GUIDE - GNN based Unified Incident Detection for Microservices Application Deployments

Anurag Dutt*, Doseok Jang†, Joao Nadkarni†, Kai Su†, Anshul Gandhi*
*Stony Brook University, †Observe, Inc.

***Abstract*—Microservices deployments in the real-world present significant challenges in detecting and localizing performance bottlenecks due to their scale, complexity, and dynamic interactions. This paper presents GUIDE, a GNN-based framework for unified incident detection and bottleneck localization, leveraging multi-source telemetry and a customizable incident trigger warning mechanism. Specifically, GUIDE employs a novel integration of Graph Attention Networks, temporal embeddings, and an expert classifier to predict and localize bottlenecks efficiently in practice. Evaluation results on real-world traces collected from Observe—a live, cloud-native platform—show that GUIDE achieves an F1-score of 87% for anomaly detection and 84% for bottleneck localization, outperforming existing baselines. Additionally, GUIDE's incident trigger warning mechanism achieves an F1-score of 85%, ensuring early and accurate detection of system failures.**

I. Introduction

The rapid adoption of microservices architectures in cloud-native environments has significantly improved scalability and maintainability for large-scale applications. However, due to the sheer scale of modern microservices applications, such as Netflix and Amazon, diagnosing and mitigating performance bottlenecks has become increasingly challenging. Microservices application graphs are inherently complex due to their large size, dynamic interactions, asynchronous calls, caching mechanisms, and evolving service dependencies [12], [15]. This complexity makes it difficult to efficiently detect and localize performance degradation as bottlenecks can propagate across services in unpredictable ways.

Traditional performance debugging tools rely on hand-crafted features or rule-based heuristics to analyze microservice interactions. While these approaches may work for simpler or synthetic systems, they struggle with modern, real-world, cloud-native microservices applications due to their dynamic nature, workload variability, and complex service dependencies. Additionally, existing methods often fail to generalize across different architectures, leading to ineffective bottleneck detection in real-world scenarios [10], [15], [16].

Much of the prior work has been designed and evaluated for synthetic benchmarks and datasets (such as DeathStarBench [2]). However, real-world traces exhibit complexities that are not present in synthetic benchmarks. For example, production microservices compete for shared resources, causing noisy neighbor effects that lead to latency spikes and resource contention. Unpredictable failures, such as cascading dependency failures, packet loss, garbage collection pauses, and database locks, further add to complexity. Further, real deployments exhibit variations in CPU usage, memory architectures, and storage backends introduce performance inconsistencies that are absent in synthetic environments.

This work presents ***an efficient and practical Graph Neural Network (GNN)-based unified incident detection approach, GUIDE, that learns latent representations of microservice interactions through a data-driven model***. GUIDE enables efficient and accurate anomaly detection and bottleneck localization, and also includes a framework to trigger incident warnings. GUIDE leverages Graph Attention Network (GAT) [19] to model dynamic service dependencies and accurately localize bottlenecks. To ensure robust detection across diverse workloads, GUIDE leverages multi-source telemetry, including latency and resource utilization metrics. The telemetry data also allows GUIDE to identify patterns that correlate with failures, thereby aiding the mitigation of failures before they escalate. To respond to dynamic variations in practice, GUIDE introduces an adaptive thresholding mechanism that automatically adjusts based on telemetry data, reducing false positives and improving early detection of persistent bottlenecks.

One of the key contributions of our work is that we have ***validated GUIDE on real-world data that we collected from a production microservices platform***. Our dataset is derived from actual operational telemetry at *Observe*, a cloud-native observability platform that enables telemetry analysis for cloud applications. The dataset includes 2.4 million distributed traces from a live environment, capturing real-world challenges that synthetic datasets often fail to reproduce. By leveraging real-world telemetry data, GUIDE accurately and efficiently predicts bottlenecks, distinguishes between transient and systemic failures, and provides a holistic view of system performance, all of which are critical requirements for real-world deployments.

