LabelEase: A Semi-Automatic Tool for Efficient and Accurate Trace Labeling in Microservices

Shenglin Zhang[†‖], Zeyu Che[†], Zhongjie Pan[†], Xiaohui Nie[‡], Yongqian Sun[*‡‡†], Lemeng Pan[§], Dan Pei[¶]
[†]Nankai University, {zhangsl, sunyongqian}@nankai.edu.cn, {chezeyu, zhongjie}@mail.nankai.edu.cn
[‡]Computer Network Information Center, Chinese Academy of Sciences, xhnie@cnic.cn
[§]Huawei, panlemeng@huawei.com
[¶]Tsinghua University, Beijing National Research Center for Information Science and Technology, peidan@tsinghua.edu.cn
[‖]Haihe Laboratory of Information Technology Application Innovation
[‡‡]Tianjin Key Laboratory of Software Experience and Human Computer Interaction

***Abstract*—Trace data is crucial for system observability and maintainability within microservices architectures, and many operation algorithms depend heavily on trace data, including anomaly detection, root cause analysis, *etc*. However, the actual performance of these algorithms might be unsatisfactory due to the absence of high-quality labeled datasets for effective training and evaluation. Since billions of traces could be generated daily for large-scale microservices, labeling overhead is the main hurdle to obtaining high-quality trace datasets.**

In this paper, we propose *LabelEase*, a novel semi-automatic trace labeling tool, which uses active learning techniques to achieve efficient and accurate trace labeling. For anomaly trace labeling, *LabelEase* clusters similar traces with a graph-based trace representation technique and selects a few representative traces for human labeling, avoiding labeling most of the traces. For root cause labeling, *LabelEase* aggregates the labeled anomalous traces and identifies the service's failures for operators to label. Our systematic experiments on two large-scale datasets show that *LabelEase* achieves over 0.98 F_1-score in anomaly trace labeling and 0.89 precision of failure detection in root cause labeling, *LabelEase* can reduce operators' labeling overhead by more than 99.9%. To the best of our knowledge, we are the first to propose a semi-automatic trace labeling tool capable of achieving efficient and accurate trace labeling.

***Index Terms*—Data labeling tool, Trace anomaly detection, Root cause localization**

I. Introduction

Nowadays microservices architecture has emerged as a leading paradigm for building modern software systems, decomposing applications into a collection of loosely coupled, independently deployable services [1]–[3]. Large-scale microservice systems could contain tens of thousands of service modules [4]. Trace data of microservices is crucial for understanding system behavior, diagnosing performance issues, and ensuring reliability. By analyzing traces, operators can gain insights into service interactions, identify bottlenecks, detect anomalies, and optimize system performance [5]–[7].

Many anomaly detection and root cause analysis algorithms based on trace data have been proposed for efficient system maintenance [3], [8]–[11]. However, evaluating and selecting the appropriate algorithm for application in practice is a challenging task. On the one hand, there are only a few public trace datasets (*e.g.*, train ticket dataset [12], NetManAIOps dataset [13]) can be found, which are not only limited in the number of traces but also in the types of faults. On the other hand, it should be evaluated by the data specific to that scenario, since the characteristics of trace data vary across different scenarios. Furthermore, although many current anomaly detection and root cause analysis algorithms are unsupervised or semi-supervised, the availability of high-quality labeled data could aid in their training process and potentially enhance their performance. Unfortunately, there is currently no efficient tool available for trace labeling. Therefore, the development of an efficient and accurate trace labeling tool is an urgent need. Such datasets would greatly benefit trace anomaly detection and root cause analysis research and practice, similar to the impact of ImageNet [14] on computer vision and deep learning.

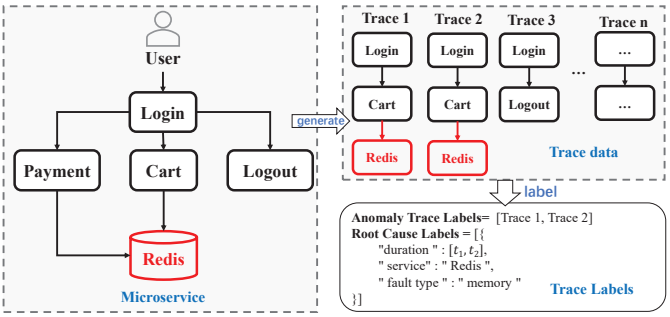

Fig. 1. An toy example of the trace labeling process for a simple e-commerce microservice system. A memory fault happened at the Redis service, the trace labels should contain the anomaly traces and root cause information of Redis.

Generally, in combination with human expertise, operators manually label all traces to acquire a high-quality dataset. Fig. 1 shows a toy example of the trace labeling process. Labeling trace involves examining the trace data to identify anomalous traces and summarizing their root causes. Unfortunately, completely manually labeling a trace dataset is not a simple task. Previous manual trace labeling methods require observing traces' call structure and latency characteristics individually and comparing them with similar requests.

* Yongqian Sun is the corresponding author.

Therefore, manually labeling trace data for system analysis requires significant time and labor. Due to the huge amount of the trace dataset and the complex structural information of the traces, operators cannot manually label the entire dataset. State-of-the-art unsupervised anomaly detection models are not up to the task, which will be discussed in § V-B.

To reduce the overhead of manual labeling, *our goal is to build a semi-automatic trace labeling tool to help operators label anomaly traces and root causes efficiently and accurately.* Three challenges are encountered in building this tool:

1) ***Massive traces***: Microservice systems could generate a massive amount of trace data every day for labeling. For instance, within eBay's infrastructure, the microservices system generates approximately 150 billion traces on a daily basis [15]. It is impossible to label all of the traces manually.
2) ***Complex trace structures***: Traces are characterized by complex and rich structures [16]. How to model the traces is a problem. A trace exhibits a complex structure arising from the hierarchy of service invocations, as well as parallel and asynchronous invocations [6]. Typically, a trace manifests as a complex graph structure and may encompass numerous service operations, ranging from dozens to hundreds in number [10].
3) ***Hard-to-determine root cause***: The root cause is challenging to label even after trace anomalies are identified. Generally, root cause labeling needs to consider multiple traces over a while. It is challenging to check all the anomalous trace data and conclude the root cause information.

