Interpretable Failure Localization for Microservice Systems Based on Graph Autoencoder

YONGQIAN SUN, Nankai University, China
ZIHAN LIN, Nankai University, China
BINPENG SHI, Nankai University, China
SHENGLIN ZHANG*, Nankai University, China
SHIYU MA, Nankai University, China
PENGXIANG JIN, Alibaba (Beijing) Software Services Co., Ltd., China
ZHENYU ZHONG, Nankai University, China
LEMENG PAN, AI Application Research Center, Huawei Technologies Co., China
YICHENG GUO, AI Application Research Center, Huawei Technologies Co., China
DAN PEI, Tsinghua University, China

Accurate and efficient localization of root cause instances in large-scale microservice systems is of paramount importance. Unfortunately, prevailing methods face several limitations. Notably, some recent methods rely on supervised learning which necessitates a substantial amount of labeled data. However, labeling root cause instances is time-consuming and laborious, especially with multiple modalities of data including logs, traces, metrics, *etc.* Moreover, some approaches favor deep learning for localization but lack interpretability and continuous improvement mechanisms.

To address the above challenges, we propose *DeepHunt*, a novel root cause localization method based on multimodal data analysis. Firstly, *DeepHunt* introduces Root Cause Score (RCS) by integrating reconstruction errors and failure propagation patterns (upstream-downstream relationships), imparting interpretability to the localization of root causes. Then, it embraces Graph Autoencoder (GAE) to address the limitation imposed by scarce labeled data. It employs data augmentation to mitigate the adverse effects of insufficient historical training samples. We evaluate *DeepHunt* on two open-source datasets, and it outperforms existing methods when facing a zero-label cold start. *DeepHunt* can be further improved by continuously fine-tuning through a feedback mechanism.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; • **Computer systems organization** → **Reliability**; • **Software and its engineering** → *Software maintenance tools.*

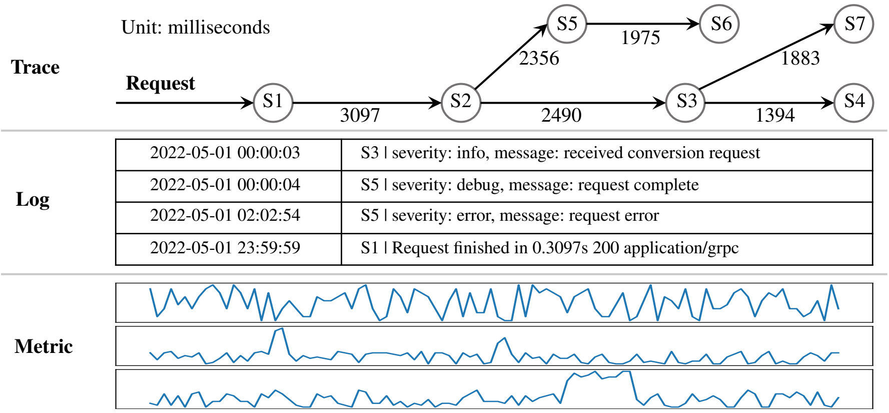
Additional Key Words and Phrases: Microservice, Failure localization, Self-supervised learning

1 INTRODUCTION

With the rising popularity of cloud-native applications, microservice architecture has emerged as an increasingly attractive choice due to its reliability and scalability [11]. However, the inherent complexity and dynamism of microservices make failures an unavoidable challenge. A single failure in a microservice instance can propagate to other interconnected instances, gradually amplifying its impact, potentially resulting in significant financial losses [56].

[*] Shenglin Zhang is the corresponding author.

Authors' addresses: Yongqian Sun, sunyongqian@nankai.edu.cn, Nankai University, Binhaixin Qu, Tianjin Shi, China; Zihan Lin, linzihan@mail.nankai.edu.cn, Nankai University, Binhaixin Qu, Tianjin Shi, China; Binpeng Shi, shibinpeng@mail.nankai.edu.cn, Nankai University, Binhaixin Qu, Tianjin Shi, China; Shenglin Zhang, zhangsl@nankai.edu.cn, Nankai University, Binhaixin Qu, Tianjin Shi, China; Shiyu Ma, mashiyu@mail.nankai.edu.cn, Nankai University, Binhaixin Qu, Tianjin Shi, China; Pengxiang Jin, jinpengxiang.jpx@alibaba-inc.com, Alibaba (Beijing) Software Services Co., Ltd., Chaoyang Qu, Beijing Shi, China; Zhenyu Zhong, zyzhong@mail.nankai.edu.cn, Nankai University, Binhaixin Qu, Tianjin Shi, China; Lemeng Pan, panlemeng@huawei.com, AI Application Research Center, Huawei Technologies Co., China; Yicheng Guo, guoyicheng3@huawei.com, AI Application Research Center, Huawei Technologies Co., China; Dan Pei, peidan@tsinghua.edu.cn, Tsinghua University, Haidian Qu, Beijing Shi, China.