We implement GUIDE as an incident-detection and warning system that can be integrated with, for example, a Kubernetes cluster. Designed for scalability, GUIDE achieves low-latency inference, making it suitable for real-time deployment in large-scale microservices environments. Our evaluation results show that, using F1-score, GUIDE outperforms existing Deep Learning-based approaches by 11%, existing state-of-the-art heuristic-based approaches by 234%, and is able to successfully detect incidents with an ***overall accuracy of 81%***. By evaluating on real-world operational conditions, we ensure that GUIDE generalizes beyond controlled lab environments.

II. Related Work

In the context of efficient and scalable incident detection solutions, prior work can be categorized into two main areas: (a) anomaly detection, and (b) bottleneck localization (or root cause analysis). Below, we discuss the most relevant related

work for incident detection in microservices applications; for a broader overview, we refer readers to survey articles [14].

A. Anomaly Detection

DeepTraLog [22] embeds log events into a unified trace event graph, leveraging a gated GNN for anomaly scoring. TraceVAE [21] employs a dual-variable graph variational autoencoder with Negative Log-Likelihood (NLL) to detect anomalies. Similarly, TraceAnomaly [7] utilizes deep Bayesian networks trained offline to recognize deviations from normal trace patterns. While these methods effectively detect anomalies, they do not address the localization of root causes, which is essential for performance mitigation.

B. Bottleneck Localization

Existing bottleneck localization techniques are often tailored for single bottlenecks and overlook the complexity of multiple, possibly interacting, failures. Groot [20] constructs a causality graph based on domain knowledge, requiring continuous manual intervention to track evolving dependencies. CRISP [24], Uber's tool for critical path analysis, necessitates constant recomputation due to the dynamic nature of microservices applications. Murphy employs a linear model that struggles to capture the complexities of microservices environments [4].

FIRM [12] relies on a Support Vector Machine (SVM) trained on handcrafted features, focusing solely on latency while disregarding structural dependencies within the call graph. Seer [3], an online debugging system using CNNs and LSTMs, has been shown to be ineffective in capturing complex graph dynamics, prompting researchers to advocate for GNN-based approaches [8]. ϵ-diagnosis [13] localizes bottlenecks using threshold-based anomaly detection and distance correlation, but its lack of structural information impacts its effectiveness in identifying multiple bottlenecks.

Other heuristic and graph-based approaches also exhibit limitations. AutoMAP [9] employs random walk heuristics for bottleneck localization but struggles with large call graphs [24]. B-MEG [16] applies a two-stage graph learning model but is restricted to single bottlenecks. GAMMA [15] is a follow-up work for multiple bottlenecks but is not designed for real-time incident detection and also does not differentiate between transient and sustained bottlenecks. Eadro [6] integrates traces, logs, and metrics for joint anomaly detection and localization but suffers from interpretability issues due to its multi-model pipeline. MicroCU [5] relies on API logs and Granger causality, despite studies indicating that logs provide limited value in bottleneck detection [6]. Sage [1] models dependencies using a Causal Bayesian Network but assumes that a microservice's latency is dictated by its child nodes, an assumption that breaks down for real-world microservices applications [8].

C. Summary of key gaps in prior works

Most existing works either focus exclusively on single bottlenecks [1], [3], [6], [9], [12], [13], [16], [20] or fail to leverage the rich telemetry and distributed tracing data available in microservices application deployments [23]. Methods that do utilize telemetry often rely on coarse-grained metrics or static thresholds, limiting their adaptability to dynamic workloads and real-time failure patterns. Additionally, many existing approaches are not trained on real-time data, making them ineffective for production environments.

Our framework, GUIDE, aligns with prior works such as GAMMA [15] and ART [17] but introduces key distinctions. While GAMMA performs multi-bottleneck localization, it does not function as a complete incident management tool with an incident trigger warning mechanism. Conversely, ART unifies anomaly detection, failure triage, and root cause localization but does not explicitly handle multiple bottleneck localization and cascading failures within microservices. GUIDE addresses these limitations by integrating real-time telemetry-driven incident warning generation and multi-bottleneck localization, ensuring both proactive incident detection and granular failure analysis within Kubernetes-based microservice deployments.

