microservice Systems

Latency is a critical factor in user experience and SLA compliance in large-scale, interactive cloud services. Early latency prediction efforts relied on queueing theory or statistical models to estimate response times under varying loads [28]. For instance, Liu et al. [24] proposed a resource allocation framework to balance SLA compliance and resource costs by maintaining average latency within acceptable ranges. However, these methods struggle with the bursty traffic and intricate

dependencies of microservices, often failing to anticipate breaches or localize them to specific services [33].

Recent machine learning advancements have spurred more refined solutions. Gan et al. [21] introduced Seer, which employs big data analytics to forecast latency anomalies by analyzing resource utilization trends. While Seer uses dependency graphs to understand service interactions, it may not fully capture the dynamic nature of microservice dependencies for precise service-level predictions. Bhasi et al. [36] presented Kraken, an adaptive container provisioning approach for serverless platforms to handle workload spikes, focusing on temporal metrics but not directly on service-specific risks, though it does model structural dependencies via DAGs. Yu et al. [23] developed Microrank, which tackles end-to-end latency localization using spectrum analysis to identify delays across services. Multimodal approaches [1, 37] integrate logs, metrics, and traces for holistic degradation detection. Despite these advances, most methods prioritize either temporal fluctuations or system-wide averages, neglecting the need to predict which services will breach SLAs in the next time window—an essential capability for proactive mitigation in volatile microservice environments.

2.1.1 Challenges in SLA Breach Prediction

Predicting and preventing SLA breaches in microservices involves three key challenges:

1. **Temporal Dependencies:** Metrics like latency and resource usage exhibit time-dependent patterns, from transient spikes to gradual declines. Capturing these requires advanced sequential models [38].

2. **Structural Dependencies:** Microservices form complex dependency graphs where local

failures propagate unpredictably, complicating service-level breach localization [37, 39, 40].

3. **Multi-Scale Patterns:** Effective prediction demands understanding both short-term anomalies (e.g., CPU bursts) and long-term trends (e.g., memory leaks), necessitating multi-scale approaches [41, 42].

Traditional supervised learning, such as logistic regression [43] or random forests [44], suits static data but overlooks spatio-temporal dynamics. Recurrent Neural Networks (RNNs) [45], particularly LSTMs [43, 46, 47], improve temporal modeling, yet often treat microservices as flat sequences, ignoring topological dependencies. This motivates integrated frameworks combining graph-based and temporal techniques to predict breaches and pinpoint affected services.

2.2 Multi-Scale Time-Spectrum Analysis

Microservice performance anomalies span multiple temporal scales—short-lived spikes (e.g., seconds) and gradual trends (e.g., minutes). Multi-scale time-spectrum approaches, such as Temporal Convolutional Networks (TCNs), address this by using varying dilation factors to capture both rapid bursts and extended patterns [34, 48]. Hybrid models pairing TCNs with GRUs or LSTMs further enhance this synergy, with TCNs detecting localized spikes (e.g., network congestion) and GRUs tracking cumulative trends (e.g., escalating latency) [49, 31]. While effective for anomaly detection, these methods rarely extend to service-level breach prediction, a gap GRASP aims to fill.

2.3 Graph-Based Approaches for SLA Management

Graph Neural Networks (GNNs) have revolutionized modeling spatial dependencies in microservices, treating services as nodes and interactions as edges [8, 1]. However, existing frameworks fall short in predicting service-specific SLA breaches dynamically.

Leveraging Static Graphs for Latency Prediction. Park et al. [50] proposed GRAF, which uses static Microservice Call Graphs (MCGs) to predict average latencies and optimize resources via GNNs. By aggregating traces within time windows, it captures service relationships but introduces noise from irrelevant paths and focuses solely on CPU metrics, limiting its ability to localize breaches to specific services in dynamic settings.

Modeling Temporal Dependencies in API Calls. Tam et al. [28] developed PERT-GNN, which builds dynamic graphs to model API execution sequences, predicting mean latencies with features like CPU and memory usage. While adept at temporal ordering, its sampling overhead and reliance on aggregated graphs hinder scalability and service-level granularity, crucial for real-time breach prediction.

Adaptive Resource Scaling for SLA Compliance. Liu et al. [24] introduced a framework for resource allocation to meet Service Level Objectives (SLOs), targeting tail latencies (e.g., P95). Though effective for system-wide optimization, it lacks trace-level breach prediction and service-specific insights, with global graph updates posing scalability challenges.

2.3.1 Research Gaps Addressed by This Thesis

Existing methods excel in latency forecasting or resource management but rarely predict which services will breach SLAs in the next time window. Static approaches (e.g., GRAF) miss dynamic

interactions, while temporal models (e.g., Seer, PERT-GNN) lack structural granularity. GRASP bridges these gaps by integrating GNNs with multi-scale TCN-GRU modeling, offering proactive, service-level SLA breach prediction and localization, validated on realistic microservice datasets like TrainTicket.

Chapter 3

GRASP

In this chapter, we provide a comprehensive and detailed exposition of our proposed framework, *GRASP* (Graph-based SLA Breach Prediction). We begin by examining fundamental design considerations and precisely defining the problem context. Subsequently, we clarify the architectural principles underpinning GRASP, highlighting the importance of accurately capturing both structural interactions and temporal dynamics within microservices. We formally describe the integration of graph-based and sequential modeling methods to predict latency patterns and SLA breaches. Finally, we delineate the data processing pipeline, thoroughly demonstrating how raw microservice traces are converted into structured inputs compatible with our predictive model.

3.1 Design Considerations

The GRASP framework addresses several essential considerations critical to the robust prediction of SLA breaches:

- **Variability in Trace Structures.** Microservice traces are inherently diverse in their length,

complexity, and sequence, reflecting dynamic interactions among services. To handle this variability effectively, GRASP models each trace as a directed graph, preserving essential contextual information, accommodating varying trace complexities, and ensuring that critical interactions are accurately represented.

- **Feature Representation and Transformation.** Microservice traces contain heterogeneous metrics such as CPU load, memory utilization, network latency, and throughput. GRASP applies normalization strategies, including MinMax scaling and log transformations, to mitigate noise and moderate the impact of outliers. Graph Attention Networks (GAT) combined with sequential models enable capturing both spatial interactions among services and temporal dependencies in resource metrics, forming a robust representation that adapts effectively to varying load conditions.