2022-05-01 00:00:03	S3 | severity: info, message: received conversion request
2022-05-01 00:00:04	S5 | severity: debug, message: request complete
2022-05-01 02:02:54	S5 | severity: error, message: request error
2022-05-01 23:59:59	S1 | Request finished in 0.3097s 200 application/grpc

Fig. 1. The multimodal data of a microservice system. S1 - S7 are different microservice instances. The values in the trace represent the latency of an invocation.

An illustration case is the failure of the microservice instances on Amazon Web Services in December 2021, which reverberated throughout the entire network. It took more than four hours to pinpoint the root cause, leading to substantial economic repercussions [4]. Consequently, it is critical to localize microservice system failures promptly and effectively. As businesses expand and demand increases, the microservice system's scale and complexity also escalate. This evolution renders traditional root cause localization methods reliant on human labor increasingly inadequate to meet the requirements. Thus, the adoption of automated methods becomes imperative.

Extensive research has been dedicated to the automatic localization of failure root causes, aiming to quickly identify the system instance responsible for failures. The monitoring data used for this task encompasses three distinct modalities, namely traces [11, 24, 29, 52], logs [9, 28, 54, 57], and metrics [27, 33, 34, 40]. Traces record invocations between microservices. Logs contain runtime messages and warnings. Metrics monitor resource usage and performance indicators. Fig. 1 shows examples of these three modalities. We use *unimodal* to refer to a single data modality, while *multimodal* means combining two or more data modalities. Earlier methods rely primarily on unimodal data for failure localization. However, recent studies have revealed that more valuable insights can be obtained by combining all three modalities, as they provide a complete view of the overall system status [56]. Consequently, an increasing number of approaches [22, 56] fused multimodal data to localize root causes more effectively.

Nonetheless, unimodal and multimodal approaches encounter a significant limitation: striking a balance between performance and the manual labeling overhead. Existing methods can achieve impressive performance but usually require extensive high-quality labeled data. For example, DiagFusion [56] and Déjàvu [26] need to label each historical failure's root cause and failure type. However, obtaining sufficient labeled data is highly challenging for two main reasons. First, deep learning-based approaches typically necessitate prolonged data collection to acquire enough training data. This challenge is amplified by the frequent changes in microservice systems due to software and hardware updates, causing frequent data distribution shifts. Second, manually annotating such a large volume of training data requires intensive effort from the operator. A similar work RCLIR [6] shows that labeling 1000 root cause cases requires four experienced operators to spend nearly a month. No current approach can simultaneously guarantee high performance while reducing manual effort satisfactorily. Overcoming this limitation is imperative for effective root cause localization in continuously evolving microservice systems.

A promising method to address this limitation is to employ self-supervised learning (SSL) [31]. SSL enables models to extract supervisory signals from large amounts of unlabeled historical data through pretext tasks, reducing dependence on manual labels [19]. Common SSL tasks include reconstruction tasks, contrastive learning, prediction tasks, *etc.* (more details can be seen in Section 2.2). SSL has been successfully employed in many domains, including computer vision, natural language processing, and graph learning [18, 32]. Given the straightforward implementation and adaptation of reconstruction tasks, along with the advantages of graph neural networks for modeling the structure of microservice systems, we choose to use a Graph Autoencoder (GAE), which is a reconstruction task (more details can be seen in Section 3.1). However, to the best of my knowledge, SSL has not yet been effectively applied for localizing root cause instances of microservice failures due to three major challenges:

(1) **Challenge 1: Lack of an interpretable method to quantify root causes.** The results of failure root cause localization need to be interpretable to help operators take appropriate measures for failure mitigation. However, the SSL models (*e.g.*, GAE) often lack interpretability, which hinders operators from making the right decisions and reduces their trust in the localization results.

(2) **Challenge 2: The models lack continuous learning capability.** Once deployed, existing approaches cannot continuously learn and adapt to new data or tasks. However, new failures persistently emerge as the system operates, evolves, upgrades [38], receives maintenance [37], and undergoes changes [36]. As the system changes gradually, the performance of deployed models will become increasingly misaligned, eventually necessitating full retraining to restore efficacy. According to our investigations, operators can provide incremental feedback to the root cause localization system. This enables on-the-job learning to improve model performance dynamically. However, current methods are static and cannot effectively leverage operator feedback.

