2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) | 978-1-6654-5701-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSE48619.2023.00150

Eadro: An End-to-End Troubleshooting Framework for Microservices on Multi-source Data

Cheryl Lee*, Tianyi Yang*, Zhuangbin Chen*, Yuxin Su†, and Michael R. Lyu*
*Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China.
Email: cheryllee@link.cuhk.edu.hk, {tyyang, zbchen, lyu}@cse.cuhk.edu.hk
†Sun Yat-sen University, Guangzhou, China. Email: suyx35@mail.sysu.edu.cn

***Abstract*—The complexity and dynamism of microservices pose significant challenges to system reliability, and thereby, automated troubleshooting is crucial. Effective root cause localization after anomaly detection is crucial for ensuring the reliability of microservice systems. However, two significant issues rest in existing approaches: (1) Microservices generate traces, system logs, and key performance indicators (KPIs), but existing approaches usually consider traces only, failing to understand the system fully as traces cannot depict all anomalies; (2) Troubleshooting microservices generally contains two main phases, i.e., anomaly detection and root cause localization. Existing studies regard these two phases as independent, ignoring their close correlation. Even worse, inaccurate detection results can deeply affect localization effectiveness. To overcome these limitations, we propose *Eadro*, the first end-to-end framework to integrate anomaly detection and root cause localization based on multi-source data for troubleshooting large-scale microservices. The key insights of Eadro are the anomaly manifestations on different data sources and the close connection between detection and localization. Thus, Eadro models intra-service behaviors and inter-service dependencies from traces, logs, and KPIs, all the while leveraging the shared knowledge of the two phases via multi-task learning. Experiments on two widely-used benchmark microservices demonstrate that Eadro outperforms state-of-the-art approaches by a large margin. The results also show the usefulness of integrating multi-source data. We also release our code and data to facilitate future research.**

***Index Terms*—Microservices, Root Cause Localization, Anomaly Detection, Traces**

I. Introduction

Microservice systems are increasingly appealing to cloud-native enterprise applications for several reasons, including resource flexibility, loosely-coupled architecture, and lightweight deployment [1]. However, anomalies are inevitable in microservices due to their complexity and dynamism. An anomaly in one microservice could propagate to others and magnify its impact, resulting in considerable revenue and reputation loss for companies [2]. Figure 1 shows an example where a failure in one microservice may delay all microservices on the invocation chain.

Therefore, developers must closely monitor the microservice status via run-time information (e.g., traces, system logs, and KPIs) to discover and tackle potential failures in their earliest efforts. Yet, thousands of microservices are usually running in distributed machines in a large-scale industrial microservice system. As each microservice can launch multiple instances, a system can produce billions of run-time records per day [1], [2]. The explosion of monitoring data makes automated troubleshooting techniques imperative.

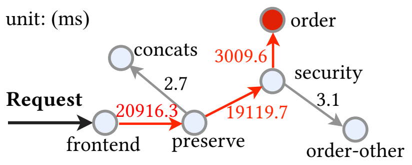


Fig. 1. A failure in "order" indirectly delays other microservices on the invocation chain, while microservices off the chain are unaffected.

Many efforts have been devoted to this end, focusing either on anomaly detection [3]–[5] or on root cause localization [6]–[11]. Anomaly detection tells whether an anomaly exists, and root cause localization identifies the culprit microservice upon the existence of an anomaly. Previous approaches usually leverage statistical models or machine learning techniques to mine information from traces, as traces profile and monitor microservice executions and record essential inter-service information (e.g., request duration). However, we identify two main limitations of the existing troubleshooting approaches.

(1) *Insufficient exploitation of monitoring data*: different from operation teams that pay close attention to diverse sources of run-time information, existing research deeply relies on traces and exploits other data sources insufficiently. This gap stems from the complexity of multi-source data analysis, which is much harder than single-source data analysis, as multi-source data is heterogeneous, frequently interacting, and very large [12]. However, on the one hand, traces contain important information for troubleshooting but are insufficient to reveal all typical types of anomalies. On the other hand, different types of data, such as logs and KPIs, can reveal anomalies collaboratively and bring more clues about potential failures. For example, a CPU exhaustion fault can cause abnormally high values in the CPU usage indicator and trigger warnings recorded in logs, but the traces may not exhibit abnormal patterns (such as high latency).

(2) *Disconnection in closely related tasks*: Generally, root cause localization follows anomaly detection since we must discover an anomaly before analyzing it. Current studies of microservice reliability regard the two phases as independent, despite their shared inputs and knowledge about the

Yuxin Su is the corresponding author.

microservice status. Existing approaches usually deal with the same inputs redundantly and waste the rich correlation information between anomaly detection and root cause localization. Moreover, the contradiction between computing efficiency and accuracy limits the simple combination of state-of-the-art anomaly detectors and root cause localizers. For a two-stage troubleshooting approach, it is generally a little late to use an advanced anomaly detector and then analyze the root cause. Thus, root cause localization-focused studies usually apply oversimplified anomaly detectors (e.g., N-sigma), and unfortunately, the resulting detection outputs can contain many noisy labels and thereby affect the effectiveness of downstream root cause localization.

To overcome the above limitations, we propose **Eadro**, the first End-to-end framework integrating Anomaly Detection and Root cause lOcalization to troubleshoot microservice systems based on multi-source monitoring data. The key ideas are 1) learning discriminative representations of the microservice status via multi-modal learning and 2) forcing the model to learn fundamental features revealing anomalies via multi-task learning. Therefore, Eadro can fully exploit meaningful information from different data sources that can all manifest anomalies. Also, it allows information to be inputted once and used to tackle anomaly detection and root cause localization together and avoids incorrect detection results hindering next-phase root cause localization.

Specifically, Eadro consists of three components: *(1) Modal-wise learning* contains modality-specific modules for learning intra-service behaviors from logs, KPIs, and traces. We apply Hawkes process [13] and a fully connected (FC) layer to model the log event occurrences. KPIs are fed into a dilated causal convolution (DCC) layer [14] to learn temporal dependencies and inter-series associations. We also use DCC to capture meaningful fluctuations of latency in traces, such as extremely high values. *(2) Dependency-aware status learning* aims to model the intra- and inter-dependencies between microservices. It first fuses the multi-modal representations via gated concentration and feeds the fused representation into a graph attention network (GAT), where the topological dependency is built on historical invocations. *(3) Joint detection and localization* contains an anomaly detector and a root cause localizer sharing representations and an objective. It predicts the existence of anomalies and the probability of each microservice being the culprit upon an anomaly alarm.

Experimental results on two datasets collected from two widely-used benchmark microservice systems demonstrate the effectiveness of Eadro. For anomaly detection, Eadro surpasses all compared approaches by a large margin in *F1* (53.82%~92.68%), and also increases *F1* by 11.47% on average compared to our derived multi-source data-based methods. For root cause localization, Eadro achieves state-of-the-art results with 290%~5068% higher in *HR@1* (Top-1 Hit Rate) than five advanced baselines and outperforms our derived methods by 43.06% in *HR@1* on average. An extensive ablation study further confirms the contributions of modeling different data sources.

Our main contributions are highlighted as follows:

- We identify two limitations of existing approaches for troubleshooting microservices, motivated by which we are the first to explore the opportunity and necessity to integrate anomaly detection and root cause localization, as well as exploit logs, KPIs, and traces together.
- We propose the first end-to-end troubleshooting framework (Eadro) to jointly conduct anomaly detection and root cause localization for microservices based on multi-source data. Eadro models intra-service behaviors and inter-service dependencies.
- We conduct extensive experiments on two benchmark datasets. The results demonstrate that Eadro outperforms all compared approaches, including state-of-the-art approaches and derived multi-source baselines on both anomaly detection and root cause localization. We also conduct ablation studies to further validate the contributions of different data sources.
- Our code and data [1] are made public for practitioners to adopt, replicate or extend Eadro.

II. Problem Statement

This section introduces important terminologies and defines the problem of integrating anomaly detection and root cause localization with the same inputs.

A. Terminologies

Traces record the process of the microservice system responding to a user request (e.g., click "create an order" on an online shopping website). Different microservice instances then conduct a series of small actions to respond to the request. For example, the request "create an order" may contain steps "create an order in pending", "reserve credit", and "update the order state." A microservice (caller) can *invoke* another microservice (callee) to conduct the following action (e.g., microservice "Query" asks microservice "Check" to check the order after finishing the action "query the stock of goods"), and the callee will return the result of the action to the caller. We name this process as *invocation*. The time consumed by the whole invocation (i.e., from initializing the invocation to returning the result) is called invocation *latency*, including the request processing time inside a microservice and the time spent on communicating between the caller and the callee. A *trace* records the information during processing a user request [15] (including multiple invocations), such as the invocation latency, the total time of processing the request, the HTTP response code, etc.

Meanwhile, system logs are generated when system events are triggered. A *log message* (or *log* for short) is a line of the standard output of logging statements, composed of constant strings (written by developers) and variable values (determined by the system) [16]. If the variable values are removed, the remaining constant strings constitute a *log event*. *KPIs* are the numerical measurements of system performance (e.g., disk I/O

[1] https://github.com/BEbillionaireUSD/Eadro

rate) and the usage of resources (e.g., CPU, memory, disk) that are sampled uniformly.