In this paper, we propose *LabelEase*, a semi-automatic tool for efficient and accurate trace labeling in microservices. (1) To address the first challenge, we use active learning, which allows us to obtain a high-quality trace dataset by labeling only a small number of traces. After labeling the anomalies of all the traces, we aggregate the anomalous period traces to visually provide reference information for operators to label the root cause. (2) To address the second challenge, we train a graph neural network to encode all trace data, ensuring that similar traces possess similar vector representations. (3) To address the third challenge, we design a trace aggregation and root cause localization technique to aggregate the anomalous traces and identify failure periods for operators to label. Our contributions can be summarized as follows:

1) To the best of our knowledge, this is the first work to focus on reducing the labeling workload for trace labeling. It integrates algorithmic approaches with operators' expertise and can assist operators in labeling trace anomalies efficiently and accurately.
2) We propose *LabelEase*, a novel tool that consists of three components: graph-based trace representation, anomaly labeling, and root cause labeling. It clusters massive amounts of trace data into a few groups and selects the most representative traces for operators to label, which significantly reduces the label overhead.
3) We conduct a series of experimental studies to evaluate *LabelEase*'s effectiveness and efficiency using two datasets. The results show that our approach labels trace anomalies in the microservices system with an average F_1-score of 0.99 and 0.98 with an extraordinarily small amount of traces that need to be labeled, respectively.
4) A high-quality dataset on trace anomaly detection is published, which not only identifies whether each trace is anomalous or normal but also shows the root cause of each anomalous trace. Moreover, we have made our source code of *LabelEase* and the labeled trace dataset openly accessible [1], helping readers better understand our work.

II. Preliminary

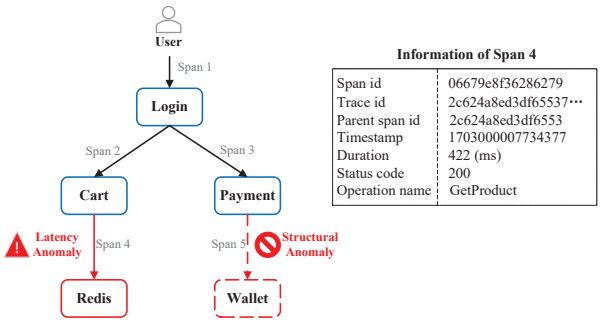

Fig. 2. An example of the trace in microservice system.

A. Microservice System

Microservice system is a software development approach characterized by decomposing applications into small, independent services [17]. This architectural decomposition facilitates enhanced flexibility, scalability, and maintainability, rendering the microservice system an increasingly favored choice among organizations striving to deliver robust and resilient software solutions [18].

For instance, the microservice system shown in Fig. 2 consists of five distinct services denoted, including "Login", "Cart", "Redis", "Payment" and "Wallet". Through the decoupling of services, the microservice system enables independent development, deployment, and scaling, empowering organizations to promptly respond to evolving business needs and technological advancements [19].

B. Trace

A trace represents the sequence of service interactions in response to a single user request or transaction in the microservice system [20]. For example, Fig. 2 shows a trace of the microservice system, consisting of Span 1 through 5.

A span refers to a specific unit of work or operation within a trace [15]. For instance, as shown in Fig. 2, the span contains essential information such as the timestamp, service name, operation name, duration, *etc*.

Trace anomalies can be categorized into two primary types: latency anomaly and structural anomaly. For example, as shown in Fig. 2, when "Cart" calls "Redis", Span 4 has

[1]https://doi.org/10.5281/zenodo.13338156

a relatively high duration, resulting in a latency anomaly. Furthermore, if the anticipated invocation sequence between "Payment" and "Wallet" fails to materialize within the trace, it indicates a disruption, *i.e.*, a structure anomaly occurs.

C. Anomaly Detection and Root Cause Localization

Anomaly detection identifies potential anomalies in service execution and generates timely alerts [21]. It reports abnormal traces, offering context for further troubleshooting. By examining these traces, one can investigate the upstream and downstream services associated with the anomaly's location.

Root cause localization identifies the components affected by a particular fault. It involves isolating the issue to the specific component or subsystem responsible for the failure and then conducting a detailed examination to determine the exact sources of errors [22]. This process is indispensable for effectively diagnosing and resolving issues inherent to complex and distributed systems.

III. LabelEase Approach

A. Design Overview

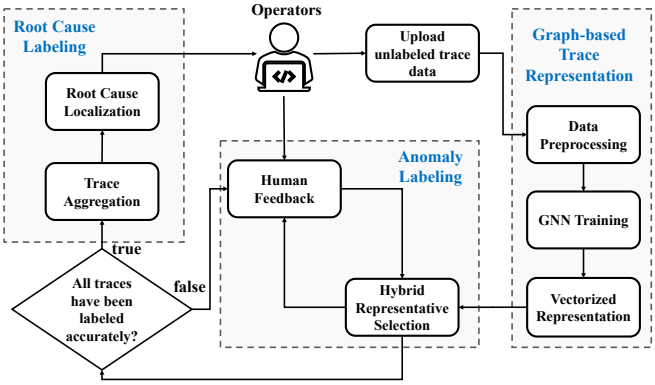

Fig. 3. The overview framework of *LabelEase*.

The overall framework of *LabelEase* is illustrated in Fig. 3. It comprises three principal components: the graph-based trace representation module, the anomaly labeling module, and the root cause labeling module. This tool enables operators to conduct anomaly labeling and root cause identification on raw traces, thereby producing a high-quality trace dataset.

In the graph-based trace representation module, the unlabeled trace data uploaded by operators is first preprocessed into graph-structured data. Subsequently, we train a Graph Neural Network (GNN) to differentiate between various traces based on their representations. The trained model then encodes the trace data into representation vectors.

In the anomaly labeling module, we utilize a hybrid representative selection approach to cluster the trace representation vectors into multiple classes. Operators then employ their expertise to identify anomalies at the center points of each cluster. With human feedback, active learning is applied to label the remaining traces accurately.

In the root cause labeling module, traces from abnormal intervals are aggregated. Operators can more effectively determine the root cause by analyzing the frequency of access to each service and examining the associated calling information.

B. Graph-based Trace Representation

1) Data Preprocessing: During the trace analysis process, we primarily focus on the semantic information (*e.g.*, service name and operation name), temporal features, status codes within the span, and the calling relationships between different services. We convert the raw trace data into a graph data format following the methodology outlined by Zhang et al. [23].

For semantic information within a span, we use common separators in microservice systems (*e.g.*, '/', '-', ':') to split the textual information. All words are then converted to lowercase, and non-character tokens (*e.g.*, punctuation marks, numbers) are removed. For out-of-vocabulary (OOV) or new compound words, we employ WordPiece tokenization [24], which involves breaking down words into individual characters and iteratively identifying the most common character combinations until the desired vocabulary is achieved. Subsequently, we utilize the feature extraction capabilities of a pre-trained BERT model to obtain semantic vectors [25], representing the span's service and operation names.