III. Deployment details and telemetry data

The dataset used in this study was collected from a real-world deployment of microservices pipelines that constitute the *Observe* platform. Briefly, the *Observe* platform ingests machine data via open endpoints and agents, and provides a framework for users to efficiently transform this data into curated temporal datasets that correspond to the logs, metrics, traces, and more for the customer's system, all viewable in a single unified product. The dataset specifically focuses on the `apiserver` and `transformer` microservices, which form a critical part of the platform.

The `apiserver` microservice manages the data query pipeline, which operates with 66 concurrent Go routines, efficiently managing incoming API requests, whereas the `transformer` microservice manages the data transformation pipeline, which executes 70 Go routines to process and optimize data transformations in real-time [11]. The `scheduler` microservice serves both the data query and data transformation pipelines and schedules queries to Snowflake.

The deployment consisted of Kubernetes-managed microservices distributed across multiple VMs, each instrumented with tracing, logging, and system metrics collection. The telemetry data was collected using the OpenTelemetry (OTEL) platform [18] and processed through *Observe*, enabling comprehensive observability and real-time performance monitoring. System-level metrics such as CPU and memory consumption are continuously recorded on the Observe platform via OpenTelemetry, while request-level trace call-graphs are used to provide detailed visibility into service-to-service interactions. Additionally, all collected traces and metrics are segmented into 10 second time windows, allowing for the identification of time-dependent trends in system performance. This windowed segmentation plays a crucial role in detecting bottlenecks as they evolve over time, ensuring that both transient and persistent anomalies are captured.

To ensure the dataset reflected real-world operational challenges, data was collected under varying workloads, includ-

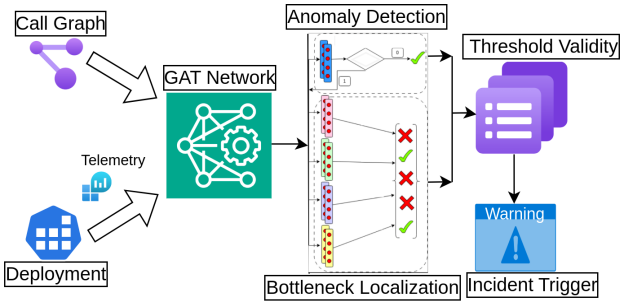


Fig. 1: Architectural overview of GUIDE.

ing normal operations, peak loads, and induced bottlenecks. The bottlenecks were induced via two techniques: (i) reducing the resource availability of critical services such as the `apiserver` to induce localized bottlenecks, and (ii) increasing workload intensity (from 600 to 1800 queries per minute). By combining these techniques, GUIDE generates different types of bottlenecks, including specific microservice bottlenecks that affect services such as the `apiserver` or `transformer`, and systemic bottlenecks that result in cascading failures across multiple services in the pipeline, including the `scheduler`. Further, by pushing the system beyond its standard operational capacity, the propagation of stress through the Kubernetes cluster can be analyzed to determine which microservices experience the most significant performance degradation.

The final dataset consists of 2.4 million traces, with 33% exhibiting bottlenecks, capturing CPU usage, memory consumption, and request propagation patterns. This dataset provides a robust foundation for training and evaluating GUIDE.

IV. Design of GUIDE

Figure 1 illustrates the architecture design of GUIDE. The design of GUIDE is broken down into two distinct components: (i) Anomaly Detection and Bottleneck Localization, and (ii) Incident Trigger. The Incident Trigger relies on a thresholding mechanism, which is abstracted out as the Threshold Validity component in the figure. GUIDE starts by ingesting the call graph of the application along with the deployment details and the telemetry data, as shown in the figure; these are fed to the Graph Attention Network (GAT), which is the core of GUIDE.

A. Architecture of GUIDE for Detection and Localization

The detection and localization component of GUIDE operates in two phases: anomaly detection and bottleneck localization. An initial binary classifier determines if an anomaly exists, while a subsequent localizer ranks microservices by their likelihood of being bottlenecks. By integrating separate classifiers tailored to individual microservices, GUIDE ensures robust and explainable outputs. The model's joint training process minimizes a combined loss function for anomaly detection and localization, leading to efficient identification of multiple, often cascading, bottlenecks. This approach distinguishes GUIDE from traditional methods that treat these tasks separately, enhancing its adaptability to real-world, dynamic scenarios.