B. Problem Formulation

Consider a large-scale system with M microservices, system logs, KPIs, and traces are aggregated individually at each microservice. In a T-length observation window (data obtained in a window constitute a sample), we have multi-source data defined as $\mathbf{X} = \{(\mathbf{X}_m^{\mathcal{L}}, \mathbf{X}_m^{\mathcal{K}}, \mathbf{X}_m^{\mathcal{T}})\}_{m=1}^{M}$, where at the m-th microservice, $\mathbf{X}_m^{\mathcal{L}}$ represents the log events chronologically arranged; $\mathbf{X}_m^{\mathcal{K}}$ is a multivariate time series consisting of k indicators; $\mathbf{X}_m^{\mathcal{T}}$ denotes the trace records. Our work attempts to build an end-to-end framework achieving a two-stage goal: Given $\mathbf{X}_{[1:M]}$, the framework predicts the existence of anomalies, denoted by y, a binary indicator represented as 0 (normal) or 1 (abnormal). If y equals one, a localizer is triggered to estimate the probability of each microservice to be the culprit, denoted by $\mathbf{P} = [p_1 \cdots p_M] \in [0,1]^M$. The framework is built on a parameterized model $\mathcal{F} : \mathbf{X} \rightarrow (y, \mathbf{P})$.

III. Motivation

This section introduces the motivation for this work, which aims to address effective root cause localization by jointly integrating an accurate anomaly detector and being driven by multi-source monitoring data. The examples are taken from data collected from a benchmark microservice system, TrainTicket [17]. Details about data collection will be introduced in § V-A.

A. Can different sources of data besides traces be helpful?

We find that *traces are insufficient to reveal all potential faults despite their wide usage.* Most, if not all, previous related works [3], [4], [7], [18]–[21] are trace-based, indicating traces are informative and valuable. However, traces focus on recording interactions between microservices and provide a holistic view of the system in practice. Such high-level information only enables basic queries for coarse-grained information rather than intra-service information. For example, latency or error rate in traces can suggest a microservice's availability, yet fine-grained information like memory usage reflecting the intra-service status is unknowable. This is consistent with our observation that latency is sensitive to network-related issues but cannot adequately reflect resource exhaustion-related anomalies. Figure 2 shows an example where a point denotes an invocation taking the microservice "travel" as the callee. When Network Jam or Packet Loss is injected, the latency is abnormally high (marked with stars), but the latency during the CPU exhaustion injection period does not display obviously abnormal patterns. This case reminds us to be careful of relying on traces only. Since traces are informative but cannot reveal all anomalies, trace-based methods may omit potential failures. We need extra information to mitigate the anomaly omission problem.

We also notice that *system logs and KPIs provide valuable information manifesting anomalies in microservices.*

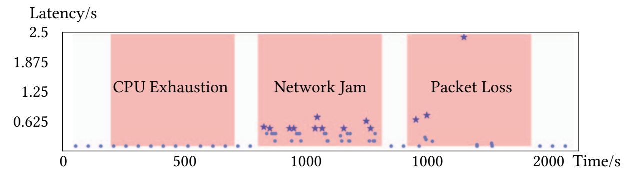


Fig. 2. Network-related faults incur obvious anomalies in latency of "travel", but the CPU exhaustion fault does not.

As for logs, we first parse all logs into events via Drain [22], a popular log parser showing effectiveness in many studies [16], [23]. It is evident that some logs can report anomalies semantically by including keywords such as "exception", "fail", and "errors". The event "Exception in monitor thread while connecting to server <*>." can be a good example.

Event occurrences can also manifest anomalies besides semantics. Take the event "Route id: <*>" recorded by the microservice "route" as an example. This event occurs when the microservice completes the routing request. Figure 3 shows that when network-related faults are injected, the example event's occurrence experiences a sudden drop and remains at low values. The reason is that the routing invocations become less since the communication between "route" and its parent microservices (callers) is blocked. This case further supports our intuition that system logs can provide clues about microservice anomalies.

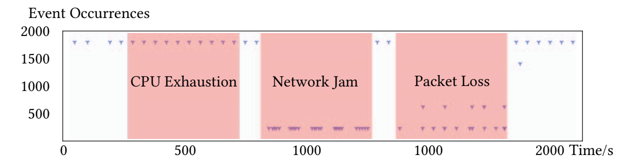


Fig. 3. The occurrences of related logs can reflect issues such as poor communication.

KPIs are responsive to anomalies by continuously recording run-time information. An example in Figure 4 gives a closer look, which displays "total CPU usage" of microservice "payment" during the period covering fault injections. Clearly, "total CPU usage" responds to the fault CPU Exhaustion by showing irregular jitters and abnormally high values. This observation aligns with our a priori knowledge that KPIs provide an external view of a microservice's resource usage and performance. Their fine-grained information can well reflect anomalies, especially resource-related issues, which require detailed analysis.

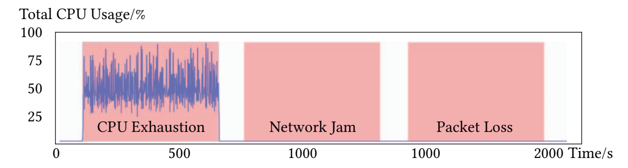


Fig. 4. A CPU exhaustion fault incurs abnormal jitters and high values in "total CPU usage".

However, only using logs and KPIs is not sufficient since they are generated by each microservice individually at a local level. As the example shown in Figure 1 (§ I), we need traces to obtain inter-service dependencies to analyze the anomaly propagation so as to draw a global picture of the system to locate the root cause.

Traces are informative yet not sufficient to reflect all anomalies. System logs and metrics provide valuable information manifesting anomalies by presenting abnormal patterns, so they can serve as additional information.

B. Can current anomaly detectors provide accurate results?

This section demonstrates that *current detectors attached with localizers cannot deliver satisfying accuracy*.

As far as we know, existing root cause localization approaches for microservices follow such a pipeline: 1) conduct anomaly detection, and 2) if an anomaly is alarmed, then the localizer is triggered. That is, the anomaly detector and root cause localizer work separately. Unfortunately, incorrect anomaly detection results can exert a negative impact on the following root cause localization by introducing noisy labels. To investigate whether current anomaly detectors are satisfactory for downstream localizers, we first summarize three main kinds of anomaly detection approaches used in root cause localization papers. Note that since this paper targets root cause localization, the listed approaches are root cause localization-oriented anomaly detectors rather than sophisticated approaches for general anomaly detection.

- N-sigma used in [20], [21] computes the mean (μ) and the standard deviation (σ) of historical fault-free data. If the maximum latency of the current observation window is larger than $\mu + n \cdot \sigma$, an alarm will be triggered, where n is an empirical parameter.
- Feature engineering + machine learning (FE+ML) [9], [24] feeds manually derived features from traces into a machine learning-based model such as OC-SVM [9] to detect anomalies in a one-class-classification manner.
- SPOT [25] is an advanced algorithm for time series anomaly detection based on the Extreme Value Theory. Recent root cause analysis studies [6], [7] have applied it for detecting anomalies.

TABLE I
COMPARISON OF COMMON ANOMALY DETECTORS

	N-sigma	**FE+ML**	**SPOT**
FOR	0.632	0.830	0.638
FDR	0.418	0.095	0
#Infer/ms	0.207	1.361	549.169

We conduct effectiveness measurement experiments based on our data on the three anomaly detectors following [6], [9], [21], respectively. We focus on the false omission rate (*FOR*= $\frac{FN}{FN+TN}$) and the false discovery rate (*FDR*= $\frac{FP}{FP+TN}$), where TN is the number of successfully predicted normal samples; FN is the number of undetected anomalies; FP is the number of normal samples incorrectly triggering alarms. Besides, #Infer/ms denotes the average inference time with the unit of microseconds.

Table I lists the experimental results, demonstrating a large improvement space for these anomaly detectors. The high *FOR* and *FDR* indicate that the inputs of the root cause localizer contain lots of noisy labels, thereby substantially influencing localization performance. We attribute this partly to the closed-world assumption relied on by these methods, that is, regarding normal but unseen data patterns as abnormal, thereby incorrectly forcing the downstream localizer to search for the "inexistent" root cause based on normal data. Also, latency is insufficient to reveal all anomalies, as stated before, especially those that do not severely delay inter-service communications, represented by the high *FOR*.

In addition, complex methods (FE+ML and SPOT) have better effectiveness than N-sigma yet burden the troubleshooting process by introducing extra computation. Since root cause localization requires anomaly detection first, the detector must be lightweight to mitigate the efficiency reduction. Even worse, these machine learning-based approaches require extra hyperparameter tuning, making the entire troubleshooting approach less practical.

Root cause localization requires anomalous data detected by anomaly detectors, but current localization-oriented detectors either deliver unsatisfactory accuracy and introduce noisy data or reduce efficiency, making the following localization troublesome.