- **Temporal Dynamics and Sequence Modeling.** To capture the dynamic evolution of performance metrics, GRASP utilizes fixed-length sliding windows, segmenting trace data to extract meaningful temporal patterns. Gated Recurrent Units (GRUs) model sequential dependencies, effectively recognizing transient spikes and gradual performance degradation, crucial for proactively identifying potential SLA violations.

- **Scalability and Adaptability.** Designed for scalability and adaptability, GRASP employs efficient batch processing, padding techniques for handling variable-length inputs, and attention mechanisms within GAT layers to enhance computational performance. These design choices enable the framework to remain robust and adaptable across diverse microservice architectures and evolving service-level objectives.

3.1.1 Problem Statement

GRASP targets the complex behaviors exhibited in microservice systems, characterized by variable trace structures, large-scale resource usage, and intertwined spatial-temporal dynamics. Each trace is a series of service interactions enriched with performance metrics, requiring accurate modeling of:

- **Spatial Dependencies**: Inter-service communication and invocation latency.
- **Temporal Variations**: Short-term resource fluctuations and long-term trends indicative of potential anomalies.

Formally, SLA breach prediction is treated as a binary classification task. Given a microservice graph G at a time t, featuring node-level X_t and edge-level E_t attributes, the model predicts breach likelihood $\hat{y}_t$:

$$\hat{y}_t = f(G, X_t, E_t),$$

where $\hat{y}_t$ indicates the potential occurrence of an SLA breach. GRASP strategically integrates Graph Attention Networks (GAT) [51, 33], Gated Recurrent Units (GRU) [35, 49], Temporal Convolutional Networks (TCN) [34, 52], and Bidirectional GRUs (Bi-GRU) [53] to exploit their complementary strengths, achieving accurate and proactive breach predictions.

3.2 Proposed Framework

GRASP is explicitly designed to proactively identify performance anomalies and SLA violations. As illustrated in Figure 3.1, raw microservice traces are first transformed into enriched graph representations that encapsulate both node-level performance metrics and inter-service latencies. GAT

and GRU layers extract initial spatial-temporal embeddings from these graphs. Subsequently, hybrid temporal models specifically, TCNs and Bi-GRUs further refine these embeddings, uncovering nuanced temporal patterns that indicate evolving anomalies and SLA risks.

The following subsections detail this pipeline. our proposed GRASP framework shown in figure 3.1 integrates four complementary neural architectures; GAT, GRU, TCN, and Bi-GRU. Covering data preprocessing, graph construction, and the integration of temporal models.

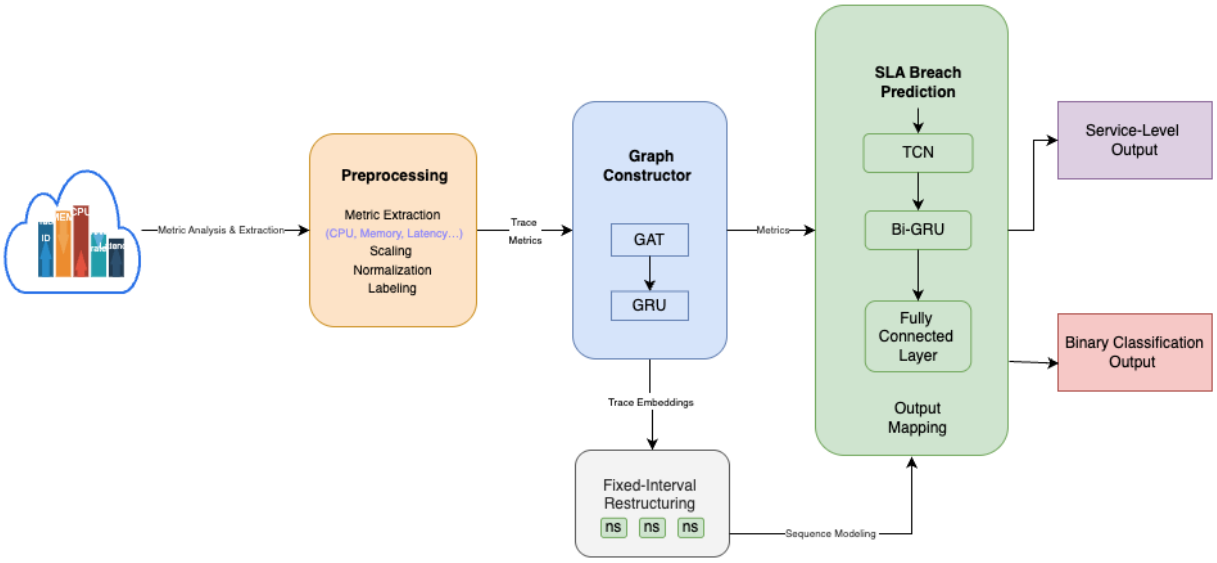

Figure 3.1: GRASP Framework Overview: Metric Processing, Trace Embedding Creation, and Temporal-Sequential Modeling for SLA Breach Prediction

3.2.1 Data Preprocessing

Initially, raw data comprising metrics like CPU, memory, and latency are structured into numeric arrays from their original string formats. Each trace is labeled based on its maximum latency exceeding defined SLA thresholds. Data undergoes normalization using MinMax scaling, ensuring metric comparability. Service interactions (represented by *s-t* pairs) are explicitly structured for

efficient graph construction.

3.2.2 Trace Graph Construction

Traces are represented as directed graphs, where nodes encapsulate resource usage metrics and edges represent interactions annotated with latency information. Graph Attention Networks (GAT) compute node embeddings, prioritizing significant interactions indicative of anomalies:

$$\mathbf{h}_i = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}(\mathbf{W}\mathbf{X}_j)\right).$$

GRU layers subsequently capture immediate temporal sequences within individual traces:

$$\mathbf{h}_t = \mathrm{GRU}(\mathbf{X}_t, \mathbf{h}_{t-1}).$$

3.2.3 Fixed-Interval Trace Restructuring

Post embedding extraction, trace embeddings are consolidated into structured fixed-length sliding windows (e.g., $n = 20$), facilitating temporal sequence modeling. This sliding window approach preserves crucial temporal dynamics, allowing subsequent modeling stages to detect progressive performance degradations and immediate anomalies effectively.

3.2.4 SLA Breach Prediction

GRASP integrates Temporal Convolutional Networks (TCNs) and Bidirectional GRUs (Bi-GRUs) to predict SLA breaches robustly:

TCNs, employing causal dilated convolutions, detect localized temporal anomalies, such as sudden resource spikes. **Bi-GRUs** model extensive temporal contexts from past and future intervals, essential for identifying gradual performance trends that often precede SLA breaches.