For time features within a span, the start time and duration of a service invocation provide valuable insights into the performance and behavior of the microservice system. By analyzing the duration, waiting time, local execution time, and relative start time of each span [23], we can comprehensively understand service interactions and performance within the system. The high dimensionality of the previous semantic encoding can lead to the neglect of time features in span representation. Additionally, the substantial variation in duration across different traces complicates model training convergence. To address these issues, we consolidate these four time-related features into a vector for time information extraction and project them into a high-dimensional vector [26].

Status codes within a span pertain to the numeric codes employed by servers in response to client requests within the HTTP protocol. These codes are typically classified into five categories: informational, success, redirection, client error, and server error codes, with each code representing a distinct outcome. For instance, the code "200" indicates a successful request. Incorporating status codes aids in comprehending the outcomes of requests and facilitates the adoption of appropriate actions. In our work, we encode the status codes using one-hot encoding methodology.

Intuitively, a trace exhibits a complex hierarchical structure of calling relationships, rendering it suitable for representation as a directed acyclic graph. Within this graph, each node corresponds to a span, while the edges denote the parent-child relationship between spans. The vector representation of each span encompasses three fundamental components of information encoding: semantic information, time features, and status code in span.

2) GNN Training: We employ GNN to capture the structural and relational intricacies within the trace data. Upon training the GNN, we utilize it to encode each trace into a fixed-size vector representation. This representation succinctly encapsulates the trace's salient features while encoding its inherent structure and inter-span relationships. Specifically,

we utilize the Graph Attention Network (GAT) [27], a GNN variant grounded on the multi-head self-attention mechanism. GAT offers a more adaptive and expressive approach to learning representations from graph-structured data.

Graph Contrastive Learning (GCL) represents a standard methodology for training GNN. The primary objective of GCL is to incentivize the model to converge representations of similar graphs towards each other within the embedding space while simultaneously driving representations of dissimilar graphs apart.

To achieve similar vector representations for analogous traces, we train the GNN utilizing preprocessed traces while integrating trace-specific domain knowledge, as outlined in He et al. [28]. The model's input comprises a triplet consisting of three traces (denoted as A, B, and C), intending to predict their relative relationships. Specifically, the task entails discerning that trace A bears greater similarity to trace B than to trace C.

At first, following He et al. [28], we produce a large number of trace triplets. We mainly focus on the structural differences and time latency between different traces. On the one hand, we generate for each trace a path ID that can uniquely identify a calling path to determine whether the traces have the same structure [15]. The same structure implies that every node position in two traces must have the same span names and caller-callee relations. On the other hand, the trace latency represents the duration of end-to-end requests. Intuitively, there are different time latencies between normal traces and abnormal traces.

Initially, the GNN independently encodes each trace, yielding representations G_A, G_B, and G_C for traces A, B, and C, respectively, with shared parameters across all traces. By learning their relative similarities, GNN training aims to accurately capture trace representations by encoding similarity information. The similarity for measuring the distance of two traces, $d(\boldsymbol{x}, \boldsymbol{y})$, is computed with cosine similarity by Equation 1:

$$d(\boldsymbol{x}, \boldsymbol{y}) = 1 - \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|} \tag{1}$$

We train the model with the triplet loss [29] to minimize the distance between G_A and G_B while maximizing the distance between G_A and G_C. The loss function $L(A, B, C)$ is defined by Equation 2:

$$L(A, B, C) = \max\{d(G_A, G_B) - d(G_A, G_B) + margin, 0\} \tag{2}$$

where $margin$ is a fixed value to control the gap of two distances. By minimizing the triplet loss via backpropagation, the GNN's ability to represent and distinguish different traces is gradually satisfying.

3) Vectorized Representation: Leveraging the trained GNN, we exploit its capacity to generate diverse vector representations for individual traces. By preserving each trace's intricate structure and attribute information, similar traces exhibit closer proximity within the vector space.

We input preprocessed graph data representing each trace into the trained GNN. This process generates a 64-dimensional vectorized representation for each trace. These representations encapsulate each trace's distinctive features, facilitating subsequent analysis and comparison tasks.

C. Anomaly Labeling Module

At this stage, the unlabeled trace data undergo a labeling process incorporating operators' knowledge and expertise. Our objective is to minimize the labeling effort while ensuring high-quality data labels. In this regard, active learning [30], a machine learning paradigm, is particularly well-suited. Active learning seeks to optimize performance gains while minimizing the number of labeled samples required for training. It selects the most informative data points from an unlabeled dataset for labeling through human feedback. It reduces labeling costs while preserving performance levels.

Following graph-based trace representation, all traces underwent vectorization via GNN. Consequently, for certain similar traces, selecting the most representative trace for labeling suffices to ensure the correct labeling of other traces within the same cluster. We partitioned traces into different classes utilizing clustering algorithms. For the traces in each class, the cluster center point is selected as the most representative trace for operators to label.

In practical terms, if an operator opts to label k traces, the trace data will be clustered into k distinct clusters using a clustering algorithm. Intuitively, as human feedback and the number of labeled traces increase, it is anticipated that the accuracy of the labeled dataset will progressively converge towards 100%. We will discuss this in § V-E.

However, with the expansion of total data size and the increasing number of labeled traces, the clustering process may incur significant space and time overhead. Traditional clustering algorithms often struggle to handle large-scale data efficiently [31]. To address this challenge, we draw insights from Huang et al. [32] and employ hybrid representative selection. Specifically, we sample one-tenth of the data from the entire dataset, ensuring that the distribution characteristics of the sampled data in the vector space remain consistent with those of the original dataset. Subsequently, we apply the k-means method to derive k clusters, utilizing the resulting k cluster centers as the set of representatives.

We provide operators with a visual representation of the topology diagram illustrating the interactive relationships between spans. Simultaneously, to ensure a precise understanding of span delay and its severity, we offer a corresponding timeline for each span, depicting its temporal latency and calling level. In summary, *LabelEase* facilitates operators in intuitively and clearly labeling trace anomalies.

D. Root Cause Labeling

Once all trace data are labeled, the subsequent task involves identifying the root cause of traces during abnormal periods. This process entails partitioning the trace data into abnormal periods based on the outcomes of abnormal labeling.