In summary, these examples motivate us to design an end-to-end framework that integrates effective anomaly detection and root cause localization in microservices based on multi-source information, i.e., logs, KPIs, and traces. Logs, KPIs, and latency in traces provide local information on intra-service behaviors, while invocation chains recorded in traces depict the interactions between microservices, thereby providing a global view of the system status. This results in Eadro, the first work to enable jointly detecting anomalies and locating the root cause, all the while attacking the above-mentioned limitations by learning the microservice status concerning both intra- and inter-service properties from various types of data.

IV. METHODOLOGY

The core idea of Eadro is to learn the intra-service behaviors based on multi-modal data and capture dependencies between microservices to infer a comprehensive picture of the system status. Figure 5 displays the overview of Eadro, containing three phases: *modal-wise learning*, *dependency-aware status learning*, and *joint detection and localization*.

A. Modal-wise Learning

This phase aims to model the different sources of monitoring data individually. We apply modality-specific models to learn an informative representation for each modality.

1) Log Event Learning: We observe that both log semantics and event occurrences can reflect anomalies (§ III-A), yet we herein focus on event occurrences because of two reasons: 1) the logging behavior of microservices highly relies on the

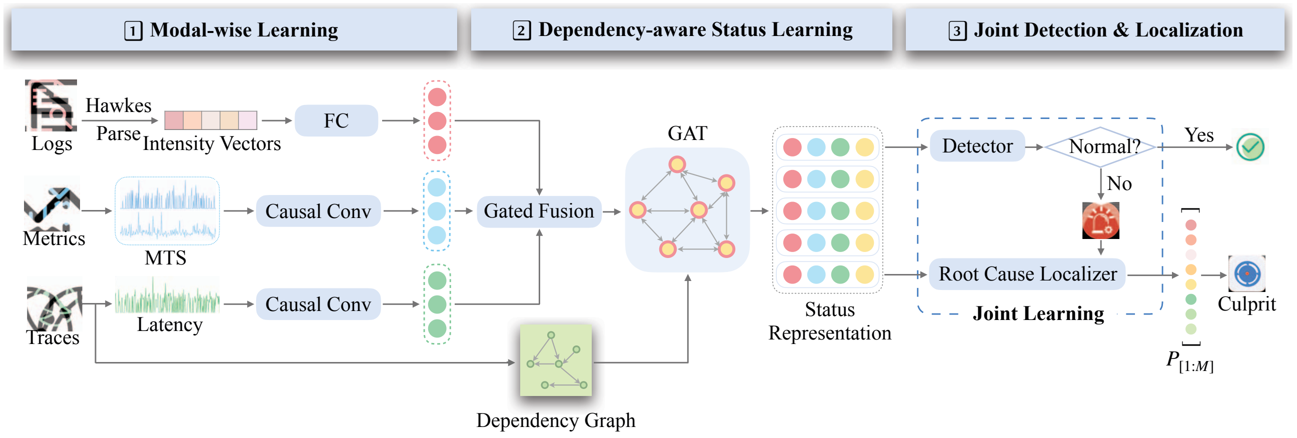


Fig. 5. Overview of Eadro

developers' expertise, so the quality of log semantics cannot be guaranteed [16]; 2) the complexity of microservices necessitates lightweight techniques. As semantic extraction requires computation-intensive natural language processing technologies, log semantic-based methods may pose challenges in practical usage.

Therefore, we focus on modeling the occurrences of log events instead of log semantics. An insight facilitates the model. We observe that the past event increases the likelihood of the event's occurrence in the near future, which fits the assumption of the self-exciting process [26]. Hence, we initially propose to adopt the Hawkes process [13], a kind of self-exciting point process, to model the event occurrences, which is defined by the conditional intensity function:

$$\lambda_l^*(t) = \mu_l(t) + \sum_{\tau<t} \phi_l(t-\tau) \tag{1}$$

where $l = 1, ..., L$ and L is the number of event types; for the l-th event, μ_l is an estimated parameter and $\phi_l(\cdot)$ is a user-defined triggering kernel function. We use an exponential parametrisation of the kernels herein following [27]: $\phi_l(\cdot) = \alpha_l \beta \exp(-\beta t)|_{t>0}$, where $\alpha_1 \cdots \alpha_L$ are estimated parameters and β is a hyper-parameter.

In brief, log learning is done in a three-step fashion:

A. Parsing: Eadro starts with parsing logs into events via Drain [22] by removing variables in log messages.
B. Estimating: we then record the timestamps of event occurrences (relative to the starting timestamp of the observation window) to estimate the parameters of the Hawkes model with an exponential decay kernel. The estimation is implemented via an open-source toolkit Tick [28]. In this way, events $\mathbf{X}^{\mathcal{L}}$ at each microservice inside a window are transformed into an intensity vector $\mathbf{\Lambda} = [\lambda_1^*, \cdots, \lambda_L^*] \in \mathbb{R}^L$.
C. Embedding: the intensity vector $\mathbf{\Lambda}$ is embedded into a dense vector $\boldsymbol{H}^{\mathcal{L}} \in \mathbb{R}^{E^{\mathcal{L}}}$ in the latent space via a fully connected layer with the hidden size of $E^{\mathcal{L}}$.

2) KPI Learning: We first organize the KPIs $\mathbf{X}^{\mathcal{K}}$ with k indicators of each microservice into a k-variate time series with the length of T. Then we use a 1D dilated causal convolution (DCC) [14] layer that is lightweight and parallelizable to learn the temporal dependencies and cross-series relations of KPIs. Previous studies have demonstrated DCC's computational efficiency and accuracy in feature extraction of time series [29]. Afterward, we apply a self-attention [30] operation to compute more reasonable representations, and the attention weights are as computed in Equation 2.

$$Attn(X) = \text{softmax}\left(\frac{W_q X \cdot (W_k X)^{\mathrm{T}}}{\sqrt{d}}(W_v X)\right) \tag{2}$$

where W_q, W_k, and W_v are learnable parameters, and d is an empirical scaling factor. This phase outputs $\boldsymbol{H}^{\mathcal{K}} \in \mathbb{R}^{E^{\mathcal{K}}}$ representing KPIs, where $E^{\mathcal{K}}$ is the number of convolution filters.

3) Trace Learning: Inspired by previous works [3], [6], [31], we extract latency from trace files and transform it into a time series by calculating the average latency at a time slot for each callee. We obtain a T-length univariate latency time series at each microservice (i.e., callee). Similarly, the latency time series is fed into a 1D DCC layer followed by a self-attention operation to learn the latent representation $\boldsymbol{H}^{\mathcal{T}} \in \mathbb{R}^{E^{\mathcal{T}}}$, where $E^{\mathcal{T}}$ is the pre-defined number of filters. Note that we simply pad time slots without corresponding invocations with zeros.

B. Dependency-aware Status Learning

In this phase, we aim to learn microservices' overall status and draw a comprehensive picture of the system. This module consists of three steps: dependency graph construction, multi-modal fusion, and dependency graph modeling. We first extract a directional graph depicting the relationships among microservices from historical traces. Afterward, we fuse the multi-modal representations obtained from the previous phases into latent node embeddings to represent the service-level status. Messages within the constructed graph will be propagated through a graph neural network so as to learn the neighboring dependencies represented in the edge weights. Eventually, we can obtain a dependency-aware representation representing the overall status of the microservice system.

1) Dependency Graph Construction: By regarding microservices as nodes and invocations as directional edges, we can extract a dependency graph $\mathcal{G} = \{\mathbb{V}, \mathbb{E}\}$ from historical traces to depict the dependencies between microservices. Specifically, $\mathbb{V}$ is the node set and $|\mathbb{V}| = M$, where M is the number of microservices; $\mathbb{E}$ is the set of edges, and $\vec{e}_{a,b} = (v_a, v_b) \in \mathbb{E}$ denotes an edge directed from v_a to v_b, that is, v_b has invoked v_a at least once in the history.

2) Multi-modal Fusion: In general, there are three fusion strategies [32]: early fusion carried out at the input level, intermediate fusion for fusing cross-modal representations, and late fusion at the decision level (e.g., voting). Research in cross-modal learning [33], [34] and neuroscience [35], [36] suggests that intermediate fusion usually facilitates modeling, so we transform single-modal representations to a compact multi-modal representation via intermediate fusion.

The fusion contains two steps:

A. We concatenate ($[\cdot||\cdot]$) all representations of each microservice obtained from the previous phase to retain exhaustive information. The resulting vector $[\boldsymbol{H}^{\mathcal{L}}||\boldsymbol{H}^{\mathcal{K}}||\boldsymbol{H}^{\mathcal{T}}]$ is subsequently fed into a fully connected layer to be projected into a lower-dimensional space, denoted by $\boldsymbol{H}'^{\mathcal{S}} \in \mathbb{R}^{2E}$, where $2E < E^{\mathcal{L}} + E^{\mathcal{K}} + E^{\mathcal{T}}$ is an even number.

B. $\boldsymbol{H}'^{\mathcal{S}}$ passes through a Gated Linear Unit (GLU) [37] to fuse representations in a non-linear manner and filter potential redundancy. GLU controls the bandwidth of information flow and diminishes the vanishing gradient problem. It also possesses extraordinary resilience to catastrophic forgetting. As we have massive data and complex stacked neural layers, GLU fits our scenario well. The computation follows $\boldsymbol{H}^{\mathcal{S}} = GLU(\boldsymbol{H}'^{\mathcal{S}}) = \boldsymbol{H}'^{\mathcal{S}}_{(1)} \otimes \sigma(\boldsymbol{H}'^{\mathcal{S}}_{(2)})$, where $\boldsymbol{H}'^{\mathcal{S}}_{(1)}$ is the first half of $\boldsymbol{H}'^{\mathcal{S}}$ and $\boldsymbol{H}'^{\mathcal{S}}_{(2)}$ is the second half; $\otimes$ denotes element-wise product, and σ is a sigmoid function.