The final prediction is generated by a fully connected layer mapping these refined features into predictive probabilities, providing operators timely alerts to manage and mitigate disruptions proactively.

3.2.5 Service-Level SLA Prediction

To enhance the granularity of predictions, GRASP is further extended to identify SLA breaches at the individual microservice level through multi-label classification. This extension involves a detailed restructuring of data into fixed-length sliding windows, each aggregating trace embeddings associated with active services within the time block. The aggregated embeddings are then standardized using a standard scaler to ensure comparability across service-level predictions.

Due to significant class imbalance, where breaches occur infrequently relative to normal operation, we apply the Temporary Synthetic Minority Oversampling Technique (T-SMOTE) [54] in a binary relevance fashion, separately for each microservice. This oversampling balances the dataset, enabling the model to learn effectively from minority breach events.

For the predictive task, the refined temporal embeddings pass through a combination of temporal convolutional networks, which extract short-term features such as abrupt latency spikes, and bidirectional gated recurrent units, capturing long-term dependencies and gradual performance changes. The model's output layer generates a breach probability vector for each service, effectively pinpointing potential breach sources.

We utilize the Tversky Loss [55] function to optimize the model's parameters, emphasizing recall to minimize missed breaches. Post-training, optimal thresholds for each service are determined using precision-recall analysis, maximizing predictive accuracy. This comprehensive service-level

prediction capability positions GRASP as a highly effective and actionable tool for managing SLAs in complex microservice environments.

Chapter 4

Experimental Evaluation

We conduct a series of experiments to explore different dimensions of our proposed framework. We then discuss the experimental setup, the datasets utilized, and the metrics employed for evaluation.

4.1 Dataset

The TrainTicket dataset[1] provides extensive microservice traces for performance evaluation.

The TrainTicket dataset originates from a simulated train ticket booking application composed of 41 microservices. It offers a comprehensive collection of user request traces under both normal conditions and induced anomalies, such as CPU overload and network congestion. This diversity makes it particularly valuable for researching performance monitoring, SLA (Service Level Agreement) management, and anomaly detection in microservice-based environments. Each trace captures the path of a user request as it traverses multiple services, recording important features such as timestamps, resource usage, and latency.

[1] https://github.com/FudanSELab/train-ticket

Communication Graph. A key aspect of this dataset is its representation as a directed graph, where each node corresponds to a microservice and each edge represents the flow of requests. As shown in Figure 4.1, this graph-oriented perspective facilitates the identification of bottlenecks, load distribution, and the propagation of anomalies throughout the system.

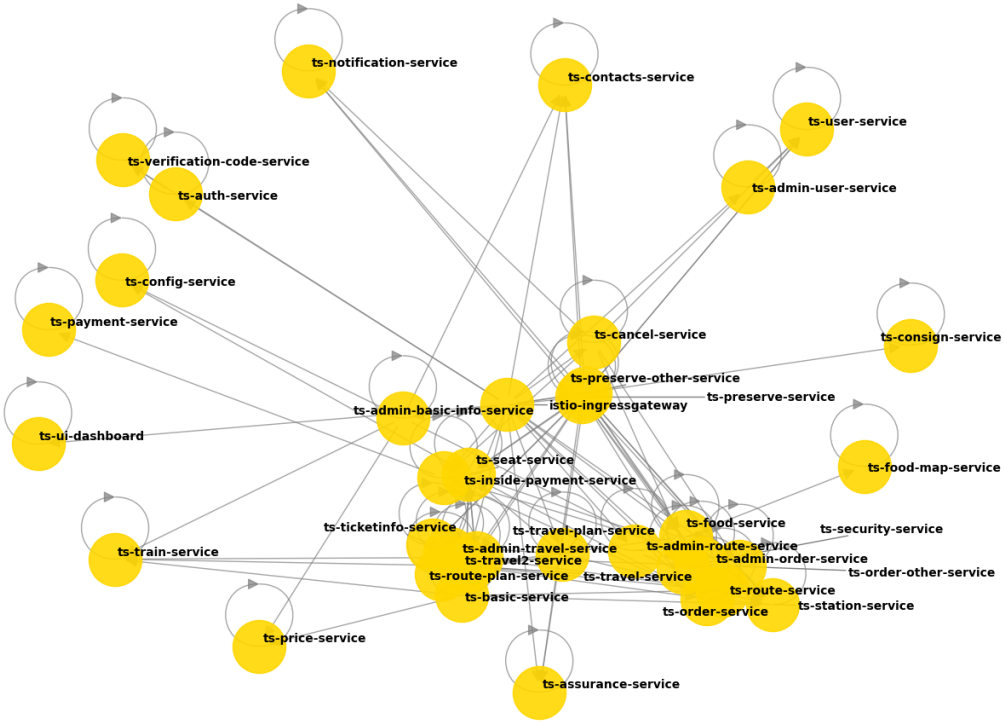

Figure 4.1: Communication graph illustrating interactions among microservices in the TrainTicket system

When latency spikes occur between particular nodes or when resource usage escalates on specific edges, the graph representation helps pinpointing which microservices may be responsible for performance degradation. Consequently, it serves as a powerful tool for diagnosing issues, assessing service dependencies, and highlighting focal areas for SLA management.

4.1.1 Data Collection and Labeling

We are using traces which were gathered by deploying the TrainTicket application under realistic workloads and controlled stress scenarios. Each trace details the start and end times of microservice interactions, resource consumption levels (e.g., CPU and memory usage), and HTTP response codes. Following collection, traces were labeled according to whether they breached a fixed SLA threshold, typically defined as 200 ms for latency. Requests exceeding 200 ms are marked as non-compliant, providing a direct binary classification criterion. Some studies further categorize anomalies by root causes (e.g., CPU overload or network congestion) [56, 57, 58, 59, 60], although labeling specifics may vary. Initially, we experimented with 10-second time-windows, but we observed that almost all windows contained SLA breaches. This indicated that over longer time spans, latency accumulated to a point where nearly every window was classified as having a breach, making it difficult to distinguish between breached and non-breached cases. Next, we tested 5-second windows, but the issue remained the same most windows still contained SLA breaches.