A service characterized by a higher frequency of abnormal traces and a lower occurrence of normal traces passing through

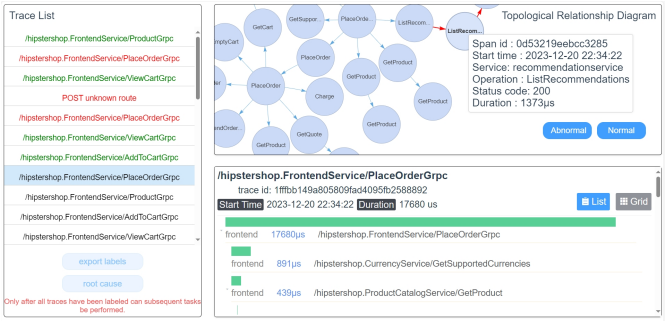

Fig. 4. Interface of trace anomalies labeling

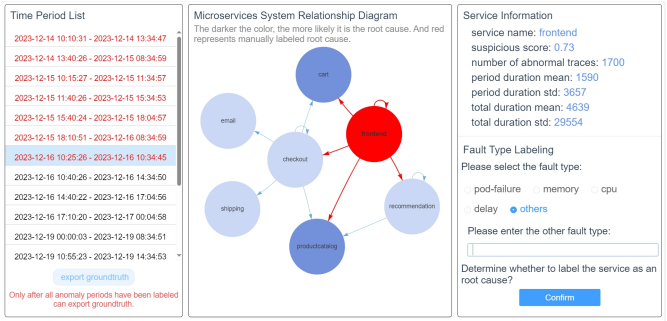

Fig. 5. Interface of root cause labeling

it is more likely to be identified as the root cause service [2], [33]. To ascertain the likelihood of a service being the root cause, we employ the spectrum-based fault localization (SBFL) technique. This method computes the suspicious score of each service within a service set P based on the presence of normal and abnormal traces. In SBFL, we focus primarily on four distinct statistics for a given service $O \in P$: O_{ef}, O_{ep}, O_{nf}, and O_{np}. Here, O_{ef} represents the count of abnormal traces covering service O, while O_{ep} signifies the count of normal traces covering service O. Conversely, O_{nf} denotes the count of abnormal traces not covering service O, and O_{np} indicates the count of normal traces not covering service O.

Based on the above statistics, we leverage Equation 3 to calculate suspicious score $Score_O$ of service O, and prior work has proved its effectiveness [34].

$$Score_O = \frac{O_{ef}}{\sqrt{(O_{ef} + O_{ep}) * (O_{ef} + O_{nf})}} \tag{3}$$

Nevertheless, spectrum analysis solely accounts for abnormal and normal traces for each service, disregarding service latency and dependency relationships. Consequently, it may fail to accurately identify the true culprits in scenarios where multiple services with closely interdependent relationships are implicated in anomalous requests [34].

Henceforth, we integrate the visualization of service dependency topology and service latency information on the front end. It aims to provide operators with an intuitive perceptual experience, facilitating rapid labeling tasks. By presenting these insights, we strive to better leverage operators' professional knowledge to enable actionable root cause localization.

IV. Implementation

We have seamlessly integrated the *LabelEase* approach into a bespoke labeling tool. Fig. 4 shows the interface of trace anomalies labeling and Fig. 5 shows the interface of root cause labeling.

A. Trace Anomalies Labeling

After *LabelEase* selects representative traces for labeling, the operators will label the given traces on the page shown in Fig. 4. It presents trace and span-related information in a visual and intuitive manner, encompassing structural topology diagrams, duration, status codes, operation names, hierarchical relationships, *etc*. Additionally, *LabelEase* also provides the average value, standard deviation, and other information about each similar call path in the entire dataset as a labeling reference. In providing this comprehensive environment, *LabelEase* facilitates operators in efficiently labeling anomalies.

In the trace list displayed on the left side of the page, each trace is visually distinguished: green signifies a label of "normal", red indicates a label of "abnormal", and black denotes traces awaiting labeling. Operators can label each trace by selecting either the "abnormal" or "normal" buttons, guided by structural information at the top right and delay information at the bottom right of the page. Operators can customize the number of traces to be labeled according to the situation that will be discussed in § V-E. This adaptive approach ensures efficient and tailored labeling processes to suit varying circumstances.

Upon completion of labeling for the provided traces, *LabelEase* can automatically label the remaining unlabeled traces, reducing the workload of manual labeling. Moreover, all trace labels can be exported directly with extremely high accuracy.

B. Root Cause Labeling

After all traces are labeled, operators will go to the page of labeling root cause in Fig. 5. *LabelEase* will merge the periods when the faults occurred based on the time when the labeled anomaly trace occurred. Traces within each period will then be aggregated. Leveraging SBFL, *LabelEase* will compute a suspicious score for each server.

However, SBFL overlooks important factors such as service latency and service dependencies. To address this limitation, the interface will incorporate additional information, including the average latency of the aggregated traces for passing through each service and the invocation relationship between the services in each period, *etc*. Meanwhile, the average latency and standard deviation of traces passing through each service across the entire dataset will be calculated as a reference for labeling information. As a result, operators can utilize their initiative and professional knowledge to correctly label the root cause of anomalous traces.

For each period, a microservice system relationship diagram is featured prominently in the center of the page. The higher the suspicious score, the more likely the root cause is, and the darker the service will be displayed on the interface. Upon selecting a service, detailed information about it is displayed in the right box. Operators can set the type of fault to label the root cause. Then, the service labeled as the root cause is

highlighted in red. Eventually, the dataset's ground truth can be exported, and this high-quality trace dataset can be used for other scientific tasks.

V. Evaluation

In this section, we first introduce the experimental setup of *LabelEase*. Then, we conduct extensive experiments to evaluate the performance of *LabelEase* and aim to answer the following research questions (RQs) :

RQ1. How does the effectiveness and efficiency of *LabelEase* compare to baseline methods in terms of trace anomaly detection?

RQ2. How effective is *LabelEase* in graph-based trace representation compared with baseline trace vectorization approaches?

RQ3. How well does the hybrid representative selection with active learning compare to other clustering methods?

RQ4. How do the number of traces operators need to label influence the performance?

RQ5. How effective is labeling the root cause?

A. Experiment Setup

Datasets: We conducted experiments to evaluate the performance of *LabelEase* using two datasets, denoted as Dataset 1 ($\mathscr{D}1$) and Dataset 2 ($\mathscr{D}2$). The traces and root causes within these datasets were meticulously labeled by professional operators. $\mathscr{D}1$ is collected from a benchmark microservice system that we developed, utilizing the open-source e-commerce application Online Boutique [35]. This dataset includes user activities such as browsing items, adding items to the cart, and making purchases. Various faults were deliberately injected into the system using Chaos Mesh [36], such as pod failure, delay, packet loss, memory issues, *etc*. We use a set of 103078 normal traces and 27285 anomaly traces to conduct experiments. $\mathscr{D}2$ is collected from a large-scale microservice system operated by a top-tier global commercial bank. It includes services such as account managements, bill payments, fund transfers and personal financial plans. This dataset emulates a real-world production environment of a large-scale life service application, incorporating end-to-end full-link logs, multivariate time series (MTS), and traces. We utilize a total of 112,786 traces from this dataset, comprising 19,207 anomaly traces, to evaluate the performance of anomaly labeling of *LabelEase*.