Finally, we obtain $\boldsymbol{H}^{\mathcal{S}} \in \mathbb{R}^{E}$, a service-level representation of each microservice.

3) Dependency Graph Learning: As interactions between microservices can be naturally described by dependency graphs, we apply graph neural networks to perform triage inference. Particularly, we employ Graph Attention Network (GAT) [38] to learn the dependency-aware status of the microservice system. GAT enables learning node and edge representations and dynamically assigns weights to neighbors without requiring computation-consuming spectral decompositions. Hence, the model can pay attention to microservices with abnormal behaviors or at the communication hub.

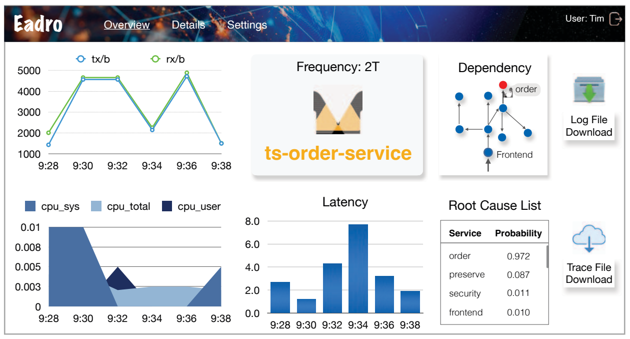
The local representation $\boldsymbol{H}^{\mathcal{S}}$ serves as the node feature, and GAT learns the whole graph's representation, where dynamic weights of edges are computed as Equation 3.

$$\omega_{a,b} = \frac{\exp(\mathsf{LeakyReLU}(v^{\mathrm{T}}[\boldsymbol{W}\boldsymbol{H}^{\mathcal{S}}_a||\boldsymbol{W}\boldsymbol{H}^{\mathcal{S}}_b]))}{\sum_{k\in\mathbb{N}_a}\exp(\mathsf{LeakyReLU}(v^{\mathrm{T}}[\boldsymbol{W}\boldsymbol{H}^{\mathcal{S}}_a||\boldsymbol{W}\boldsymbol{H}^{\mathcal{S}}_k]))} \quad (3)$$

where $\omega_{a,b}$ is the computed weight of edge $\vec{e}_{a,b}$; $\mathbb{N}_a$ is the set of neighbor nodes of node a; $\boldsymbol{H}^{\mathcal{S}}_a$ is the inputted node feature of a; $\boldsymbol{W} \in \mathbb{R}^{E^{\mathcal{G}} \times E}$ and $v \in \mathbb{R}^{2E^{\mathcal{G}}}$ are learnable parameters. $E^{\mathcal{G}}$ is the dimension of the outputted representation, which is calculated by $\hat{\boldsymbol{H}}^{\mathcal{S}}_a = \psi(\sum_{b\in\mathbb{N}_a}\omega_{a,b}\boldsymbol{W}\boldsymbol{H}^{\mathcal{S}}_b)$, where $\psi(\cdot)$ is a customized activation function, usually ReLU. Eventually, we perform global attention pooling [39] on the multi-modal representations of all nodes. The final output is $\boldsymbol{H}^{\mathcal{F}} \in \mathbb{R}^{E^{\mathcal{F}}}$, a dependency-aware representation of the overall system status.

C. Joint Detection and Localization

Lastly, Eadro predicts whether the current observation window is abnormal and if so, it identifies which microservice the root cause is. As demonstrated in § III-B, existing troubleshooting methods regard anomaly detection and root cause localization as independent and ignore their shared knowledge. Besides, current anomaly detectors deliver unsatisfactory results and affect the next-stage localization by incorporating noisy labels. Therefore, we fully leverage the shared knowledge and integrate two closely related tasks into an end-to-end model.



Root Cause List

Service	Probability
order	0.972
preserve	0.087
security	0.011
frontend	0.010

Fig. 6. A demo for reviewing the suspicious status.

In particular, based on the previously obtained representation $\boldsymbol{H}^{\mathcal{F}}$, a detector first conducts binary classification to decide the existence of anomalies. If no anomaly exists, Eadro directly outputs the result; if not, a localizer ranks the microservices according to their probabilities of being the culprit. The detector and the localizer are both composed of stacked fully-connected layers and jointly trained by sharing an objective. The detector aims to minimize the binary cross-entropy loss:

$$\mathfrak{L}_1 = \sum_{i=1}^{N}[-(y_i\log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i))] \quad (4)$$

where N is the number of historical samples; $y_i \in \{0,1\}$ is the ground truth indicating the presence of anomalies (1 denotes presence while 0 denotes absence), and $\hat{y}_i \in [0,1]$ is the predicted indicator. Subsequently, all samples predicted as normal (0) are masked, and samples predicted as abnormal (1) pass through the localizer. The localizer attempts to narrow the distance between the predicted and ground-truth probabilities, whose objective is expressed by:

$$\mathfrak{L}_2 = \sum_{i=1}^{N}\sum_{s=1}^{M} c_{i,s}\log(p_{i,s}) \quad (5)$$

where M is the number of involved microservices. In the i-th sample, $c_{i,s} \in \{0,1\}$ is 1 if the culprit microservice is s and 0 otherwise; $p_{i,s}$ is the predicted probability of microservice s being the culprit. The objective of Eadro is the weighted sum of the two sub-objectives $\mathfrak{L} = \beta \cdot \mathfrak{L}_1 + (1-\beta) \cdot \mathfrak{L}_2$, where β is a hyper-parameter balancing the two tasks. Eventually, Eadro outputs a ranked list of microservices to be checked according to their predicted probabilities of being the root cause.

To sum up, Eadro can provide explicit clues about the microservice status. Hence, troubleshooting is much more convenient for operation engineers with the ranked list of microservices. Figure 6 presents a visualized demo.

V. Evaluation

This section answers the following research questions:

- **RQ1**: How effective is Eadro in anomaly detection?
- **RQ2**: How effective is Eadro in root cause localization?
- **RQ3**: How much does each data source contribute?

A. Data Collection

Since existing data collections of microservice systems [40], [41] contain traces only, we deploy two benchmark microservice systems and generate requests to collect multi-source data, including logs, KPIs, and traces. Afterward, we inject typical faults to simulate real-world anomalies. To our best knowledge, it is the first triple-source data collection with injected faults in the context of microservices.

1) Benchmark microservice systems: We first deploy two open-source microservice benchmarks: TrainTicket [17] (TT) and SocialNetwork [42] (SN). TT provides a railway ticketing service where users can check, book, and pay for train tickets. It is widely used in previous works [3], [15] with 41 microservices actively interacting with each other, and 27 of them are business-related. SN implements a broadcast-style social networking site. Users can create, read, favorite, and repost posts. In this system, 21 microservices communicate with each other via Thrift RPCs [43]. SN has 21 microservices, 14 of which are related to business logic.

We construct a distributed testbed to deploy the two systems running in Docker containers and develop two request simulators to simulate valid user requests. A series of open-source monitoring tools are deployed for data collection. Microservice instances send causally-related traces to a collector Jaeger [44]. We employ cAdvisor [45] and Prometheus [46] to monitor the KPIs per second of each microservice. The KPIs are stored in an instance of InfluxDB [47], including "CPU system usage", "CPU total usage", "CPU user usage", "memory usage", the amount of "working set memory", "rx bytes" (received bytes), and "tx bytes" (transmitted bytes). We also utilize Elasticsearch [48], Fluentd [49], and Kibana [50] to collect, aggregate, and store logs, respectively.

2) Fault Injection: Eadro can troubleshoot anomalies that manifest themselves in performance degradations (logs and KPIs) or latency deviations (traces). Referring to previous studies [6], [21], [24], we inject three typical types of faults via Chaosblade [51]. Specifically, we simulate CPU exhaustion by putting a hog to consume CPU resource heavily. To simulate a network jam, we delay the network packets of a microservice instance. We also randomly drop network packets to simulate stochastic packet loss that frequently occurs when excessive data packets flood a network.

We generate 0.2∼0.5 and 2∼3 requests per second for TT and SN at a uniform rate, respectively. Before fault injection, we collect normal data under a fault-free setting for 7 hours for TT and 1.2 hours for SN. Then, we set each fault duration to 10 mins (with a 2-min interval between two injections) for TT, while the fault duration is 2 mins and SN's interval is half a minute. Each fault is injected into one microservice once. In total, we conduct 162 and 72 injection operations in TT and SN, respectively. Such different setups are attributed to the different processing capacities of the two systems, i.e., TT usually takes more time to process a request than SN.

In this way, we collect two datasets ($\mathcal{TT}$ and $\mathcal{SN}$) with 48,296 and 126,384 traces, respectively. Data produced in different periods are divided into training (60%) data and testing (40%) data, respectively. The data divisions share similar distributions in abnormal/normal ratios and root causes.