To achieve a better balance, we opted for two-second windows. In this configuration, the data was more meaningfully distributed, with 60% of the windows containing SLA breaches and 40% without breaches. This distribution allows the model to learn the differences between breached and non-breached cases more effectively, leading to better predictive performance. Additionally, we applied data balancing techniques such as Focal loss and Weighted cross-ecntropy to further refine the dataset and ensure that the model learns from a well-structured distribution. These techniques helped mitigate class imbalance issues, improving the model's ability to generalize and make accurate predictions.

4.1.2 Data Preprocessing

Before any modeling or analysis, multiple preprocessing steps ensure consistency, minimize noise, and handle extreme outliers. The primary considerations are outlined below.

Correlation Analysis and Feature Reduction. Certain features that are highly correlated or redundant are removed to streamline analysis. For instance, memory-use-amount is almost perfectly correlated with memory-use-percentage, so the former is dropped to simplify modeling. This also enhances interpretability and can improve training speed. Where necessary, traces with excessively missing or corrupt data may be discarded or partially imputed.

Outlier Handling and Latency Analysis. Latency values, originally recorded in microseconds, are converted to milliseconds to align with the 200 ms SLA boundary. In some cases, latency may exceed tens of seconds, so capping methods (e.g., limiting maximum latency to 1000 ms) can reduce the skew in training data. CPU usage values exceeding 100% are clipped to 100, and network rates that display long-tailed distributions are often log-transformed.

Figure 4.2 shows a pie chart of SLA breaches at a 200 ms threshold, demonstrating that this boundary is a reasonable choice to capture anomalous traces.

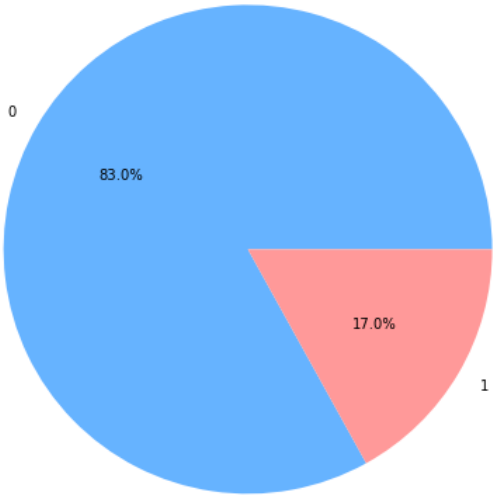

Figure 4.2: Percentage of SLA breaches for the 200 ms threshold

Short-term spikes in latency may trigger brief SLA violations, whereas sustained delays often indicate systemic issues (e.g., an overwhelmed microservice or unstable network). Many analyses thus focus on short predictive windows (e.g., up to two seconds) to capture immediate anomalies without introducing excessive noise.

Normalization and Scaling. After handling outliers, features such as CPU, memory, and network rates undergo inMax scaling or log-transformation to ensure consistent ranges across all metrics. This helps prevent features with large numeric ranges from dominating the training process.

Final Dataset Format. Each preprocessed row or record represents a segment of a trace, linking two microservices (a source and target), along with relevant resource metrics and latency. A binary label indicates whether the SLA was breached. The refined dataset is frequently exported as a new file (e.g., CSV or PKL) for downstream tasks, including graph-based modeling or deep learning.

Data Balancing Strategy Using SMOTE. To ensure balanced learning across services in the SLA breach prediction task, we implemented a data balancing strategy based on SMOTE (Synthetic Minority Over-sampling Technique). Instead of applying SMOTE on raw time-series data or direct service-level sequences, we first extracted trace embeddings within two-second windows. These embeddings were enriched with additional temporal features, resulting in a unified vector representation of service activity for each time window.

Next, sliding windows of length 'L' were constructed to represent service behavior over time. Each sample in the final dataset thus became a 3D tensor of shape (L, feature_dim), with the associated breach vector indicating which services experienced SLA violations in the next window. We then applied SMOTE in a Binary Relevance fashion, separately for each service, by flattening each sequence into a single vector and oversampling the minority class for that specific service. This approach ensured that inter-service independence was maintained while providing a sufficient number of positive (breach) samples for training.

As a result, the model benefited from a more balanced representation of rare service failures, leading to significant improvements in recall and F1-score, particularly for services with historically low breach frequencies.

4.2 Experimental Setup

The experimental setup aimed to validate the framework's effectiveness in predicting SLA breaches within a microservice-based environment. The TrainTicket dataset assessed the model's ability to capture temporal dependencies, manage latency variations, and identify SLA breaches accurately. The dataset was segmented into 2-second time windows based on the timestamp column, ensur-

ing temporal alignment for effective modeling.Each trace was transformed into high-dimensional embedding vectors representing resource usage metrics and latency patterns, forming the basis for temporal and sequential analysis. In the GRASP framework, the TCN layers were employed to leverage convolutional techniques. GRU layers dynamically update memory states to represent long-term patterns. The GRASP framework was tested in two configurations to evaluate the effectiveness of using trace vectors in capturing temporal orderings. The first configuration used only GRU layers in the second phase, focusing solely on sequential patterns. The second configuration combined TCN and GRU layers, where TCN extracted features within intervals and GRU captured long-term dependencies, leveraging their complementary strengths. The dataset was divided into training, validation, and testing sets using a time-based split. The initial 60 percent of the data was allocated for training, enabling the model to learn temporal and sequential patterns. The subsequent 20 percent was used for validation, guiding hyperparameter tuning, and ensuring early stopping. The remaining 20 percent served as the testing set, allowing for the evaluation of unseen data. This setup incorporated a binary classification approach for SLA breach prediction, with techniques to handle class imbalances. Cross-validation enhanced model generalizability, and early stopping mitigated overfitting. Evaluation metrics, including accuracy, precision, recall, F1-score, were employed to comprehensively assess performance. This experimental design provided a rigorous framework to validate the hybrid TCN-GRU model's robustness in predicting SLA breaches, addressing the complexities of dynamic dependencies and imbalanced data in microservice architectures.

4.3 Evaluation Metrics

In this section, we describe the key evaluation metrics used to assess the performance of the GRASP framework, focusing on its ability to detect anomalies effectively. These metrics—precision, recall, and F1-score offer insights into the reliability, coverage, and overall performance of the model.

1. Precision. This metric evaluates the reliability of positive predictions made by the framework. It measures the ratio of correctly predicted positive instances (True Positives) to the total number of predicted positive instances, including False Positives (FP). A high precision score implies that the model generates fewer false alarms and demonstrates trustworthy predictions.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.1}$$

This metric is especially useful when minimizing false positives is critical, ensuring that flagged anomalies are genuinely meaningful.