(3) **Challenge 3: The requirement of graph autoencoders for a large amount of historical training data.** Training a specific GAE model typically requires a significant amount of historical data, although these data do not require labeling. We have verified this in Fig. 3. Insufficient training samples make it challenging to ensure the model's effectiveness, consequently impacting the quality of the features. However, obtaining such data can be challenging in real-world scenarios, particularly when a system is newly online or undergoes substantial changes.

To address the aforementioned challenges, we propose *DeepHunt*, an interpretable failure root cause localization method based on GAEs. We devise an interpretable and learnable root cause score, which provides a quantified root cause probability for each instance, addressing Challenge 1. Furthermore, a feedback mechanism has been incorporated to tackle Challenge 2. During online localization, operators can contribute valuable feedback labels based on diagnostic results. *DeepHunt* can then fine-tune its parameters based on the feedback, enabling continuous learning. Lastly, in the GAE training process, we introduce a data augmentation module to address Challenge 3.

Our contributions are summarized as follows:

(1) We propose *DeepHunt*, a GAE-based method for failure root cause instance localization. *DeepHunt* achieves a zero-label cold start and demonstrates commendable performance without necessitating an abundance of labeled failure samples for training. Moreover, it enhances its generalization capabilities by incorporating a data augmentation module during training.

(2) We design a root cause score that combines reconstruction error and failure propagation pattern to execute an interpretable process for quantifying root causes. This overcomes the challenge of SSL's lack of interpretability.

(3) We design a feedback mechanism to ensure continuous fine-tuning of *DeepHunt* through operators' feedback. This addresses the challenge of the SSL model's lack of continuous learning capability. Additionally, we propose a ranking-oriented loss function, which performs better when dealing with the imbalance between the root cause instances and non-root cause instances.

(4) Extensive experiments on the datasets collected from two benchmark microservice systems demonstrate *DeepHunt*'s effectiveness and efficiency. The outcomes demonstrate that *DeepHunt* achieves a 90+% A@5 accuracy in both datasets, even when trained with merely 1% of labeled failure samples. *DeepHunt*'s implementation is publicly available[1] to promote transparency and reproducibility. We make the dataset D1 used in our work publicly available[2].

2 BACKGROUND

2.1 Microservice Systems and System Behavior Graphs

Microservice systems divide a large application into several small, autonomous services, with each service dedicated to fulfilling a specific business function. Each service can be independently deployed, extended, and managed, and communicate with each other through lightweight communication mechanisms such as remote procedure calls (RPC).

Referring to the example in Fig. 2 , we introduce some essential terms and concepts:

System instance. A microservice system consists of multiple types of instances, including microservice instances and host instances. We refer to them collectively as *system instances* (or *instances* for short). These system instances collectively constitute a microservice system and serve as a foundation for achieving high availability, scalability, and failure tolerance.

Dependency relationship. There are various dependencies between system instances, such as invocation relationships between microservice instances, deployment relationships between microservice instances and host instances, *etc.* The impact of a failure can propagate along the direction of dependency relationships, resulting in cascading failures and making online failure root cause instance localization more challenging.

Multimodal feature. To identify the root cause instances of failures, operators meticulously monitor the system and record monitoring data. Traces, logs, and metrics are three common modalities of monitoring data that stand as the three pillars of microservice systems' observability [56]. In this paper, we concentrate on these three modalities since they collectively encompass more extensive and comprehensive failure information. To fuse the three modalities together, we extract a unified vector representation from them as the *multimodal feature* for each system instance (see Section 4.2 for details).

System behavior graph. To depict the attributes of system instances and the diverse interdependencies among them, we conceptualize a microservice system utilizing a *system behavior graph (SBG).* A SBG is a directed graph, $G = (V, E, F)$. V is the set of all candidate instances in a microservices system. An edge $(v_i, v_j) \in E$ indicates an actual invocation or deployment from instance v_j to instance v_i, which can be interpreted as v_j depending on v_i. F is the feature vectors extracted from the multimodal monitoring data of each instance (see Section 4.2.2). In our work, we aggregate monitoring data on a minute-by-minute basis. This allows us to model the microservice system as an SBG every minute, taking into account various monitoring data and the relationships involving invocation and deployment. Considering that the features of nodes and edges in SBG vary over time, we construct an SBG every minute. Fig. 2 shows how to construct an SBG from a microservice system when a failure occurs. SBGs provide valuable insights into

[1] https://github.com/bbyldebb/DeepHunt
[2] https://github.com/bbyldebb/Aiops-Dataset