B. Baselines

We compare Eadro with previous approaches and derived methods integrating multi-source data. As our task is relatively novel by incorporating more information than existing single-source data-based studies, simply comparing our model with previous approaches seems a bit unfair.

1) Advanced baselines: In terms of anomaly detection, we consider two state-of-the-art baselines. TraceAnomaly [3] uses a variational auto-encoder (VAE) to discover abnormal invocations. MultimodalTrace [4] extracts operation sequences and latency time series from traces and uses a multi-modal Long Short-term Memory (LSTM) network to model the temporal features. For root cause localization, we compare Eadro with five trace-based baselines: TBAC [10], NetMedic [52], MonitorRank [8], CloudRanger [11], and DyCause [6]. As far as we know, no root cause localizers for microservices rely on multi-modal data.

These methods use statistical models or heuristic methods to locate the root cause. For example, TBAC, MonitorRank, and DyCause applied the Pearson correlation coefficient, and MonitorRank and DyCause also leveraged Random Walk. We implement these baselines referring to the codes provided by the original papers [3], [6], [21]. For the papers without open-source codes, we carefully follow the papers and refer to the baseline implementation released by [6].

2) Derived multi-source baselines: We also derive four multi-source data-based methods for further comparison. Inspired by [4], we transform all data sources into time series and use learning-based algorithms for status inference. Specifically, logs are represented by event occurrence sequences; traces are denoted by latency time series; KPIs are natural time series. Since previous studies are mainly machine learning-based, we train practical machine learning methods, i.e., Random Forest (RF) and Support Vector Machine (SVM), on the multi-source time series. We derive MS-RF-AD and MS-SVM-AD for anomaly detection as well as MS-RF-RCL and MS-SVM-RCL for root cause localization. We also derive two methods (MS-LSTM and MS-DCC) that employ deep learning techniques, i.e., LSTM and 1D DCC, to extract representations from multi-modal time series. The learned representations are fed into the module of joint detection and localization, which is described in IV-C.

C. Implementation

The experiments are conducted on a Linux server with an NVIDIA GeForce GTX 1080 GPU via Python 3.7. As for the hyper-parameters, the hidden size of all fully-connected layers is 64, and every DCC layer shares the same filter number of 64 with a kernel size of three. The GAT's hidden size and the fusion dimension (i.e., $2E$) are 128. We use a 4-head mechanism of GAT's attention layer, and the layer number of all modalities' models is only one for speeding up. Moreover, Batch Normalization [53] is added after DCCs to mitigate overfitting. We train Eadro using the Adam [54] optimizer with an initial learning rate of 0.001, a batch size of 256, and an epoch number of 50. All the collected data and our code are released for replication.

D. Evaluation Measurements

The anomaly detection challenge is modeled in a binary classification manner, so we apply the widely-used binary classification measurements to gauge the performance of models: Recall (*Rec*)= $\frac{TP}{TP+FN}$, Precision (*Pre*)= $\frac{TP}{TP+FP}$, F1-score (*F1*)= $\frac{2\cdot Pre\cdot Rec}{Pre+Rec}$, where TP is the number of discovered abnormal samples; FN and FP are defined in § III-B.

For root cause localization, we introduce the Hit Rate of top-k (*HR@k*) and Normalized Discounted Cumulative Gain of top-k (*NDCG@k*) for localizer evaluation. Herein, we set $k = 1, 3, 5$. *HR@k*= $\frac{1}{N}\sum_{i=1}^{N}(s_i^t \in S_{i,[1:k]}^p)$ calculates the overall probability of the culprit microservice within the top-k predicted candidates $S_{i,[1:k]}^p$, where s_i^t is the ground-truth root cause for the i-th observation window, and N is the number of samples to be tested. *NDCG@k*= $\frac{1}{N}\sum_{i=1}^{N}(\sum_{j=1}^{M}\frac{p_j}{\log_2(j+1)})$ measures the ranking quality, where p_j is the predicted probability of the j-th microservice, and M is the number of microservices. *NDCG@1* is left out because it is the same with *HR@1* in our scenario. The two evaluation metrics measure how easily engineers find the culprit microservice. *HR@k* directly measures how likely the root cause will be found within k checks. *NDCG@k* measures to what extent the root cause appears higher up in the ranked candidate list. Thus, the higher the above measurements, the better.

E. **RQ1**: Effectiveness in Anomaly Detection

Ground truths are based on the known injection operations, i.e., if a fault is injected, then the current observation window is abnormal; otherwise, it is normal. Table II displays a comparison of anomaly detection, from which we draw three observations:

(1) Eadro outperforms all competitors significantly and achieves very high scores in *F1* (0.988), *Rec* (0.996), and *Pre* (0.981), illustrating that Eadro generates very few missing anomalies or false alarms. Eadro's excellence can be attributed to 1) Eadro applies modality-specific designs to model various sources of data as well as a multi-modal fusion to wrangle these modalities so that it can learn a distinguishable representation of the status; 2) Eadro learns dependencies between microservices to enable extraction of anomaly propagation to facilitate tracing back to the root cause.

TABLE II
PERFORMANCE COMPARISON FOR ANOMALY DETECTION

Approaches	$\mathcal{TT}$			$\mathcal{SN}$		
	F1	*Rec*	*Pre*	*F1*	*Rec*	*Pre*
TraceAnomaly	0.486	0.414	0.589	0.539	0.468	0.636
MultimodalTrace	0.608	0.576	0.644	0.676	0.632	0.726
MS-RF-AD	0.817	0.705	0.971	0.773	0.866	0.700
MS-SVM-AD	0.787	0.678	0.938	0.789	0.770	0.808
MS-LSTM	0.967	0.997	0.940	0.948	0.959	0.937
MS-DCC	0.965	0.993	0.938	0.948	0.962	0.934
Eadro	**0.989**	**0.995**	**0.984**	**0.986**	**0.996**	**0.977**

(2) Generally, multi-source data-based approaches, including Eadro, perform much better than trace-relied baselines because they incorporate extra essential information (i.e., logs and KPIs) besides traces. The results align with our observations in § III-A that logs and KPIs provide valuable clues about microservice anomalies, while traces cannot reveal all anomalies. Trace-based methods can only detect anomalies yielding an enormous impact on invocations, so they ignore anomalies reflected by other data sources.

(3) Moreover, Eadro, MS-LSTM, and MS-DDC perform better than MS-SVM and MS-RF. The superiority of the former ones lies in applying deep learning and joint learning. Deep learning has demonstrated a powerful capacity in extracting features from complicated time series [29], [55], [56]. Joint learning allows capturing correlated knowledge across detection and localization to exploit commonalities across the two tasks. These two mechanisms are beneficial to troubleshooting by enhancing representation learning.

In brief, Eadro is very effective in anomaly detection of microservice systems and improves *F1* by 53.82%∼92.68% compared to baselines and 3.13%∼25.32% compared to derived methods. The detector is of tremendous assistance for next-stage root cause localization by reducing noisy labels inside the localizer's inputs.

F. **RQ2**: Effectiveness in Root Cause Localization

To focus on comparing the effectiveness of root cause localization, we provide ground truths of anomaly existence for baselines herein. In contrast, Eadro, MS-LSTM, and MS-DCC use the predicted results of their detectors as they are end-to-end approaches integrating the two tasks. Table III presents the root cause localization comparison, underpinning three observations:

(1) Eadro performs the best, taking all measurements into consideration, achieving *HR@1* of 0.982, *HR@5* of 0.990, and *NDCG@5* of 0.989 on average. With the incorporation of valuable logs and KPIs ignored by previous approaches,

TABLE III
PERFORMANCE COMPARISON FOR ROOT CAUSE LOCALIZATION

Approaches	$\mathcal{TT}$					$\mathcal{SN}$				
	HR@1	*HR@3*	*HR@5*	*NDCG@3*	*NDCG@5*	*HR@1*	*HR@3*	*HR@5*	*NDCG@3*	*NDCG@5*
TBAC	0.037	0.111	0.185	0.079	0.109	0.001	0.085	0.181	0.048	0.087
NetMedic	0.094	0.257	0.425	0.195	0.209	0.069	0.187	0.373	0.146	0.218
MonitorRank	0.086	0.199	0.331	0.142	0.196	0.068	0.118	0.221	0.095	0.137
CloudRanger	0.101	0.306	0.509	0.218	0.301	0.122	0.382	0.629	0.269	0.370
DyCause	0.231	0.615	0.808	0.448	0.607	0.273	0.636	0.727	0.301	0.353
MS-RF-RCL	0.637	0.922	0.970	0.807	0.827	0.704	0.908	0.970	0.825	0.851
MS-SVM-RCL	0.541	0.908	0.944	0.814	0.820	0.614	0.838	0.955	0.741	0.790
MS-LSTM	0.756	0.930	0.969	0.859	0.877	0.757	0.884	0.907	0.834	0.844
MS-DCC	0.767	0.938	0.972	0.870	0.882	0.789	0.968	0.985	0.898	0.905
Eadro	**0.990**	**0.992**	**0.993**	**0.994**	**0.994**	**0.974**	**0.988**	**0.991**	**0.982**	**0.983**

Eadro can depict the system status more accurately. Trace-based approaches have difficulties in troubleshooting resource exhaustion-related anomalies or severe network-related anomalies that block inter-service communications resulting in few invocations. Besides, Eadro enables eavesdropping across detection and localization via joint learning, which encourages full use of the shared knowledge to enhance status learning. Eadro also leverages powerful techniques to capture meaningful patterns from multi-modal data, including designs of modality-specific models and advanced GAT to exploit graph-structure dependencies. Moreover, Eadro achieves a much higher score in *HR@1* than derived methods, while its superiority in *HR@5* and *NDCG@5* is not particularly prominent. The reason is that Eadro learns the dependency-aware status besides intra-service behaviors, allowing to catch the anomaly origin by tracing anomaly propagation. Other multi-modal approaches capture dependency-agnostic information, so they can pinpoint the scope of suspicious microservices effectively rather than directly deciding the culprit.