At a high-level, GUIDE's architecture uses a novel integration of GATs, temporal embeddings, and an expert classifier framework to target complex, multi-bottlenecked systems. GUIDE processes collected telemetry data through its architecture to predict and localize bottlenecks effectively. GUIDE begins with multi-source temporal embeddings, analyzed via a Dilated Causal Convolution Network (DCC) where it analyzes temporal changes in resource usage to identify patterns indicative of failures. By leveraging DCC embeddings, GUIDE efficiently captures long-range dependencies in time-series data. This capability allows it to detect early warning signs of microservice stress or failure before they escalate into major incidents.

Next, GUIDE constructs dependency graphs from the collected traces, mapping microservice interactions across the Kubernetes cluster. Using GATs, it prioritizes nodes (pods) that contribute most significantly to bottlenecks. GATs are particularly helpful in identifying cascading failures, as they dynamically assign higher attention weights to nodes whose anomalies propagate across the system. This ensures that the analysis reflects real-time issues, making it easier to pinpoint problematic services that disrupt overall system performance.

Then, GUIDE performs detection and localization to determine whether the observed behavior constitutes an anomaly, such as resource contention. In the final stage, the Bottleneck Localizer module employs a set of microservice-specific classifiers to predict whether a microservice is experiencing a bottleneck. These classifiers are trained to recognize patterns indicative of degraded performance based on resource utilization and request handling characteristics. By accurately identifying affected microservices, GUIDE helps engineers diagnose performance issues efficiently and take targeted mitigation actions, minimizing system disruption.

The model design of GUIDE incorporates key hyperparameters that were optimized for performance in detecting and localizing bottlenecks. The temporal convolution layers use DCC with kernel sizes of [3, 3], allowing efficient long-range dependency capture. GATs, with 4 attention heads, enhance the identification of critical microservices, while fully connected layers, with 64 hidden units, process embeddings for detection and localization. The model employs a weighted loss function, where a balancing parameter (λ) controls the trade-off between anomaly detection and bottleneck localization, ensuring both tasks contribute effectively to the training objective.

B. Incident Detection and Warning Paradigm

A key component of GUIDE, that distinguishes it from prior efforts, is the incident warning system. This is achieved by setting an appropriate threshold to trigger incident warnings while minimizing both false positives and false negatives. The goal of this thresholding mechanism is to ensure that genuine bottlenecks are accurately flagged while avoiding unnecessary alerts caused by transient or minor fluctuations in resource availability. To achieve this, GUIDE utilizes a queue-based approach where simulated incidents are sent into a predetermined processing pipeline at a controlled rate. Within this pipeline,

failures are dynamically induced by progressively reducing resource availability and introducing workload variability. By doing so, GUIDE simulates realistic failure conditions and observes how microservices react under varying stress.

The challenge in setting this threshold lies in striking a balance between sensitivity and specificity. To optimize this balance, the system undergoes an iterative calibration process where different threshold values are tested against historical incident data to evaluate their impact on detection accuracy. This calibration process takes into account various factors, including workload variability, microservice dependency structures, and observed failure propagation patterns. In particular, the relationship between leading indicators of failure—such as increasing response latency and declining throughput—and actual incident occurrences is analyzed to determine the most effective threshold setting. By setting the threshold based on real-world observations, GUIDE remains robust to changes in workload dynamics and infrastructure configurations.

To trigger the warning in practice, before flagging an incident, GUIDE continuously monitors service traces within a rolling 2-minute window, analyzing the proportion of traces that exhibit bottleneck characteristics. This window size was determined empirically to be the smallest duration that provides sufficient statistical power while enabling timely incident detection. For each simulated failure scenario, the predefined threshold (discussed above) is applied to determine whether an incident warning should be triggered. If the percentage of traces classified as bottlenecks within the 2-minute window surpasses the predefined threshold, the system generates an incident trigger warning.