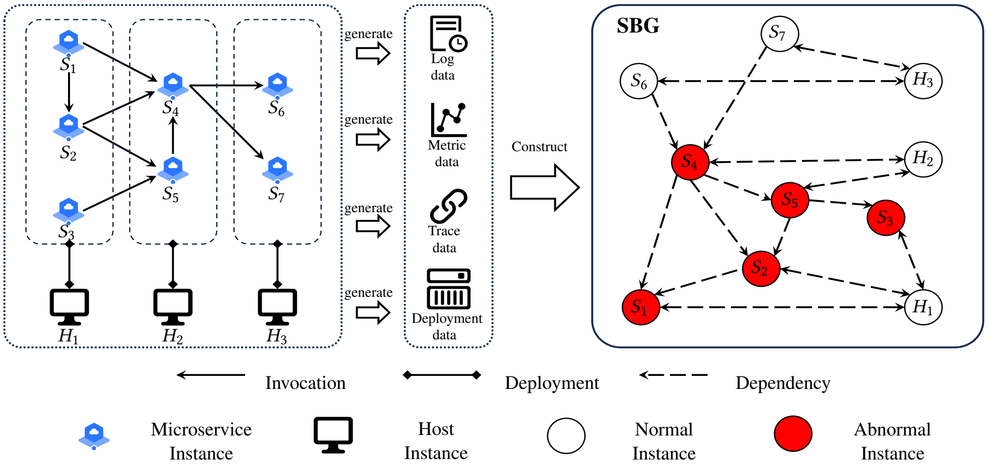


Fig. 2. A demonstration of constructing a *system behavior graph* (SBG) from a microservice system. In this illustration, the microservice system is simplified. The SBG's dependencies originate from invocation and deployment relationships. The instances within the SBG denote either microservice instances or host instances, each instance has a feature vector extracted from monitoring data as attributes.

the interactions and dependencies between various instances, helping to understand the system's overall behavior and potential points of failure.

Root cause instance. Root cause instances are the primary system instances that trigger system failures, such as S_4 in Fig. 2. These instances are responsible for system performance degradation, functional failures, or other issues. Given the intricate dependencies between system instances, the failure of a root cause instance can propagate to some other cases through these relationships, leading to widespread failures. Promptly identifying the root cause instance when a system failure occurs is crucial, enabling appropriate measures to be taken to resolve the issue and enhance system performance. We aim to identify the real root cause instances for failures in microservices systems.

2.2 Self-Supervised Learning and Graph Autoencoders

Self-supervised learning (SSL) is a machine learning paradigm that leverages the inherent information within data for training models, eliminating the need for manual labeling. In contrast to supervised learning, SSL does not depend on external labels; instead, it designs pretext tasks that enable model learnings [19]. These tasks generate pseudo-labels automatically, compelling the model to extract meaningful features from the data to address the problem. Common SSL tasks include reconstruction tasks [44], prediction tasks (*e.g.*, language modeling [42], image inpainting [41]), generative adversarial networks [10], and contrastive learning [43].

Among them, graph autoencoders (GAEs) are effective tools for handling graph data. GAEs combine graph neural networks (GNNs) and autoencoders (AEs) for representation learning and reconstruction of graph data. The training process of GAEs is accomplished by minimizing the reconstruction error, which measures the difference between the reconstructed and original graphs. Benefiting from the graph convolution mechanisms of GNNs (*e.g.*, GCN [21], GraphSAGE [12], GAT [47]), GAEs enable vertices to learn representations not only of themselves but also of their neighbors by aggregating and propagating information. This allows for capturing the dependency relationships within the graph structure.

In this paper, we model a microservice system as an SBG and employ GAEs to learn the underlying dependency patterns within and between the system instances. When a failure occurs, the reconstruction error of each system instance can serve as a crucial indicator for identifying the culprit system instances.

2.3 Problem Statement

A formal description of localizing the root cause instances of a failure in microservice systems is as follows. For a failure $\mathbb{F}$, given the trace data $\mathbb{T}$, log data $\mathbb{L}$, metric data $\mathbb{M}$, and deployment data $\mathbb{D}$ in time window before and after the failure, we extract the multimodal data's features and dependencies to construct a system behavior graph $G = (V, E, F)$, where V is the collection of system instances, E is the collection of dependencies, and F is the multimodal data's feature matrix. The objective is to find the set of root cause instances $\{V_{rc}\}$ which are responsible for this failure. To address the challenges outlined, the localization approach should 1) possess an interpretable root cause localization method, 2) be continuously upgraded and optimized based on operators' feedback, and 3) not require too much labeled historical data. In addition, please note that failure detection is not within our research scope in this paper, and there are already many methods available [11, 39, 56].