(2) Multi-modal approaches considerably outperform single-modal baselines, similar to the results in anomaly detection. The superiority of multi-source derived methods is more evident since localization is a more complicated task than detection, so the advantage of incorporating diverse data sources to learn the complementarity is fully demonstrated. This situation is more revealing in $\mathcal{TT}$ because TrainTicket responds more slowly, leading to sparse trace records, and trace-based models get into trouble when few invocations occur in the current observation window. In contrast, derived approaches can accurately locate the culprit microservice in such a system since they leverage various information sources to obtain more clues.

(3) Considering multi-modal approaches, Eadro, MS-LSTM, and MS-DCC deliver better performance (measured by *HR@1*) than MS-RF-RCL and MS-SVM-RCL. The superiority of the former approaches can be attributed to the strong fitting ability of deep learning and the advantages brought by the joint learning mechanism. However, MS-LSTM performs poorer in narrowing the suspicious scope, especially in $\mathcal{SN}$ (measured by *HR@5* and *NDCG@5*). This may be because that LSTMs' training process is a lot more complicated than DCCs or simple machine learning techniques. The scale of $\mathcal{SN}$ is relatively small, so MS-LSTM cannot be thoroughly trained and capture the most meaningful features.

To sum up, the results demonstrate the effectiveness of Eadro in root cause localization. Eadro increases *HR@1* by 290%$\sim$5068% than baselines and 26.93%$\sim$66.16% than derived methods. Our approach shows effectiveness both in anomaly detection and root cause localization, suggesting its potential to automate labor-intensive troubleshooting.

G. ***RQ3***: *Contributions of Different Data Sources*

We perform an ablation study to explore how different data sources contribute by conducting source-wise-agnostic experiments, so we derive the following variants:

- Eadro w/o $\mathcal{L}$: drops logs while inputs traces and KPIs by removing the log modeling module in § IV-A1.
- Eadro w/o $\mathcal{M}$: drops KPIs while inputs traces and logs by removing the KPI modeling module in § IV-A2.
- Eadro w/o $\mathcal{T}$: drops latency extracted from traces by removing the trace modeling module in § IV-A3.
- Eadro w/o $\mathcal{G}$: replaces GAT by an FC layer to learn dependency-agnostic representations.

TABLE IV
EXPERIMENTAL RESULTS OF THE ABLATION STUDY

Variants	$\mathcal{TT}$			$\mathcal{SN}$		
	HR@1	*HR@5*	*F1*	*HR@1*	*HR@5*	*F1*
Eadro	**0.990**	**0.993**	**0.989**	**0.974**	**0.991**	**0.986**
Eadro w/o $\mathcal{L}$	0.926	0.993	0.964	0.902	0.954	0.972
Eadro w/o $\mathcal{M}$	0.776	0.962	0.960	0.684	0.947	0.974
Eadro w/o $\mathcal{T}$	0.785	0.930	0.945	0.627	0.930	0.957
Eadro w/o $\mathcal{G}$	0.803	0.982	0.970	0.791	0.960	0.946

The ablation study results are shown in Table IV. Considering that root cause localization is a more difficult and our major target and that all variants achieve relatively good performance in anomaly detection, we focus on root cause localization. Clearly, each source of information contributes to the effectiveness of Eadro as it performs the best, while the degrees of their contributions are not exactly the same.

Specifically, logs contribute the least as Eadro w/o $\mathcal{L}$ is second-best. We attribute it to the lack of log semantics and the low logging frequency. As the two benchmark systems were recently proposed without multiple version iterations, only a few events are recorded. We believe that logs would play a greater value in the development of microservices.

In addition, we observe that the performance of Eadro w/o $\mathcal{M}$ and Eadro w/o $\mathcal{T}$ degrades dramatically, especially in *HR@1*, since traces and KPIs are essential information that contributes the most to the identification of the root cause microservice. This observation aligns with our motivating cases, where we show some anomaly cases that can be directly revealed by traces and KPIs.

Moreover, *HR@5* of Eadro w/o $\mathcal{G}$ degrades slightly, indicating that dependency-agnostic representations are useful to narrow the suspicious scope. However, *HR@1* of Eadro w/o $\mathcal{G}$ decreases 23.21% as Eadro uses readily applicable GAT to modal graph-structure inter-service dependencies, while FC layers model the dependencies linearly, unable to capture anomaly propagation well, leading to performance degradation in determining the culprit.

To further demonstrate the benefits brought by KPIs and logs, we visualize the latent representations of abnormal data samples learned by Eadro, Eadro w/o $\mathcal{L}$, and Eadro w/o $\mathcal{M}$ via t-SNE [57] of the test set of $\mathcal{SN}$, shown in Figures 7.

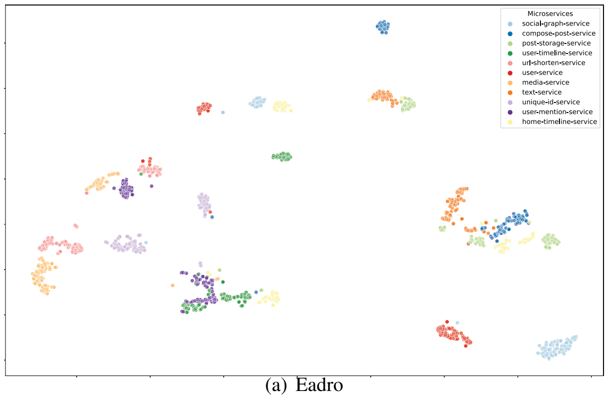
We can see that the representations learned by Eadro are the most discriminative, and those learned by Eadro w/o $\mathcal{L}$ are second-best, while those learned by Eadro w/o $\mathcal{M}$ are the worst. Specifically, Eadro distributes representations corresponding to different root causes into different clusters distant from each other in the hyperspace. In contrast, Eadro w/o $\mathcal{M}$ learns representations close in space, making it difficult to distinguish them for triage. That is why Eadro w/o $\mathcal{M}$ delivers poorer performance in localization than Eadro. The visualization intuitively helps us grasp the usefulness of KPIs in helping pinpoint the root cause. The discriminativeness of the representations learned by Eadro w/o $\mathcal{L}$ is in-between, where some clusters are pure while others seem to be a mixture of representations corresponding to different root causes, in line with the experiment results. We can attribute part of the success of Eadro to incorporating KPIs and logs, which encourages more discriminative representations of the microservice status with extra clues.

In conclusion, the involved data sources can all contribute to the effectiveness of Eadro to some degree, and traces contribute the most to the overall effectiveness. This emphasizes the insights about appropriately modeling multi-source data to troubleshoot microservices effectively.

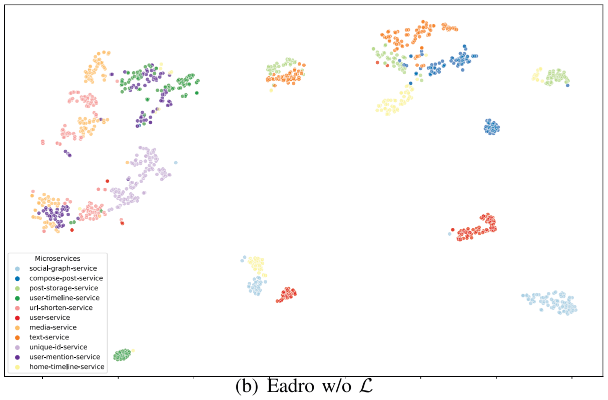
VI. Disucussion

A. Limitations

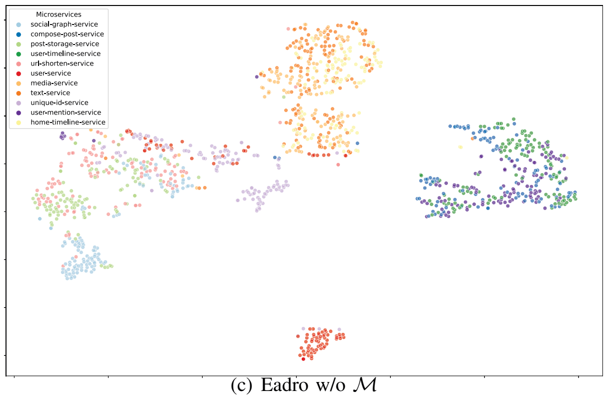
We identify three limitations of Eadro: 1) the incapacity to deal with bugs related to program logic; 2) the prerequisites for multi-source data collection; 3) the requirement of annotated data for training.