V. Evaluation

To evaluate the effectiveness of GUIDE in general, we conducted extensive experiments on a Kubernetes-based deployment of *Observe* (deployment details in Section III). To monitor and analyze the system's behavior, we collect a combination of Prometheus metrics and OpenTelemetry traces. Prometheus metrics track critical system states, including CPU usage, memory consumption, and latency across all microservices. These metrics provided a detailed view of resource utilization and system health during bottlenecks. Simultaneously, the microservices were instrumented to generate distributed OpenTelemetry traces which were collected, enabling the visualization and analysis of request propagation and interdependencies among microservices. Together, these tools provided comprehensive observability into the cluster's performance.

To evaluate the specific incident warning framework, we designed an experiment where service traces are queued and grouped into 10-second windows. These windows are continuously monitored over a 2-minute observation period. If the proportion of bottlenecked windows exceeds the predefined threshold, an incident trigger warning is generated.

Our experiments were conducted on an AWS back-end consisting of `m6i.2xlarge` instances, with a total of 10 nodes, each equipped with 32GB of RAM and 8 Intel Xeon 8375C cores. This infrastructure supports approximately 200 pods, 46 deployments, and 64 microservices, providing a realistic microservices environment for evaluating incident detection and bottleneck localization. We trained GUIDE for 250 epochs on an NVIDIA H100 GPU.

A. Comparison Baselines

To evaluate the empirical performance of GUIDE for anomaly detection and bottleneck localization, we experimentally compare it with three other recent solutions.

1) Seer [3]: applies deep learning to distributed RPC-level tracing to predict and mitigate Quality of Service (QoS) violations in microservices. It detects spatial and temporal patterns in tracing data to localize performance degradation and inform resource management decisions.

2) FIRM [12]: is an ML-based framework for managing resources in SLO-oriented microservices. It employs an SVM-based detection mechanism to identify contention for shared resources and a reinforcement learning agent to optimize resource allocation.

ϵ-Diagnosis [13]: is an unsupervised heuristic-based framework for identifying root causes of small-window long-tail latency issues in microservices. It uses a two-sample hypothesis testing algorithm with ϵ-statistics to detect significant deviations in system metrics, narrowing down root causes from large-scale telemetry data.

B. Evaluation Results

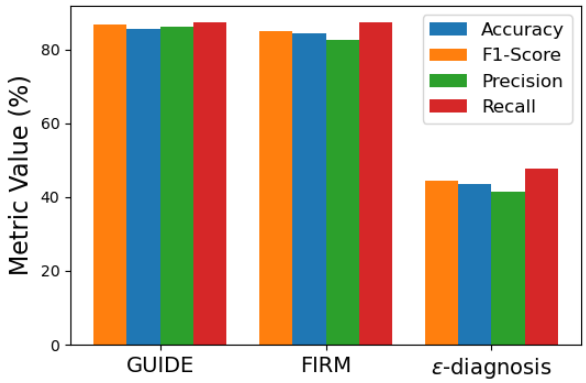
Our evaluation focused on three key aspects: (i) the accuracy of anomaly detection, (ii) the precision of bottleneck localization, and (iii) performance of the incident warning system.

1) Anomaly detection and localization evaluation: Bottleneck localization in GUIDE consists of two tasks: (1) *trace window anomaly detection*, which determines whether a 10 second trace window contains a bottleneck, and (2) *localization*, which identifies the specific microservice functions (e.g., Go-routines) contributing to the performance degradation, for subsequent mitigation efforts.

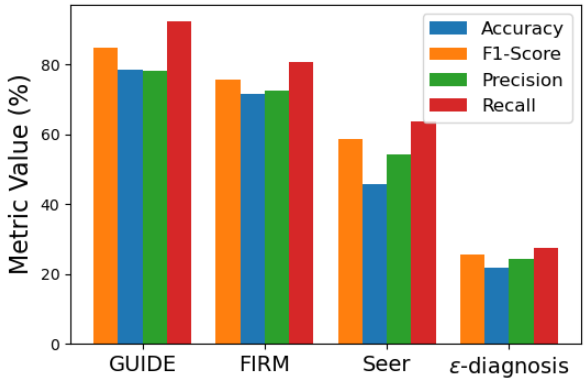
For trace window anomaly detection, shown in Figure 2a, GUIDE achieves an accuracy of **86%** and an F1-score of **87%**, outperforming FIRM (accuracy: 84%, F1-score: 85%) and ϵ-diagnosis (accuracy: 44%, F1-score: 44%).