Environment and Hyperparameters: All experiments are run on a server with two 16C32T Intel(R) Xeon(R) Gold 5218 CPU @ 2.30 GHz, one NVIDIA(R) Tesla(R) V100S, and 192 GB RAM. The implementation of *LabelEase* is in Python 3.7.0, with PyTorch 1.5.0 serving as the primary deep learning framework. During the GNN training phase, we employed a batch size of 32, a learning rate of 0.0001, and conducted training over 15 epochs.

Baselines: To evaluate the performance of *LabelEase* on anomaly labeling, we employ six recently proposed trace unsupervised anomaly detection methods as baseline methods, including MultimodalTrace [37], TraceAnomaly [8], CRISP [9], TraceCRL [10], TraceVAE [11] and TraceSieve [3]. These unsupervised algorithms use the given trace data for training and labeling to suit our application scenario of labeling traces. By comparing with *LabelEase*, we aim to evaluate whether these state-of-the-art anomaly detection models exhibit similar proficiency in dataset labeling even if numerous anomalous traces are present in the training data.

Performance Metrics: In the root cause labeling task, operators are provided with a visually enriched labeling environment, facilitating rapid and accurate labeling. Operators' professional expertise is pivotal in locating root causes, with their subjective initiative often guiding the process. Ideally, root cause labeling aims for 100% accuracy. Consequently, we focus on evaluating the effectiveness of anomaly detection labeling for the entire dataset, assuming the correctness of trace anomalies labeled by operators. The task of labeling trace anomalies can be framed as a binary classification for each trace, with the F_1score serving as a widely accepted performance metric, which is given by Equation 4. Here, TP represents true positives, FP represents false positives, and FN represents false negatives. $Precision$ denotes the proportion of predicted positive samples that are true positive samples. $Recall$ denotes the proportion of true positive samples that are correctly predicted as positive. Consequently, F_1score provides a balanced assessment by computing the harmonic mean of precision and recall.

$$\begin{aligned} Precision &= \frac{TP}{TP+FP} \\ Recall &= \frac{TP}{TP+FN} \\ F_1-score &= 2 \times \frac{Precision \times Recall}{Precision + Recall} \end{aligned} \tag{4}$$

B. LabelEase vs. Baseline Algorithms (RQ1)

The results of trace anomaly labeling of *LabelEase* and the baseline approaches are shown in Table I. *LabelEase* exhibits superior performance over all baseline approaches across both datasets, achieving F_1 scores of 0.99 and 0.98, respectively. Notably, this performance is attained with operators labeling only 35 traces in $\mathscr{D}1$ and 67 traces in $\mathscr{D}2$.

Among the baseline methods, MultimodalTrace achieves the lowest performance. MultimodalTrace trains an LSTM-based model based on the operation sequences of traces, primarily utilizing recorded timestamps for reconstruction. However, its feature extraction process is relatively simplistic and limited, leading to inadequate discrimination between normal and abnormal data.

TraceAnomaly and CRISP utilize a service trace vector (STV) representation to characterize traces and detect anomalies through Variational Autoencoder (VAE) reconstruction. However, they solely incorporate service sequences and response times from traces, neglecting the status code information. Due to the presence of more normal data and simpler service calls on $\mathscr{D}1$ than on $\mathscr{D}2$, TraceAnomaly and CRISP can learn more accurate normal pattern features on $\mathscr{D}1$, achieve higher precision than on $\mathscr{D}2$. This variance in

dataset complexity accounts for the differences in performance between $\mathscr{D}1$ and $\mathscr{D}2$.

TraceCRL constructs an operation invocation graph for each trace and employs a contrastive learning mechanism to train a graph neural network-based model. However, while TraceCRL learns the features of traces through graph structure and data augmentation strategies, proper feature selection is still crucial for anomaly detection. If the model fails to capture all the features associated with anomalies, it may lead to missed or false alarms.

TraceVAE employs a dual-variable graph VAE architecture and an innovative dispatching layer to separately encode the structure of traces and the time consumption of nodes. TraceVAE proposes techniques to reduce the entropy gap, but the effectiveness may vary depending on specific data distributions and anomaly types. Hence, adaptation or further optimization may be necessary in different application scenarios.

TraceSieve integrates the Variational Graph Auto-Encoder (VGAE) with Elastic Weight Consolidation (EWC) to formulate an unsupervised trace anomaly detection method. It utilizes an auto-encoder architecture within an adversarial training framework to filter out noise data. Thereby, TraceSieve is more accurate in identifying anomalies and reduces the number of false alarms with high precision on $\mathscr{D}1$. Unfortunately, the sensitivity of noise filtering impacts the features and patterns learned by the model. Excessive sensitivity may cause the model to overlook the key information, misclassify real data as noise, and fail to filter noise effectively. Consequently, this affects the model's generalization ability and prediction performance, leading to a subpar performance on $\mathscr{D}2$.

In summary, the results presented in Table I underscore the inadequacy of existing unsupervised trace anomaly detection methods in effectively labeling the dataset. The subpar performance of these baseline approaches can be attributed primarily to the substantial presence of anomalous data in the training set, hindering the models from learning accurate feature representations. Consequently, none of these methods can accurately differentiate between normal and anomalous traces.

Regarding efficiency, we evaluate the time cost of trace anomalies labeling with all baseline methods and *LabelEase*. As depicted in Table I, we present the time overhead incurred during the testing and labeling processes with the trained baseline models. *LabelEase* exhibits superior efficiency compared to the baseline methods, boasting the shortest time overhead and highest labeling efficiency. The baseline methods are characterized by intricate neural network architectures, which entail complex forward propagation operations, resulting in significant time overhead. In contrast, *LabelEase* leverages active learning, relying on the efficient hybrid representative selection, and acquires all trace labels through human feedback. As a result, *LabelEase* outperforms other baseline methods with extremely high effectiveness and efficiency.