(a) Eadro



(b) Eadro w/o $\mathcal{L}$



(c) Eadro w/o $\mathcal{M}$

Fig. 7. Distributions of representations learned by Eadro and its variants.

As Eadro is an entirely data-driven approach targeting the scope of reliability management, it is only applicable to troubleshooting anomalies manifested in the involved data, so logical bugs out of our scope and silent issues that do not incur abnormal patterns in observed data can not be detected or located.

Moreover, Eadro is basically well-suited for all microservices where anomalies can be reflected in the involved three types of data we employ. However, some practical systems may lack the ability to collect the three types of data. Though the low-coupled nature of the modal-wise learning module allows the absence of some source of data, it is better to provide all data types to fully leverage Eadro. Since we apply standard open-source monitoring toolkits and these off-the-shelf toolkits can be directly instrumented, enabling microservices with the data collection ability is not difficult.

In addition, the supervised nature of Eadro requires a large amount of labeled training data, which may be time-consuming in the real world. Nevertheless, our approach outperforms compared with unsupervised approaches by a large margin, indicating that in practice, unsupervised methods may be difficult to use because the accuracy rate is not up to the required level, especially considering that realistic microservices systems are much larger and more complex. A common solution in companies is to use an unsupervised model to generate coarse-grained pseudo-labels. Afterward, experienced engineers manually review the labels with lower confidence. The hybrid-generated labels are used for training the supervised model, and eventually, the supervised approach performs the troubleshooting work. Hence, Eadro will still play an important role in practice and fulfill its potential.

B. Threat to Validity

1) Internal Threat: The main internal threat lies in the correctness of baseline implementation. We reproduce the baselines based on our understanding of their papers since most baselines, except DyCause and TraceAnomaly, have not released codes, but the understanding may not be accurate. To mitigate the threat, we carefully follow the original papers and refer to the baseline implementation released by [6].

2) External Threat: The external threats concern the generalizability of our experimental results. We evaluate our approach on two simulated datasets since there is no publicly available dataset containing multi-modal data. It is yet unknown whether the performance of Eadro can be generalized across other datasets. We alleviate this threat from two aspects. First, the benchmark microservice systems are widely used in existing comparable studies, and the injected faults are also typical and broadly applied in previous studies [6], [21], [24], thereby supporting the representativeness of the datasets. Second, our approach is request- and fault-agnostic, so an anomaly incurred by a fault beyond our injections can also be discovered if it causes abnormalities in the observations.

VII. Related Work

Previous anomaly detection approaches are usually based on system logs [58]–[62] or KPIs [63]–[67], or both [68], targeting traditional distributed systems without complex invocation relationships. Recently, some studies [3], [4], [31] have been presented to automate anomaly detection in microservice systems. [3] proposed to employ a variational autoencoder with a Bayes model to detect anomalies reflected by latency. [4] extracted operation sequence and invocation latency from traces and fed them into a multi-modal LSTM to identify anomalies. These anomaly detection methods rely on single-source data (i.e., traces) and ignore other informative data such as logs and KPIs.

Tremendous efforts [7], [8], [10], [11], [19], [24], [52] have been devoted to root cause localization in microservice or service-oriented systems, most of which rely on traces only and leverage traditional or naive machine learning techniques. For example, [15] conducted manual feature engineering in trace logs to predict latent errors and identify the faulty microservice via a decision tree. [9] proposed a high-efficient approach that dynamically constructs a service call graph and ranks candidate root causes based on correlation analysis. A recent study [6] designed a crowd-sourcing solution to resolve user-space diagnosis for microservice kernel failures. These methods work well when the latent features of microservices are readily comprehensible but may lack scalability to larger-scale microservice systems with more complex features. Deep learning-based approaches explore meaningful features from historical data to avoid manual feature engineering. Though deep learning has not been applied to root cause localization as far as we know, some approaches incorporated it for performance debugging. For example, to handle traces, [69] used convolution networks and LSTM, and [70] leveraged causal Bayesian networks.

However, they rely on traces and ignore other data sources, such as logs and KPIs, that can also reflect the microservice status. Also, they either focus on anomaly detection or root cause localization leading to the disconnection in the two closely related tasks. The inaccurate results of naive anomaly detectors affect the effectiveness of downstream localization. Moreover, many methods combine manual feature engineering with traditional algorithms, making it insufficiently practical in large-scale systems.

VIII. Conclusion

This paper first identifies two limitations of current troubleshooting approaches for microservices and aims to address them. The motivation is based on two observations: 1) the usefulness of logs and KPIs and the insufficiency of traces; 2) the unsatisfactory results delivered by current anomaly detectors. To this end, we propose an end-to-end troubleshooting approach for microservices, Eadro, the first work to integrate anomaly detection and root cause localization based on multi-source monitoring data. Eadro consists of powerful modality-specific models to learn intra-service behaviors from various data sources and a graph attention network to learn inter-service dependencies. Extensive experiments on two datasets demonstrate the effectiveness of Eadro in both detection and localization. It achieves *F1* of 0.988 and *HR@1* of 0.982 on average, vastly outperforming all competitors, including derived multi-modal methods. The ablation study further validates the contributions of the involved data sources. Lastly, we release our code and data to facilitate future research.

Ackownledgement

The work described in this paper was supported by the National Natural Science Foundation of China (No. 62202511), and the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14206921 of the General Research Fund).

References

[1] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu, "Characterizing microservice dependency and performance: Alibaba trace analysis," in *SoCC '21: ACM Symposium on Cloud Computing, Seattle, WA, USA, November 1 - 4, 2021*. ACM, 2021, pp. 412–426.

[2] 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 Trans. Software Eng.*, vol. 47, no. 2, pp. 243–260, 2021.

[3] P. Liu, H. Xu, Q. Ouyang, R. Jiao, Z. Chen, S. Zhang, J. Yang, L. Mo, J. Zeng, W. Xue, and D. Pei, "Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks," in *31st IEEE International Symposium on Software Reliability Engineering, ISSRE 2020, Coimbra, Portugal, October 12-15, 2020*. IEEE, 2020, pp. 48–58.

[4] S. Nedelkoski, J. Cardoso, and O. Kao, "Anomaly detection from system tracing data using multimodal deep learning," in *12th IEEE International Conference on Cloud Computing, CLOUD 2019, Milan, Italy, July 8-13, 2019*. IEEE, 2019, pp. 179–186.

[5] 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 *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2022, pp. 623–634.

[6] Y. Meng, S. Zhang, Y. Sun, R. Zhang, Z. Hu, Y. Zhang, C. Jia, Z. Wang, and D. Pei, "Localizing failure root causes in a microservice through causality inference," in *28th IEEE/ACM International Symposium on Quality of Service, IWQoS 2020, Hangzhou, China, June 15-17, 2020*. IEEE, 2020, pp. 1–10.

[7] ——, "Localizing failure root causes in a microservice through causality inference," in *28th IEEE/ACM International Symposium on Quality of Service, IWQoS 2020, Hangzhou, China, June 15-17, 2020*. IEEE, 2020, pp. 1–10.

[8] M. Kim, R. Sumbaly, and S. Shah, "Root cause detection in a service-oriented architecture," in *ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '13, Pittsburgh, PA, USA, June 17-21, 2013*. ACM, 2013, pp. 93–104.

[9] 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 *43rd IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2021, Madrid, Spain, May 25-28, 2021*. IEEE, 2021, pp. 338–347.

[10] N. Marwede, M. Rohr, A. van Hoorn, and W. Hasselbring, "Automatic failure diagnosis support in distributed large-scale software systems based on timing behavior anomaly correlation," in *13th European Conference on Software Maintenance and Reengineering, CSMR 2009, Architecture-Centric Maintenance of Large-SCale Software Systems, Kaiserslautern, Germany, 24-27 March 2009*. IEEE Computer Society, 2009, pp. 47–58.

[11] P. Wang, J. Xu, M. Ma, W. Lin, D. Pan, Y. Wang, and P. Chen, "Cloudranger: Root cause identification for cloud native systems," in *18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2018, Washington, DC, USA, May 1-4, 2018*. IEEE Computer Society, 2018, pp. 492–502.

[12] S. P. Uselton, L. Treinish, J. P. Ahrens, E. W. Bethel, and A. State, "Multi-source data analysis challenges," in *9th IEEE Visualization Conference, IEEE Vis 1998, Research Triangle Park, North Carolina, USA, October 18-23, 1998, Proceedings*. IEEE Computer Society and ACM, 1998, pp. 501–504.

[13] A. G. Hawkes, "Markov processes in APL," in *Conference Proceedings on APL 90: For the Future, APL 1990, Copenhagen, Denmark, August 13-17, 1990*. ACM, 1990, pp. 173–185.

[14] C. Lea, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks: A unified approach to action segmentation," in *Computer Vision - ECCV 2016 Workshops - Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III*, ser. Lecture Notes in Computer Science, vol. 9915, 2016, pp. 47–54.

[15] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*. ACM, 2019, pp. 683–694.