For bottleneck localization, shown in Figure 2b, GUIDE surpasses other baselines with an accuracy of 78% and an F1-score of 84%, compared to FIRM (72%, 76%), Seer (46%, 59%), and ϵ-diagnosis (22%, 25%); we note that Seer only focuses on localization and not anomaly detection. These results highlight GUIDE's ability to detect and localize bottlenecks more effectively than existing approaches.

2) Incident warning evaluation: For the incident warning system, shown in Figure 3, GUIDE must first determine the threshold value which when exceeded for the proportion of bottlenecked windows, triggers the incident warning (see Section IV-B). This threshold is customizable and can be adjusted based on operational needs, allowing the system to be more sensitive in environments where missing failures could lead to cascading pod failures and broader service disruptions.



(a) Performance evaluation results for anomaly detection.



(b) Performance evaluation results for bottleneck localization.

Fig. 2: Performance comparison of GUIDE against baselines.

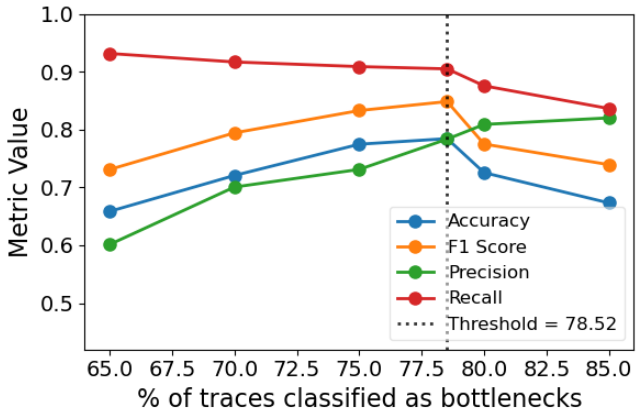


Fig. 3: Impact of threshold value on incident trigger warning performance.

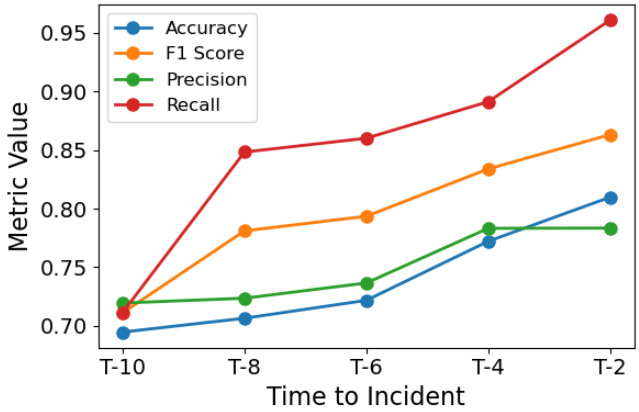


Fig. 4: Performance of incident trigger warning subsystem over a 10-minute observation window.

To determine the optimal threshold, we evaluated multiple values and assessed their impact on detection accuracy, F1-score, precision, and recall. The objective was to identify a threshold that strikes a balance between sensitivity (capturing genuine incidents) and specificity (avoiding false positives due to transient fluctuations). Our results indicated that a threshold of 78.5% provided the best trade-off between sensitivity and specificity (referring to the point where the sensitivity and specificity curves intersect). At this threshold, we also achieve the highest accuracy (78%) and F1-score (85%), while maintaining competitive precision (78%) and recall (92%); see Figure 3. Lower thresholds (e.g., 65%) led to higher recall (93%) but significantly lower precision (60%), resulting in frequent false positives. Conversely, higher thresholds (e.g., 85%) increased precision (82%) but at the cost of reduced recall (84%), leading to undetected failures.