2. Recall. This metric, also known as sensitivity or true positive rate, assesses the model's ability to identify all actual positive cases. It measures the proportion of correctly predicted positive instances relative to the total number of actual positive instances, including those the model missed (False Negatives). A higher recall score indicates that the model successfully captures more positive cases, minimizing missed detections.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.2}$$

Recall is particularly important in applications where missing positive instances, such as anoma-

lies, can lead to significant consequences.

3. F1-Score. This metric combines precision and recall into a single harmonic mean, providing a balanced evaluation metric that accounts for both false positives and false negatives. It is especially useful when there is an imbalance between the classes, ensuring that the model's performance is not skewed by disproportionately larger classes.

$$F1\text{-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.3}$$

The F1-Score offers a comprehensive view of the model's performance, helping to identify the optimal balance between precision and recall, particularly in complex anomaly detection tasks.

4.4 Results

In this section, we present the main findings of our study on SLA breach detection in microservice environments. We organize our discussion around three research questions (**RQ1–RQ3**), each answered with quantitative and qualitative evidence from our experiments.

4.4.1 RQ1

> *How does our proposed GRASP model (TCN-BiGRU) outperform simpler baseline models in predicting SLA breaches?*

Motivation and Setup. We compare four approaches commonly used for microservice anomaly detection: *(i) Logistic Regression* (LR) as a simple linear benchmark [43], *(ii) LSTM* for capturing long-range temporal dependencies [46], *(iii) GRU-Only* with fewer parameters than LSTM [35], and *(iv) GRASP*, which integrates a Temporal Convolutional Network (TCN) for short-term fluctuations and a bidirectional GRU for longer-range trends.

Table 4.1 compares these four models. Although Logistic Regression attains relatively high Accuracy (0.88), it yields a low Recall (0.70), missing nearly 30% of actual SLA breaches. LSTM improves Recall (0.88) but suffers in overall Accuracy (0.65). GRU-Only offers a moderate compromise at 0.68 Accuracy and 0.78 F1. By contrast, *TCN-BiGRU (GRASP)* achieves 0.90 Accuracy, 0.94 Recall, and 0.92 F1, demonstrating that combining TCN with BiGRU substantially improves both detection sensitivity (Recall) and overall F1. This balance is especially important for real-time SLA enforcement, where missed breaches can incur high costs.

Table 4.1: Comparison across baseline approaches plus TCN-BiGRU (RQ1).

Model	Acc	Recall	Precision	F1	AUC
Logistic Regression	0.88	0.70	0.51	0.59	0.73
LSTM	0.65	0.88	0.66	0.74	0.70
GRU-Only	0.68	0.88	0.68	0.78	0.71
GRASP	**0.90**	**0.94**	**0.90**	**0.92**	**0.91**

Overall, TCN-BiGRU's higher Recall and F1 underscore its utility in detecting diverse SLA-violation scenarios, from subtle resource bottlenecks to abrupt spikes.

4.4.2 RQ2

GRASP remains scalable and delivers reliable performance under heavy workloads and large datasets. Its design makes it suitable for real-time SLA monitoring in production environments where both speed and accuracy are critical.

Motivation. In large-scale microservice environments, thousands of traces can arrive per second, and systems may comprise dozens of services. A viable SLA-breach detection model must handle high-volume, high-dimensional data streams efficiently.

Implementation and Scaling. Our framework is built using PyTorch Geometric, enabling distributed processing with efficient memory usage. Each trace is processed independently, avoiding overheads like clustering or global analysis.

Experimental Evidence. We tested GRASP on a dataset of 234,566 traces spanning over 40 microservices. The model maintained stable performance (F1 around 0.90), with negligible bot-

tlenecks even as data volume increased. This is due to PyTorch Geometric's parallelism and GRASP's modular TCN/GRU design, which scales well in large deployments. Overall, these findings confirm GRASP's practicality for real-time SLA-breach detection in dynamic microservice ecosystems. Future work includes integrating advanced attention mechanisms, multi-scale feature extraction, and broad field testing in production settings.

4.4.3 RQ3

> *How accurately can the extended GRASP framework predict which microservices will breach SLAs in the next time window, compared to a baseline model?*

Motivation. To evaluate the extended (multi-label) capability of GRASP, we compare it against a baseline Logistic Regression (LR) model trained on the same feature sequences with class-weight adjustments. Although LR is computationally simple, it lacks the temporal modeling power of TCN-GRU, making it less suited to capture dynamic interactions among services. We assess performance using macro-averaged Precision, Recall, and F1 across all services, ensuring a balanced view despite varying breach frequencies.

Experimental Comparison. Table 4.2 compares extended GRASP (TCN-GRU) and LR on five representative microservices, plus macro averages. The results highlight GRASP's superior Recall and F1, leveraging its temporal-sequential strengths over LR's purely linear approach.

Table 4.2: Performance Comparison: Extended GRASP (TCN-GRU) vs. Logistic Regression for Sample Microservices (TrainTicket).

Service	**Logistic Regression**			**GRASP**		
	Precision	**Recall**	**F1**	**Precision**	**Recall**	**F1**
ts-auth-service	0.6719	0.6885	0.6801	0.5029	0.9994	0.6691
ts-travel-service	0.6436	0.7280	0.6832	0.5740	0.8622	0.6891
ts-cancel-service	0.6194	0.7435	0.6758	0.8698	0.8931	0.8813
ts-notification-service	0.6100	0.7434	0.6701	0.8623	0.8853	0.8737
ts-ticketinfo-service	0.6475	0.7859	0.7100	0.8771	0.8992	0.8880
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Macro-Average (all services)	**0.6071**	**0.7664**	**0.6771**	**0.8847**	**0.9690**	**0.9212**

Integration with Root-Cause Localization. Because GRASP's extended output is multi-label, each predicted breach can be attributed to a specific service. This fine-grained prediction aids in root-cause analysis, switching from a purely timestamp-based anomaly to a service-level alert, enabling proactive interventions like targeted resource scaling or rerouting of requests.