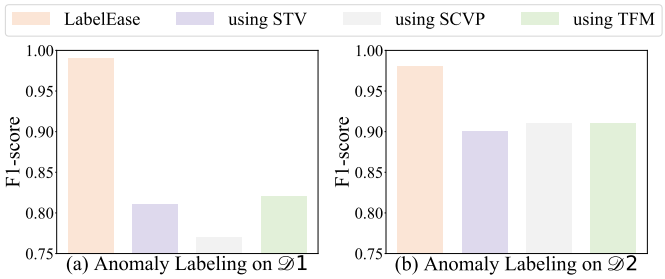

Fig. 6. The effects of graph-based trace representation in comparison with different trace vectorization approaches on two datasets

C. The Effects of Graph-based Trace Representation (RQ2)

We conduct a series of experiments to evaluate the effects of graph-based trace representation on two datasets compared to other trace vectorization approaches. Fig. 6 shows *LabelEase* with the graph-based representation can outperform other trace vectorization methods.

In *LabelEase* using STV, we change the graph-based trace representation into STV [8]. A specific trace's STV is constructed with the call path list. Within this representation, the value of each dimension signifies the response time associated with a particular invocation path for a given trace. Notably, when a dimension's value is -1, it signifies the absence of this specific call path. However, STV may not capture the intricate interactions and dependencies prevalent in complex systems. Consequently, it risks overlooking crucial invocation information between spans, potentially undermining model performance or leading to erroneous outcomes.

In *LabelEase* using SCVP, the graph-based trace representation is replaced with service critical path vectors (SCVP) [9]. CRISP encodes only on the call paths for those on the critical path spans. Compared with the STV, SCPV encoding reduces the feature dimensions and decreases the inference time. However, this selective encoding strategy raises concerns. First, by exclusively considering call paths on the critical path, SCPV may overlook essential information, particularly regarding paths not on the critical path that could nonetheless impact system performance. Second, accurate identification and definition of critical paths are imperative for SCPV. This often necessitates domain knowledge and experience and potentially relies on assumptions about the system's structure and operation.

In *LabelEase* using TFM, we replace the graph-based trace representation used in the existing method with trace feature matrix (TFM) [3]. TFM stores extracted features, encompassing both execution and waiting times from traces. However, despite TFM's inclusion of additional temporal features, it remains deficient in adequately capturing the invocation relationships between services and the dependency associations between spans.

In *LabelEase*, we leverage trained GNN to generate a vectorized representation for each trace to measure the distance between different traces. The graph-based trace representation approach promises to accurately capture the intricate structure and attributes of traces in a microservice system. Conse-

TABLE I
THE EFFECTS OF *LabelEase* IN COMPARISON WITH DIFFERENT APPROACHES ON TWO DATASETS

Approach	$\mathscr{D}1$				$\mathscr{D}2$			
	Precision	Recall	F_1-score	Time	Precision	Recall	F_1-score	Time
LabelEase	1	0.98	0.99	6.53s	0.96	0.99	0.98	21.68s
MultimodalTrace [37]	0.2	0.15	0.17	1.9min	0.17	0.15	0.16	1.6min
TraceAnomaly [8]	0.94	0.67	0.78	27.2min	0.21	0.2	0.2	22.2min
CRISP [9]	0.8	0.57	0.67	26.1min	0.24	0.21	0.23	17.3min
TraceCRL [10]	0.39	0.28	0.33	8.9h	0.48	0.43	0.45	7.2h
TraceVAE [11]	0.4	0.3	0.35	1.7h	0.14	0.77	0.23	51.3min
TraceSieve [3]	1	0.74	0.85	7.6min	0.17	0.15	0.15	9.8min

quently, adopting this approach enables the acquisition of richer and more comprehensive trace information, potentially leading to improved F_1-score.

D. Effectiveness of Clustering Strategies (RQ3)

We leverage the hybrid representative selection strategy to select the most representative data for active learning. In particular, after filtering out a part of the trace collection refer to [32], clustering methods are employed to identify cluster centroids. Our experimental evaluation encompasses four classical clustering algorithms to determine the most effective and efficient approach for this task.

K-means is an iterative clustering algorithm that partitions data into k clusters by minimizing the sum of squared distances from data points to cluster centroids. Hierarchical clustering constructs a hierarchical structure by evaluating the similarity between objects. DBSCAN, a density-based clustering method, groups closely situated points together based on a defined minimum number of points within a specified radius. Spectral clustering partitions data into clusters by analyzing the eigenvectors of a similarity matrix derived from the data.

Table II presents the performance and efficiency of *LabelEase* employing various clustering algorithms. The results indicate generally satisfactory performance across most clustering methods evaluated. Notably, the K-means algorithm exhibits superior effects and the shortest computational time among the considered algorithms, indicating heightened efficiency compared to alternatives. Consequently, we opt for the K-means algorithm as the preferred clustering method for *LabelEase*.

In addition, we conducted a comparison by randomly selecting traces for labeling to assess the effectiveness of the hybrid representative selection strategy. Table II demonstrates the worst results of using random selection. As traces are chosen randomly as representatives, their selection time is not factored into the analysis. On the one hand, after using graph-based trace representation in the previous step, similar traces have similar embedding representations, so the precision of *LabelEase* using random selection is not low. On the other hand, the proportion of normal traces is relatively large, and random selection makes it easy to select normal samples, resulting in inferior recall. In conclusion, adopting random selection to replace clustering shows unsatisfactory results, underscoring the importance of accurately selecting the most representative trace data.

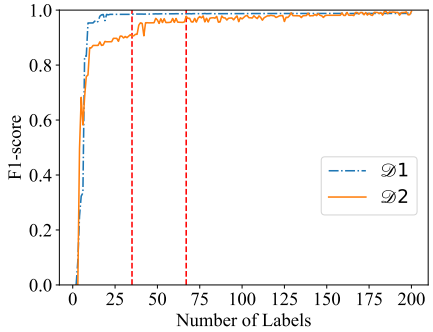

Fig. 7. The effect of different numbers of traces to be labeled

E. Sensitivity of the Number of Traces to be Labeled (RQ4)

We focus on the impact of the sensitivity of different numbers of traces to be labeled on the overall labeling results with *LabelEase*. Fig. 7 shows how the F_1-score of *LabelEase* changes with different trace numbers of labels. In *LabelEase*, We employ K-means to cluster the data into k clusters, selecting the trace at the center of each cluster for manual labeling. Operators can determine the number of traces to be labeled, significantly enhancing flexibility and initiative.