To assess the reliability of our trigger warnings, we evaluated their accuracy over a 10-minute observation window leading up to an incident. The system analyzes service traces at different time intervals (T-10, T-8, T-6, T-4, T-2), where T represents the actual moment of failure and T−x represents x minutes before the failure. Note that at T−x, GUIDE has access to the preceding 2 minutes (T−x−2 to T−x) of data.

Our results, shown in Figure 4, indicate that accuracy increases gradually from 69% at T-10 to 81% at T-2, reflecting the increasing difficulty of early-stage detection. Similarly, the F1-score improves as the system gets closer to the failure event, rising from 71% at T-10 to 86% at T-2. The system maintains a competitive balance between precision and recall, with precision ranging from 78% at T-2 to 72% at T-10, while recall is highest at T-2 (96%) and lowest at T-10 (71%). Overall, across the 10-minute observation window, ***GUIDE achieves an F1-score of 0.85, with a precision of 0.78 and a recall of 0.92***. These results highlight the system's ability to generate early trigger warnings with high accuracy while maintaining sensitivity to evolving failure patterns.

C. Impact on Mitigation and Incident Resolution Time

C. Impact on Mitigation and Incident Resolution Time A key advantage of GUIDE is its ability to reduce the time required for engineers to diagnose and mitigate incidents. Prior to deploying GUIDE, incident resolution often required extensive manual investigation. When integrated with production systems, GUIDE can incorporate a binary feedback

mechanism, allowing operators to validate or reject detection results, which can enable continuous improvement of the model through active learning techniques.

Given the large-scale nature of modern microservices environments, the computational efficiency of GUIDE was a critical evaluation criterion. GUIDE is designed to operate with minimal overhead while processing millions of traces in real-time. The end-to-end inference time for incident detection and localization averaged **0.2 seconds per prediction**, making it feasible for deployment in high-throughput production environments. The inference times can be further reduced by processing larger batch sizes. In comparison, simple rule-based approaches exhibited lower computational overhead but at the cost of significantly reduced accuracy and adaptability.

VI. Conclusion

GUIDE advances existing research on bottleneck detection and localization in microservices by leveraging real-time distributed tracing and telemetry data. Our evaluation results, using real-world traces collected from a production cloud-native system, demonstrate that GUIDE outperforms existing baselines (including FIRM and Seer) achieving an F1-score of 87% for trace window anomaly detection— 98% higher than existing heuristic-based methods. For bottleneck localization, GUIDE surpasses all baselines we compared with, with an F1-score of 85%, 11% higher than FIRM and, over 234% higher than ϵ-Diagnosis.

Compared to prior works, GUIDE not only excels at multi-bottleneck localization but also introduces an integrated incident warning mechanism. Our practical and robust incident trigger warning system effectively predicts failures in advance, with an accuracy of 81%, while demonstrating that the system gains sensitivity as incidents approach. Future work will explore ML-based techniques for threshold optimization to further improve incident warning accuracy and reduce manual calibration requirements. These results highlight GUIDE's efficacy as a scalable, real-time incident management tool for modern microservices deployments.

References

[1] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. Sage: Practical and Scalable ML-Driven Performance Debugging in Microservices. ASPLOS '21, pages 135–151, 2021.

[2] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. ASPLOS '19, pages 3–18, 2019.

[3] Yu Gan, Yanqi Zhang, Kelvin Hu, Dailun Cheng, Yuan He, Meghna Pancholi, and Christina Delimitrou. Seer: Leveraging Big Data to Navigate the Complexity of Performance Debugging in Cloud Microservices. ASPLOS '19, pages 19–33, 2019.

[4] Vipul Harsh, Wenxuan Zhou, Sachin Ashok, Radhika Niranjan Mysore, Brighten Godfrey, and Sujata Banerjee. Murphy: Performance Diagnosis of Distributed Cloud Applications. SIGCOMM '23, pages 438–451, 2023.

[5] Xinrui Jiang, Yicheng Pan, Meng Ma, and Ping Wang. Look Deep into the Microservice System Anomaly through Very Sparse Logs. WWW '23, pages 2970–2978, 2023.

[6] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R. Lyu. Eadro: An End-to-End Troubleshooting Framework for Microservices on Multi-Source Data. ICSE '23, pages 1750–1762, 2023.