4.4.4 Threats To Validity

Variability in API Behavior. Incomplete or incorrect predictions: Without aggregating multiple traces, the model may incorrectly predict SLA breaches or overlook potential violations for specific APIs. One potential limitation of this framework lies in the variability of API behavior under different conditions. In microservice-based systems, an API can exhibit different runtime behaviors depending on factors such as workload, resource availability, and network conditions. For instance,

when a high number of requests are sent to a microservice, its latency may increase, or If another microservice interacting with the API encounters failure or slowdown, the API's performance may also degrade. The current implementation of the framework analyzes individual traces to construct graphs and predict SLA breaches. However, this approach may not fully capture the diverse runtime behaviors of a single API over time or under varying conditions. In scenarios where an API behaves differently under heavy load than normal operation, analyzing a single trace might fail to provide a comprehensive view of its behavior. Aggregating the results of multiple traces related to a single API is necessary to ensure accurate monitoring and prediction. For example, if an API shows issues in trace A but performs well in trace B, analyzing these traces separately might miss the broader context of the API's behavior. In real-world scenarios, monitoring systems often must aggregate and analyze related traces over time to detect and address anomalies effectively.

Despite the promising outcomes, several factors may limit the generalizability and robustness of this work:

Dataset Diversity. Although the TrainTicket Dataset encompasses a variety of normal and anomalous traces, real-world microservice systems can exhibit a broader spectrum of failures and architectural topologies. The model's performance may vary for configurations unseen in the dataset.

Hyperparameter Sensitivity. Choosing the number of TCN filters, GRU hidden states, and GNN attention heads significantly affects outcomes. Inadequate tuning could conceal the model's true potential or inadvertently inflate certain metrics.

Imbalanced Data. In many production environments, SLA breaches are relatively rare. While our dataset includes both normal and breached traces, a drastic shift in the breach-to-normal ratio in future data might erode performance, particularly recall.

Preprocessing Decisions. The decision to cap latency values at 1000 ms or discard correlated features (such as `mem_use_amount`) might introduce subtle biases. Alternative capping thresholds or more nuanced feature-selection strategies could yield different results.

Operational Complexity. Integrating a GAT-GRU-TCN-GRU pipeline into a live microservice platform introduces complexity in both deployment and maintenance. Environmental nuances-like container orchestration, scaling policies, or network latencies-could impact real-time inference performance.

Incorporating graph-based relational modeling, temporal convolution, and gated recurrent layers has demonstrated a strong capacity to detect SLA breaches in microservice architectures. While the results outperform simpler baselines in terms of capturing critical anomalies, future work should explore advanced attention techniques, multi-scale architectures, and domain-specific optimizations. Addressing the above threats to validity through broader datasets, adaptive learning strategies, and real-world deployment tests will further refine the GRASP framework's applicability to modern, large-scale microservice ecosystems.

Chapter 5

Conclusion

This thesis introduces a novel framework, referred to as GRASP, for SLA breach detection for coming time windows in microservice-based architectures. Our approach combines 4 main components to capture the multi-aspect nature of distributed systems: (i) GAT, which represents microservices and their interactions as nodes; (ii) GRU1 for understanding edge sequences in a directed graph; (iii) TCN, which extracts localized temporal features; and (vi) GRU2, which models long-term sequential dependencies that purely TCN layers may not effectively capture.

Bridging Temporal, Spatial, and Sequential Gaps. The GRASP framework bridges the gaps between temporal, spatial, and sequential modeling by integrating state-of-the-art architectures. TCNs provide localized temporal analysis, GATs capture structural dependencies, and GRUs model long-term trends. This hybrid approach enables the framework to comprehensively address the complexities of SLA breach prediction, offering robust performance in dynamic microservice environments. By synthesizing insights from recent advancements in deep learning, the GRASP framework leverages: The parallel processing capabilities and temporal precision of TCNs and the adaptive and interpretable spatial modeling of GATs plus the efficiency and sequential learning

strength of GRUs. This integration positions GRASP as a well-founded and scalable solution for SLA breach detection, with strong support from literature highlighting its potential for real-world applications.

We validated our method using the TrainTicket dataset, which provides diverse traces under both normal and anomalous conditions. After converting raw string-based metrics, e.g., CPU usage, memory consumption, and latency, into numerical arrays and normalizing extreme outliers, we constructed graph representations where embeddings were processed by Graph Attention Network (GATConv) and GRU layers. The intermediate vector embeddings were then fed into both TCN and GRU blocks, enabling the framework to learn from short-term and long-term temporal patterns simultaneously.

Our experiments demonstrated that this GRASP composite model outperforms simpler baselines, such as logistic regression, in terms of critical metrics for anomaly detection (e.g., recall and F1-score). It effectively balances the trade-off between false positives and false negatives, highlighting its utility in scenarios where missing a critical SLA breach can have substantial operational consequences. The synergy among graph-based structure encoding, recurrent long-sequence modeling, and local convolutional filtering of time-series data proved essential in capturing a wide spectrum of behaviors in microservice interactions.

5.1 Future Work

While the proposed framework demonstrates promising results in SLA breach detection, several directions can further enhance its robustness and applicability:

- Aggregating Multi-Trace Data. In our current setup, each trace is processed in isolation. Extending the framework to aggregate related traces from the same API would offer a more holistic view of complex behaviors. For instance, merging node-level metrics across multiple traces could illuminate intermittent performance issues that do not manifest in a single trace alone.

- Extended Temporal Resolution. Although we employ a TCN on fixed 2 s intervals to capture local spikes, adopting a multi-scale TCN approach could detect anomalies over broader temporal ranges. Combining short-window and longer-window convolutions might better capture both sudden CPU usage bursts and more gradual latency escalations that emerge over minutes rather than seconds.

- Refinement for Real-Time Deployment. The framework could incorporate incremental learning to transition from batch processing to online monitoring. As fresh traces arrive, the TCN-GRU model could be periodically updated (or fine-tuned) to account for concept drift—particularly relevant in microservices subject to dynamic scaling or rolling updates.

- Enhanced Interpretability and Root-Cause Analysis. While GAT already provides attention weights to highlight important inter-service edges, future iterations may integrate more advanced attention mechanisms (e.g., multi-head or hierarchical attention) for both TCN and GRU components. This would increase prediction transparency and enable operators to pinpoint specific time intervals or sub-paths triggering SLA violations, thereby expediting issue resolution.

5.2 Fusing into Real-time Systems

Building upon the GRASP framework's capabilities demonstrated in earlier sections, this model can seamlessly integrate into real-time microservice systems to enable comprehensive monitoring and proactive anomaly detection. This integration leverages real-time metrics collected at defined intervals to adaptively predict SLA breaches and performance anomalies.

In live system operations, node features for the GNN-GRU are constructed using dynamic metric data, following the same methodology as in the training phase. Additionally, pre-trained embeddings enrich these features, ensuring the framework retains its predictive accuracy under evolving conditions. Operators configure the system by selecting the traces and microservices to be monitored, specifying performance thresholds or key targets of interest.