[16] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *ACM Comput. Surv.*, vol. 54, no. 6, pp. 130:1–130:37, 2021.

[17] X. Zhou, X. Peng, T. Xie, J. Sun, C. Xu, C. Ji, and W. Zhao, "Benchmarking microservice systems for software engineering research," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*. ACM, 2018, pp. 323–324.

[18] C. Pham, L. Wang, B. Tak, S. Baset, C. Tang, Z. T. Kalbarczyk, and R. K. Iyer, "Failure diagnosis for distributed systems using targeted fault injection," *IEEE Trans. Parallel Distributed Syst.*, vol. 28, no. 2, pp. 503–516, 2017.

[19] Z. Li, J. Chen, R. Jiao, N. Zhao, Z. Wang, S. Zhang, Y. Wu, L. Jiang, L. Yan, Z. Wang, Z. Chen, W. Zhang, X. Nie, K. Sui, and D. Pei, "Practical root cause localization for microservice systems via trace analysis," in *29th IEEE/ACM International Symposium on Quality of Service, IWQOS 2021, Tokyo, Japan, June 25-28, 2021*. IEEE, 2021, pp. 1–10.

[20] J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint performance issues with causal graphs in micro-service environments," in *Service-Oriented Computing - 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings*, ser. Lecture Notes in Computer Science, vol. 11236. Springer, 2018, pp. 3–20.

[21] G. Yu, P. Chen, H. Chen, Z. Guan, Z. Huang, L. Jing, T. Weng, X. Sun, and X. Li, "Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments," in *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*. ACM / IW3C2, 2021, pp. 3087–3098.

[22] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services, ICWS 2017, Honolulu, HI, USA, June 25-30, 2017*, I. Altintas and S. Chen, Eds. IEEE, 2017, pp. 33–40.

[23] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, "Experience report: Deep learning-based system log analysis for anomaly detection," *CoRR*, vol. abs/2107.05908, 2021. [Online]. Available: https://arxiv.org/abs/2107.05908

[24] M. Ma, J. Xu, Y. Wang, P. Chen, Z. Zhang, and P. Wang, "Automap: Diagnose your microservice-based web applications automatically," in *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*. ACM / IW3C2, 2020, pp. 246–258.

[25] A. Siffer, P. Fouque, A. Termier, and C. Largouët, "Anomaly detection in streams with extreme value theory," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*. ACM, 2017, pp. 1067–1075.

[26] A. Reinhart, "A review of self-exciting spatio-temporal point processes and their applications," *Statistical Science*, vol. 33, no. 3, pp. 299–318, 2018.

[27] K. Zhou, H. Zha, and L. Song, "Learning social infectivity in sparse low-rank networks using multi-dimensional hawkes processes," in *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2013, Scottsdale, AZ, USA, April 29 - May 1, 2013*, vol. 31. JMLR.org, 2013, pp. 641–649. [Online]. Available: http://proceedings.mlr.press/v31/zhou13a.html

[28] E. Bacry, M. Bompaire, S. Gaïffas, and S. Poulsen, "tick: a Python library for statistical learning, with a particular emphasis on time-dependent modeling," *ArXiv e-prints*, Jul. 2017.

[29] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *CoRR*, vol. abs/1803.01271, 2018. [Online]. Available: http://arxiv.org/abs/1803.01271

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017, pp. 5998–6008.

[31] T. Yang, J. Shen, Y. Su, X. Ling, Y. Yang, and M. R. Lyu, "AID: efficient prediction of aggregated intensity of dependency in large-scale cloud systems," in *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 2021, pp. 653–665.

[32] H. R. V. Joze, A. Shaban, M. L. Iuzzolino, and K. Koishida, "MMTM: multimodal transfer module for CNN fusion," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020,*

Seattle, WA, USA, June 13-19, 2020. Computer Vision Foundation / IEEE, 2020, pp. 13 286–13 296.

[33] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. IEEE Computer Society, 2014, pp. 1725–1732.

[34] W. Liu, W. Zheng, and B. Lu, "Multimodal emotion recognition using multimodal deep learning," *CoRR*, vol. abs/1602.08225, 2016. [Online]. Available: http://arxiv.org/abs/1602.08225

[35] M. M. Murray, A. Thelen, S. Ionta, and M. T. Wallace, "Contributions of intraindividual and interindividual differences to multisensory processes," *J. Cogn. Neurosci.*, vol. 31, no. 3, 2019.

[36] M. Marucci, G. Di Flumeri, G. Borghini, N. Sciaraffa, M. Scandola, E. F. Pavone, F. Babiloni, V. Betti, and P. Aricò, "The impact of multisensory integration and perceptual load in virtual reality settings on performance, workload and presence," *Scientific Reports*, vol. 11, no. 1, p. 4831, Mar. 2021.

[37] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 933–941.

[38] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" *CoRR*, vol. abs/2105.14491, 2021. [Online]. Available: https://arxiv.org/abs/2105.14491

[39] D. Beck, G. Haffari, and T. Cohn, "Graph-to-sequence learning using gated graph neural networks," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*. Association for Computational Linguistics, 2018, pp. 273–283.

[40] N. L. of Tsinghua University. (2020) 2020 international aiops challenge. [Online]. Available: https://github.com/NetManAIOps/AIOps-Challenge-2020-Data

[41] H. Qiu, S. S. Banerjee, S. Jha, Z. T. Kalbarczyk, and R. Iyer. (2020) Pre-processed tracing data for popular microservice benchmarks.

[42] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, and C. Delimitrou, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*. ACM, 2019, pp. 3–18.

[43] Apache. (2022) Apache thrift. [Online]. Available: https://thrift.apache.org/

[44] C. N. C. Foundation. (2022) Jaeger. [Online]. Available: https://www.jaegertracing.io/

[45] Google. (2022) Container advisor. [Online]. Available: https://github.com/google/cadvisor

[46] C. N. C. Foundation. (2022) Prometheus. [Online]. Available: https://prometheus.io/

[47] InfluxData. (2022) Influxdb. [Online]. Available: https://www.influxdata.com/

[48] Elastic. (2022) Elasticsearch. [Online]. Available: https://www.elastic.co/

[49] S. Furuhashi. (2022) Fluentd. [Online]. Available: https://www.fluentd.org/architecture

[50] Elastic. (2022) Kibana. [Online]. Available: https://www.elastic.co/cn/kibana/

[51] Alibaba. (2022) Chaosblade. [Online]. Available: https://github.com/chaosblade-io/chaosblade

[52] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl, "Detailed diagnosis in enterprise networks," in *Proceedings of the ACM SIGCOMM 2009 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Barcelona, Spain, August 16-21, 2009*. ACM, 2009, pp. 243–254.

[53] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, ser. JMLR Workshop and Conference Proceedings, vol. 37. JMLR.org, 2015, pp. 448–456.

[54] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[55] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. Muller, "Deep learning for time series classification: a review," *Data Min. Knowl. Discov.*, vol. 33, no. 4, pp. 917–963, 2019.

[56] T. Fu, "A review on time series data mining," *Eng. Appl. Artif. Intell.*, vol. 24, no. 1, pp. 164–181, 2011.

[57] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[58] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. ACM, 2017, pp. 1285–1298.

[59] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, and R. Zhou, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. ijcai.org, 2019, pp. 4739–4745.

[60] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*. ACM, 2019, pp. 807–817.

[61] 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 *31st IEEE International Symposium on Software Reliability Engineering, ISSRE 2020, Coimbra, Portugal, October 12-15, 2020*. IEEE, 2020, pp. 92–103.

[62] V. Le and H. Zhang, "Log-based anomaly detection without log parsing," *CoRR*, vol. abs/2108.01955, 2021. [Online]. Available: https://arxiv.org/abs/2108.01955

[63] Z. Chen, J. Liu, Y. Su, H. Zhang, X. Ling, Y. Yang, and M. R. Lyu, "Adaptive performance anomaly detection for online service systems via pattern sketching," *CoRR*, vol. abs/2201.02944, 2022. [Online]. Available: https://arxiv.org/abs/2201.02944

[64] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, and Q. Zhang, "Time-series anomaly detection service at microsoft," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*. ACM, 2019, pp. 3009–3017.

[65] Z. Li, Y. Zhao, J. Han, Y. Su, R. Jiao, X. Wen, and D. Pei, "Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding," in *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*. ACM, 2021, pp. 3220–3230.

[66] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "USAD: unsupervised anomaly detection on multivariate time series," in *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*. ACM, 2020, pp. 3395–3404.

[67] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*. ACM, 2019, pp. 2828–2837.

[68] C. Lee, T. Yang, Z. Chen, Y. Su, Y. Yang, and M. R. Lyu, "Heterogeneous anomaly detection for software systems via semi-supervised cross-modal attention," 2022.

[69] Y. Gan, Y. Zhang, K. Hu, D. Cheng, Y. He, M. Pancholi, and C. Delimitrou, "Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*. ACM, 2019, pp. 19–33.

[70] Y. Gan, M. Liang, S. Dev, D. Lo, and C. Delimitrou, "Sage: practical and scalable ml-driven performance debugging in microservices," in *ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021*. ACM, 2021, pp. 135–151.