[7] Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, and Dan Pei. Unsupervised Detection of Microservice Trace Anomalies through Service-Level Deep Bayesian Networks. ISSRE '20, pages 48–58, 2020.

[8] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis. SoCC '21, pages 412–426, 2021.

[9] Meng Ma, Jingmin Xu, Yuan Wang, Pengfei Chen, Zonghua Zhang, and Ping Wang. AutoMAP: Diagnose Your Microservice-Based Web Applications Automatically. WWW '20, pages 246–258, 2020.

[10] Hoa Xuan Nguyen, Shaoshu Zhu, and Mingming Liu. A survey on graph neural networks for microservice-based cloud applications. *Sensors*, 22(23), 2022.

[11] Daniel Odievich. How observe uses snowflake to deliver the observability cloud, March 2024.

[12] Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. FIRM: An Intelligent Fine-grained Resource Management Framework for SLO-Oriented Microservices. OSDI '20, pages 805–825, 2020.

[13] Huasong Shan, Yuan Chen, Haifeng Liu, Yunpeng Zhang, Xiao Xiao, Xiaofeng He, Min Li, and Wei Ding. ϵ-Diagnosis: Unsupervised and Real-Time Diagnosis of Small- Window Long-Tail Latency in Large-Scale Microservice Platforms. WWW '19, pages 3215–3222, 2019.

[14] Jacopo Soldani and Antonio Brogi. Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey. *ACM Comput. Surv.*, 55(3), Feb 2022.

[15] Gagan Somashekar, Anurag Dutt, Mainak Adak, Tania Lorido Botran, and Anshul Gandhi. GAMMA: Graph Neural Network-Based Multi-Bottleneck Localization for Microservices Applications. WWW '24, pages 3085–3095, 2024.

[16] Gagan Somashekar, Anurag Dutt, Rohith Vaddavalli, Sai Bhargav Varanasi, and Anshul Gandhi. B-MEG: Bottlenecked-Microservices Extraction Using Graph Neural Networks. ICPE '22, pages 7–11, 2022.

[17] Yongqian Sun, Binpeng Shi, Mingyu Mao, Minghua Ma, Sibo Xia, Shenglin Zhang, and Dan Pei. ART: A Unified Unsupervised Framework for Incident Management in Microservice Systems. ASE '24, pages 1183–1194, 2024.

[18] Aadi Thakur and M. B. Chandak. A review on opentelemetry and http implementation. *International journal of health sciences*, 6(S2):15013–15023, Jun. 2022.

[19] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[20] Hanzhang Wang, Zhengkai Wu, Huai Jiang, Yichao Huang, Jiamu Wang, Selcuk Kopru, and Tao Xie. Groot: An Event-Graph-Based Approach for Root Cause Analysis in Industrial Settings. ASE '21, pages 419–429, 2022.

[21] Zhe Xie, Haowen Xu, Wenxiao Chen, Wanxue Li, Huai Jiang, Liangfei Su, Hanzhang Wang, and Dan Pei. Unsupervised Anomaly Detection on Microservice Traces through Graph VAE. WWW '23, pages 2874–2884, 2023.

[22] Chenxi Zhang, Xin Peng, Chaofeng Sha, Ke Zhang, Zhenqing Fu, Xiya Wu, Qingwei Lin, and Dongmei Zhang. DeepTraLog: Trace-Log Combined Microservice Anomaly Detection through Graph-based Deep Learning. ICSE '22, pages 623–634, 2022.

[23] Chenxi Zhang, Xin Peng, Tong Zhou, Chaofeng Sha, Zhenghui Yan, Yiru Chen, and Hong Yang. TraceCRL: Contrastive Representation Learning for Microservice Trace Analysis. ESEC/FSE '22, pages 1221–1232, 2022.

[24] Zhizhou Zhang, Murali Krishna Ramanathan, Prithvi Raj, Abhishek Parwal, Timothy Sherwood, and Milind Chabbi. CRISP: Critical Path Analysis of Large-Scale Microservice Architectures. ATC '22, pages 655–672, 2022.