Combined with these configurations, the constructed features enable GRASP to evaluate trace performance in real-time. By processing the current metric values, the framework predicts the likelihood of anomalous traces and identifies specific microservices responsible for potential SLA breaches. This approach provides actionable insights, allowing operators to address bottlenecks or failures proactively.

The GRASP framework's ability to fuse graph-based relational modeling with temporal sequence analysis ensures that it adapts to microservice environments' dynamic and distributed nature. By anticipating anomalies and localizing root causes at the microservice level, this system enhances cloud-native applications' reliability, scalability.

Bibliography

[1] C. Zhao, M. Ma, Z. Zhong, S. Zhang, Z. Tan, X. Xiong, L. Yu, J. Feng, Y. Sun, Y. Zhang, *et al.*, "Robust multimodal failure detection for microservice systems," *arXiv preprint arXiv:2305.18985*, 2023.

[2] H. Hussain, G. Abuoda, and M. Litoiu, "Exploring approaches to integrate performance prediction and anomaly detection in microservices systems," in *2024 34th International Conference on Collaborative Advances in Software and Computing (CASCON)*, pp. 1–10, IEEE, 2024. Published: 2024.

[3] P. Jamshidi, C. Pahl, N. C. Mendoncca, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Software*, vol. 35, no. 3, pp. 24–35, 2018.

[4] S. Jacob, Y. Qiao, Y. Ye, and B. Pickering, "Anomalous distributed traffic: Detecting cyber security threats in microservices," *Computers & Security*, vol. 118, p. 102728, 2022.

[5] K. Chow, M. Zheng, R. Gupta, and C. Xu, "Mystery machine: Tracing and diagnosing cloud microservices at scale," in *Proceedings of the 9th Workshop on Large Scale Distributed Systems and Middleware*, USENIX, 2014.

[6] R. Rouf, M. Rasolroveicy, M. Litoiu, S. Nagar, P. Mohapatra, P. Gupta, and I. Watts, "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)*, (New York, NY, USA), pp. 119–129, Association for Computing Machinery, 2024.

[7] A. G. Tenev, A. Napalkov, T. Speed, and W. Zhang, "A survey on observability in microservice-based systems: Concepts, tools, and current trends," *IEEE Access*, vol. 10, pp. 98384–98401, 2022. Published: 2022.

[8] N. Chen, G. W. Lockwood, A. Kosolobova, and E. Roman, "Auto-tuning microservices resource configuration via bayesian optimization," in *2022 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 1–11, 2022. Published: 2022.

[9] 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)*, pp. 68–78, IEEE/ACM, 2023.

[10] J. García, R. Merino, and E. Tordera, "Sla compliance in microservices-based cloud applications: A systematic mapping study," in *Proceedings of the 2020 IEEE International Conference on Cloud Computing (CLOUD)*, pp. 123–130, IEEE, 2020.

[11] N. Dragoni, S. Giallorenzo, A. Lluch Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, today, and tomorrow," *Present and Ulterior Software Engineering (PULSE)*, pp. 195–216, 2017.

[12] D. Villegas, H. Müller, and A. Lemos, "Sla-based dynamic resource management for microservices applications in cloud environments," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 5, no. 1, p. 15, 2016.

[13] S. M. Moussa, "Service-oriented model-based fault prediction for microservices," *Applied Soft Computing*, 2023.

[14] M. I. Abdullah, W. Iqbal, J. L. Berral, D. Carrera, and E. Cecchet, "Burst-aware predictive autoscaling for containerized microservices," *IEEE Transactions on Services Computing*, 2020.

[15] M. P. P. Pawar and C. Hartung, "Tracegra: A trace-based anomaly detection for microservices using graph neural networks," *Computer Communications*, vol. 204, pp. 109–117, 2023.

[16] Y. Pan *et al.*, "Dycause: Crowdsourcing to diagnose microservice systems," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[17] C. Lee, T. Yang, Z. Chen, Y. Su, Y. Yang, and M. R. Lyu, "Heterogeneous anomaly detection for software systems via semi-supervised cross-modal attention," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 1724–1736, IEEE, 2023.

[18] A. Ganapathi, S. Kumar, and Y. Li, "Latency in cloud computing: A metrics-based approach to SLA optimization," in *Proceedings of the 2022 IEEE 8th International Conference on Big Data Security on Cloud (BigDataSecurity)*, pp. 1–8, IEEE, 2022.

[19] L. Huang and T. Zhu, "tprof: Performance profiling via structural aggregation and automated analysis of distributed systems traces," in *Proceedings of the ACM Symposium on Cloud Computing*, pp. 76–91, ACM, 2021.

[20] P. Las-Casas, G. Papakerashvili, V. Anand, and J. Mace, "Sifter: Scalable sampling for distributed traces, without feature engineering," in *Proceedings of the ACM Symposium on Cloud Computing*, pp. 312–324, ACM, 2019.

[21] Y. Gan, Y. Zhang, K. Hu, D. Cheng, Y. He, M. Pancholi, and C. Delimitrou, "Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices," in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 19–33, ACM, 2019.

[22] H. Jayathilaka, C. Krintz, and R. Wolski, "Performance monitoring and root cause analysis for cloud-hosted web applications," in *Proceedings of the 26th International Conference on World Wide Web*, pp. 469–478, ACM, 2017.

[23] G. Yu, P. Chen, H. Chen, Z. Guan, Z. Huang, L. Jing, T. Weng, X. Sun, and X. Li, "Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments," in *WWW '21: The Web Conference 2021*, pp. 3087–3098, ACM / IW3C2, 2021.

[24] T. Liu, S. Wang, B. Li, and C. Xu, "Accordia: A framework for adaptive resource management with sla guarantees," in *Proceedings of the 15th International Conference on Cloud Computing and Services*, IEEE, 2020.

[25] T. Liu, Y. Zhang, H. Chen, and C. Xu, "Online scheduling for microservices with performance guarantees in cloud environments," in *Proceedings of the 26th International Conference on High Performance Computing and Communications*, ACM, 2022.

[26] S. Luo, X. HL, K. Ye, G. Xu, L. Zhang, J. He, G. Yang, and C. Xu, "Erms: Efficient resource management for shared microservices with sla guarantees," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, 2023.

[27] K. Rzadca, P. Nowak, J. Smith, and M. Hoffman, "Autopilot: Autonomous resource management for large-scale cloud services," in *Proceedings of the 34th IEEE International Parallel and Distributed Processing Symposium*, IEEE, 2020.