As the number of labels increases, the F_1 score of *LabelEase* improves. However, It also leads to higher time and cost overhead. Therefore, to balance the labeling workload and achieve satisfactory results, referring to the red dotted lines in Fig. 7, we choose 35 as the number of labels for $\mathscr{D}1$ and 67 as the number of labels for $\mathscr{D}2$. In this case, the F_1-score on the two datasets will be 0.99 and 0.98, respectively.

The number of traces to be labeled can be increased if higher performance is required. As shown in Fig. 7, the F_1-score grows slowly and gradually converges to 1 when the number of labels reaches a certain threshold. Our experimental evaluation results indicate that for any dataset, operators can use 0.1% of the total number of traces as the labeling threshold. By adopting this threshold, we demonstrate the universality of *LabelEase*, significantly reducing operators' labeling overhead by more than 99.9% while acquiring a high-quality dataset.

F. Effectiveness of Labeling Root Cause (RQ5)

After labeling all anomalies, we evaluate the effectiveness of root cause localization. We use *LabelEase* to aggregate anomaly traces and determine the period during which faults occurred. Since operators' expertise is employed for labeling, we believe they have correctly identified the root cause for

TABLE II
THE EFFECTS OF THE DIFFERENT CLUSTER METHODS ON TWO DATASETS

Approach	$\mathscr{D}1$				$\mathscr{D}2$			
	Precision	Recall	F_1-score	Time(s)	Precision	Recall	F_1-score	Time(s)
using K-means	1	0.98	0.99	6.53	0.96	0.99	0.98	21.68
using hierarchical clustering	0.99	0.98	0.99	96.9	0.96	0.95	0.96	51.98
using DBSCAN	0.99	0.92	0.95	116.07	0.83	1	0.91	97.55
using Spectral clustering	0.99	0.98	0.98	200.84	0.96	0.93	0.94	264.24
using random selection	0.8	0.2	0.32	-	0.95	0.33	0.49	-

each fault period. Consequently, it is essential to ensure that all fault periods in the dataset are accurately identified.

Since $\mathscr{D}2$ doesn't contain abundant fault cases, we only used $\mathscr{D}1$ for the experiments. We compared the aggregated fault periods identified by *LabelEase* with the ground truth labeled in the dataset. To evaluate the effectiveness of *LabelEase* in detecting fault periods, we used point-wise PA, which can give an inflated score if some anomaly segments persist for a long duration [38]. The experimental results show that the precision of fault period detection is 0.89. This indicates that as long as an abnormal trace can be labeled, it is possible to accurately and efficiently identify the period in which the corresponding fault occurred, thus enabling precise root cause labeling.

However, there are many false negatives when locating the fault period, with a recall of 0.44. This discrepancy arises from two main issues. Firstly, anomalies in the unreported fault periods are not reflected in the traces but primarily in the multivariate time series (MTS). Secondly, in $\mathscr{D}1$, the unreported periods predominantly occur during pod failures. Other faults, such as "loss", "delay", "CPU", and "memory", do not crash the program but degrade its performance. These faults still allow traces to be recorded, with anomalies evident in latency and call patterns. In this case, if the pod failure occurs, the anomalies will be reflected in traces only when the root cause is on the recommendation service instance. Therefore, we conclude that the factors affecting trace anomalies are the type of fault and the instance where it is in effect.

VI. Related work

Distributed tracing plays a crucial role in the microservice system. In recent years, various open-source distributed trace recording infrastructures have been developed and integrated into the modern microservice system, including Jaeger [39], Zipkin [40] and SkyWalking [41]. As fundamental tools, they are used to troubleshoot programs but cannot label data.

In other areas of operations and maintenance within the microservice system, Zhao et al. [42] introduces a semi-automatic labeling tool *Label-Less* for the MTS. It employs robust and rapid anomaly similarity search to save operators from scanning and checking the long KPIs back and forth for abnormal patterns or label consistency, improving labeling efficiency. However, there is a lack of such a labeling tool to promote the field of trace analysis in the microservice system.

Although numerous trace anomaly detection [3], [8]–[11], [37] and root cause localization algorithms [2], [3], [8], [33], [34], [43] have been proposed, none can reduce the workload of trace data labeling and still obtain a high-quality dataset. Therefore, a semi-automatic, trace-specific labeling tool that can reduce costs while achieving high-quality labels is urgently needed.

To our knowledge, *LabelEase* represents the initial effort in this direction. Serving as the inaugural semi-automatic labeling tool, it is designed to mitigate the labeling overhead associated with trace datasets through algorithmic approaches.

VII. Conclusion

In this paper, we argue that traces with labels hold significant importance in training and evaluating trace anomaly detection and root cause localization. However, manual trace labeling will generate a considerable workload and cost overhead due to the extensive volume and the rich and complex structure of traces. To address this issue, we propose a semi-automatic trace labeling tool *LabelEase*. We evaluate *LabelEase* using two datasets, achieving F_1-score of 0.99 and 0.98 with a small number of traces manually labeled, respectively. We reduce the workload of manual labeling by at least 99.9% while ensuring the acquisition of the high-quality trace dataset. Furthermore, we also exhibit that employing unsupervised trace anomaly detection models for training and labeling raw data is not feasible.

In future work, we aim to leverage transfer learning to optimize model training and reduce the time cost of model training before the labeling process. Additionally, we intend to deploy *LabelEase* onto a server, thus enabling accessibility for individuals requiring its functionalities online. We believe that access to high-quality trace datasets will surely contribute to the rapid development of intelligent AIOps in academia and industry.

VIII. Acknowledgement

This work is supported by the Natural Science Foundation of China (62272249, 62302244, 62072264).

References

[1] T. Bi, Y. Pan, X. Jiang, M. Ma, and P. Wang, "Vecrosim: A versatile metric-oriented microservice fault simulation system (tools and artifact track)," in *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2022, pp. 297–308.

[2] Z. Li, J. Chen, R. Jiao, N. Zhao, Z. Wang, S. Zhang, Y. Wu, L. Jiang, L. Yan, Z. Wang *et al.*, "Practical root cause localization for microservice systems via trace analysis," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 2021, pp. 1–10.

[3] S. Zhang, Z. Pan, H. Liu, P. Jin, Y. Sun, Q. Ouyang, J. Wang, X. Jia, Y. Zhang, H. Yang *et al.*, "Efficient and robust trace anomaly detection for large-scale microservice systems," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 69–79.

[4] D. Liu, C. He, X. Peng, F. Lin, C. Zhang, S. Gong, Z. Li, J. Ou, and Z. Wu, "Microhecl: High-efficient root cause localization in large-scale microservice systems," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2021, pp. 338–347.

[5] 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*, 2021, pp. 76–91.