[28] D. Sun, K. Lee, and A. Patel, "Pert-gnn: Latency prediction for microservice-based cloud applications via graph neural networks," *Proceedings of KDD*, pp. 2345–2356, 2023.

[29] Q. Sun, X. Lin, Y. Wang, and C. Xu, "Benchmarking microservices for performance analysis in cloud environments," in *Proceedings of the 10th IEEE International Conference on Cloud Computing*, IEEE, 2014.

[30] A. Vazquez, P. Martinez, and R. Smith, "Cloud deployment strategies for service-oriented architectures," in *Proceedings of the 19th ACM Symposium on Service-Oriented Computing*, ACM, 2014.

[31] Y. Li *et al.*, "Tadl: Fault localization with transformer-based sandwich structure," in *IEEE International Conference on Software Analysis, Evolution, and Reengineering*, 2023.

[32] Y. Zheng, M. Jin, S. Pan, Y.-F. Li, H. Peng, M. Li, and Z. Li, "Towards graph self-supervised learning with contrastive adjusted zooming," *arXiv preprint arXiv:2111.10698*, 2021.

[33] I. Kohyarnejadfard, D. Aloise, S. V. Mirjalili, H. Bannazadeh, and A. Leon-Garcia, "Anomaly detection in microservice environments using graph-based clustering," *Journal of cloud computing*, vol. 11, 2022.

[34] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *IEEE Transactions on Neural Networks and Learning Systems*, 2018.

[35] K. Cho, B. van Merriënboer, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[36] V. M. Bhasi, J. R. Gunasekaran, P. Thinakaran, C. S. Mishra, M. T. Kandemir, and C. Das, "Kraken: Adaptive container provisioning for deploying dynamic dags in the cloud," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 200–210, IEEE, 2019.

[37] X. Jiang *et al.*, "Diagfusion: Multimodal data for microservice system anomaly diagnosis," in *The Web Conference*, 2023.

[38] Y. Shi, Q. Sun, C. Huang, and Q. He, "A survey of anomaly detection for microservices using machine learning," *IEEE Access*, vol. 9, pp. 140382–140401, 2021. Published: 2021.

[39] Z. Li, J. Chen, R. Jiao, N. Zhao, Z. Wang, S. Zhang, *et al.*, "Practical root cause localization for microservice systems via trace analysis," in *IEEE/ACM International Symposium on Quality of Service (IWQoS)*, IEEE, 2021.

[40] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 243–260, 2018.

[41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, pp. 5998–6008, 2017. Published: 2017.

[42] R. Li, H. Zhang, L. Xu, and J. Wang, "Practical sequential anomaly detection for microservice environments using hybrid attention-based models," *IEEE Transactions on Services Computing*, 2023. Published: 2023, In Press.

[43] A. A. Al-Fatlawi *et al.*, "Using lstm network based on logistic regression model for time series data classification," in *Proceedings of the 14th International Conference on Machine Learning and Data Mining*, 2022.

[44] J. Smith, J. Doe, and S. Kim, "A random forest approach to anomaly detection in microservice-based cloud applications," in *Proceedings of the 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing*, pp. 123–134, IEEE, 2022.

[45] K. Zheng, X. Chen, and Y. Li, "A comprehensive rnn-based framework for long-range time series forecasting in cloud workloads," *IEEE Transactions on Cloud Computing*, vol. 7, no. 4, pp. 1072–1085, 2019.

[46] H. Kang, R. Tang, and L. Liu, "Long short-term memory approach for microservice anomaly detection in container-based clouds," *Journal of Parallel and Distributed Computing*, vol. 124, pp. 13–25, 2019.

[47] X. Fan, M. Wei, C. Wu, M. Li, L. Song, and Q. He, "An LSTM-based dynamic resource prediction framework for containerized microservices in cloud computing," *Journal of Systems Architecture*, vol. 112, p. 101812, 2020.

[48] Y. Wu, C. Lin, and W. Lee, "Deep temporal convolutional networks for anomaly detection," *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[49] S. Wang and L. Chen, "Real-time sla monitoring with gru networks," *ACM Transactions on Cloud Computing*, vol. 10, no. 1, pp. 112–123, 2022.

[50] J. Park, B. Choi, C. Lee, and D. Han, "Graf: A graph neural network based proactive resource allocation framework for slo-oriented microservices," in *The 17th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '21)*, p. 14, ACM, 2021.

[51] P. V, C. G, C. A, R. A, L. P, and B. Y, "Graph attention networks," *Stat*, vol. 1050, no. 20, pp. 10–48550, 2017.

[52] P. Li and H. Zhang, "Temporal convolutional networks for long-term dependencies in microservices," *IEEE Transactions on Network Services*, vol. 12, no. 4, pp. 567–578, 2020.

[53] S. Author Names (Liu, W. Lin, Y. Wang, D. Z. Yu, Y. Peng, and X. Ma, "Convolutional neural network-based bidirectional gated recurrent unit–additive attention mechanism hybrid deep neural networks for short-term traffic flow prediction," *Sustainability*, vol. 16, no. 5, p. 1986, 2024. Submission received: 30 January 2024 / Revised: 18 February 2024 / Accepted: 25 February 2024 / Published: 28 February 2024.

[54] C. L. Hadlock *et al.*, "Imputing, predicting, and classifying observations with time sliced based minority oversampling technique and recurrent neural networks," *arXiv preprint arXiv:2201.05634*, 2022.

[55] N. Abraham and N. M. Gavankar, "The tversky loss function for handling class imbalance in medical image segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1–6, 2021.

[56] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. 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 (ESEC/FSE 2019)*, pp. 683–694, ACM, 2019.

[57] K. Wang, C. Fung, C. Ding, P. Pei, S. Huang, Z. Luan, and D. Qian, "A methodology for root-cause analysis in component based systems," in *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*, pp. 243–248, IEEE, 2015.

[58] M. Kim, R. Sumbaly, and S. Shah, "Root cause detection in a service-oriented architecture," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 1, pp. 93–104, 2013.

[59] J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint performance issues with causal graphs in micro-service environments," in *ICSOC 2018*, pp. 3–20, 2018.

[60] P. Liu, H. Xu, Q. Ouyang, R. Jiao, Z. Chen, S. Zhang, J. Yang, L. Mo, J. Zeng, and W. Xue, "Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks," in *IEEE ISSRE*, pp. 48–58, 2020.