[6] C. Zhang, X. Peng, C. Sha, K. Zhang, Z. Fu, X. Wu, Q. Lin, and D. Zhang, "Deeptralog: Trace-log combined microservice anomaly detection through graph-based deep learning," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 623–634.

[7] Z. Zeng, Y. Zhang, Y. Xu, M. Ma, B. Qiao, W. Zou, Q. Chen, M. Zhang, X. Zhang, H. Zhang *et al.*, "Traceark: Towards actionable performance anomaly alerting for online service systems," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2023, pp. 258–269.

[8] P. Liu, H. Xu, Q. Ouyang, R. Jiao, Z. Chen, S. Zhang, J. Yang, L. Mo, J. Zeng, W. Xue *et al.*, "Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 48–58.

[9] Z. Zhang, M. K. Ramanathan, P. Raj, A. Parwal, T. Sherwood, and M. Chabbi, "{CRISP}: Critical path analysis of {Large-Scale} microservice architectures," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022, pp. 655–672.

[10] C. Zhang, X. Peng, T. Zhou, C. Sha, Z. Yan, Y. Chen, and H. Yang, "Tracecrl: contrastive representation learning for microservice trace analysis," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 1221–1232.

[11] Z. Xie, H. Xu, W. Chen, W. Li, H. Jiang, L. Su, H. Wang, and D. Pei, "Unsupervised anomaly detection on microservice traces through graph vae," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 2874–2884.

[12] 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.

[13] Z. Li, N. Zhao, S. Zhang, Y. Sun, P. Chen, X. Wen, M. Ma, and D. Pei, "Constructing large-scale real-world benchmark datasets for aiops," *arXiv preprint arXiv:2208.03938*, 2022.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[15] X. Guo, X. Peng, H. Wang, W. Li, H. Jiang, D. Ding, T. Xie, and L. Su, "Graph-based trace analysis for microservice architecture understanding and problem diagnosis," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1387–1397.

[16] C. Zhang, Z. Dong, X. Peng, B. Zhang, and M. Chen, "Trace-based multi-dimensional root cause localization of performance issues in microservice systems," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–12.

[17] Y. Cai, B. Han, J. Li, N. Zhao, and J. Su, "Modelcoder: A fault model based automatic root cause localization framework for microservice systems," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 2021, pp. 1–6.

[18] V. Velepucha and P. Flores, "A survey on microservices architecture: Principles, patterns and migration challenges." *IEEE Access*, 2023.

[19] P. Vitharana and S. A. Daya, "Adopting and sustaining microservice-based software development: Organizational challenges can be more difficult than technical ones." *Communications of the ACM*, 2024.

[20] Z. Xie, C. Pei, W. Li, H. Jiang, L. Su, J. Li, G. Xie, and D. Pei, "From point-wise to group-wise: A fast and accurate microservice trace anomaly detection approach," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 1739–1749.

[21] B. Li, X. Peng, Q. Xiang, H. Wang, T. Xie, J. Sun, and X. Liu, "Enjoy your observability: an industrial survey of microservice tracing and analysis," *Empirical Software Engineering*, vol. 27, pp. 1–28, 2022.

[22] P. Notaro, J. Cardoso, and M. Gerndt, "A survey of aiops methods for failure management," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 12, no. 6, pp. 1–45, 2021.

[23] K. Zhang, C. Zhang, X. Peng, and C. Sha, "Putracead: Trace anomaly detection with partial labels based on gnn and pu learning," in *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2022, pp. 239–250.

[24] M. Schuster and K. Nakajima, "Japanese and korean voice search," in *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2012, pp. 5149–5152.

[25] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 492–504.

[26] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 92–103.

[27] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[28] S. He, B. Feng, L. Li, X. Zhang, Y. Kang, Q. Lin, S. Rajmohan, and D. Zhang, "Steam: Observability-preserving trace sampling," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 1750–1761.

[29] G. Chechik, V. Sharma, U. Shalit, and S. Bengio, "Large scale online learning of image similarity through ranking." *Journal of Machine Learning Research*, vol. 11, no. 3, 2010.

[30] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, B. B. Gupta, X. Chen, and X. Wang, "A survey of deep active learning," *ACM computing surveys (CSUR)*, vol. 54, no. 9, pp. 1–40, 2021.

[31] Y. Zhou, "Scalable clustering: Large scale unsupervised learning of gaussian mixture models with outliers," Ph.D. dissertation, The Florida State University, 2023.

[32] D. Huang, C.-D. Wang, J.-S. Wu, J.-H. Lai, and C.-K. Kwoh, "Ultra-scalable spectral clustering and ensemble clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 6, pp. 1212–1226, 2019.

[33] G. Yu, P. Chen, H. Chen, Z. Guan, Z. Huang, L. Jing, T. Weng, X. Sun, and X. Li, "Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments," in *Proceedings of the Web Conference 2021*, 2021, pp. 3087–3098.

[34] G. Yu, Z. Huang, and P. Chen, "Tracerank: Abnormal service localization with dis-aggregated end-to-end tracing data in cloud native systems," *Journal of Software: Evolution and Process*, vol. 35, no. 10, p. e2413, 2023.

[35] "Online boutique." [Online]. Available: https://github.com/GoogleCloudPlatform/microservices-demo

[36] "Chaos mesh," 2022. [Online]. Available: https://chaos-mesh.org

[37] S. Nedelkoski, J. Cardoso, and O. Kao, "Anomaly detection from system tracing data using multimodal deep learning," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 179–186.

[38] H. Si, C. Pei, H. Cui, J. Yang, Y. Sun, S. Zhang, J. Li, H. Zhang, J. Han, D. Pei *et al.*, "Timeseriesbench: An industrial-grade benchmark for time series anomaly detection models," *arXiv preprint arXiv:2402.10802*, 2024.

[39] Jaegertracing.io, "Jaeger," 2022, last accessed 1 August 2022. [Online]. Available: https://www.jaegertracing.io/

[40] Twitter, "Zipkin," 2022, last accessed 1 August 2022. [Online]. Available: https://zipkin.io/

[41] A. SkyWalking, "Skywalking," 2022, last accessed 30 July 2022. [Online]. Available: https://skywalking.apache.org/

[42] N. Zhao, J. Zhu, R. Liu, D. Liu, M. Zhang, and D. Pei, "Label-less: A semi-automatic labelling tool for kpi anomalies," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1882–1890.

[43] Y. Gan, M. Liang, S. Dev, D. Lo, and C. Delimitrou, "Sage: practical and scalable ml-driven performance debugging in microservices," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 135–151.