3 MOTIVATIONS

In this section, we introduce the motivation behind *DeepHunt*.

3.1 Why GAE?

To reduce the dependence of model training on manual labeling, we employ SSL [31]. As mentioned in Section 2.2, common SSL tasks include reconstruction tasks, prediction tasks, generative adversarial networks, and contrastive learning. Among them, prediction tasks require the manual design of complex and difficult tasks that may require domain expertise and experience [32]. Generative adversarial networks encounter challenges such as unstable training process, potential mode collapse, and high training complexity [45], all of which necessitate careful manual adjustments. As for contrastive learning, the method of selecting negative samples is likely to affect the performance of the model, requiring complex designs of sampling strategies [31]. In contrast, reconstruction tasks such as autoencoders do not require task-specific guidance and do not necessitate intricate task design, making their implementation and adaptation more straightforward [31].

Autoencoders are commonly used self-supervised learning models, and their reconstruction errors have been widely applied for metric anomaly detection [1, 2, 8, 16, 25, 50, 60]. Since microservice systems exhibit a graph structure, we employ GAE to capture complex structures and dependencies effectively.

3.2 Observation

When a failure occurs in a microservice system, it will not only affect one instance (*i.e.*, the root cause instance), but it can also propagate to other instances in multiple ways [48]. Thus, the root cause instance in a microservice system should exhibit both *local* and *global* features. We will demonstrate this idea through an empirical study of 63 failure cases collected from a microservice system for e-commerce.

3.2.1 Reconstruction Error. As mentioned earlier, the reconstruction error is commonly employed for anomaly detection as it indicates the extent to which data deviates from the expected normal pattern. Therefore, in this work, we explore

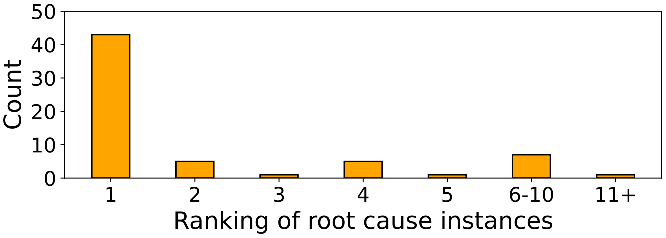


Fig. 3. The distribution of root cause instances according to reconstruction errors in 63 failure cases. GAE is trained using SBG samples constructed from normal uptime data.

whether it could be beneficial for root cause instance localization. We find that the reconstruction errors of the root cause instances indeed rank higher.

Specifically, we utilize the collected historical data to construct SBGs to train a GAE and reconstruction errors for each instance. Subsequently, we rank instances in descending order based on their reconstruction errors and obtain the ranking information for root case instances. The results are displayed in Fig. 3, where the root cause instances show high rankings (within the top five) in most failure cases, suggesting that the reconstruction error is an effective feature for root cause instance localization. Additionally, in approximately 30% of failure cases we collected, the root cause instance is not ranked first. Therefore, accurately localizing the root cause using only reconstruction errors is challenging.

3.2.2 Failure Propagation Pattern. As previously discussed, failures demonstrate a propagation behavior within microservice systems, where an initial failure in a root cause instance may extend to some other instances. Consequently, when a failure occurs, the monitoring data of multiple instances may exhibit anomalies, leading to elevated reconstruction errors that can complicate the process of identifying the root cause instance. For example, consider a partial SBG of a certain failure $\mathbb{F}$ depicted in Fig. 4, where nodes denote system instances, values adjacent to instances denote reconstruction errors, edges between instances denote dependencies, and arrows denote the direction of failure propagation. The root cause instance of this failure is S_4. However, instances S_1, S_2, S_3, and S_5 also exhibit abnormal behavior, influenced by the propagation of the failure. To make things worse, S_2 and S_5 exhibit higher reconstruction errors due to the more pronounced anomalies they present. This highlights that relying solely on reconstruction error for localizing the root cause instance is inadequate, as it does not account for failure propagation. To enhance the accuracy of root cause instance localization, it is imperative to further investigate the role of failure propagation pattern in conjunction with reconstruction error.

In real-world microservice systems, SBG is much more complex than what is shown in Fig. 4. Therefore, to accurately localize failure root causes, we need to combine reconstruction error and failure propagation pattern efficiently and effectively to jointly model them. Additionally, operators expect good interpretability, such as which instance becomes anomalous earlier, and how a failure propagates across different instances. This aids them in promptly determining whether the result is correct and taking corresponding measures to mitigate the failure.

3.2.3 Feedback. GAE is trained by reconstructing the SBGs constructed from historical data, a process that does not require manual labeling. However, the method will inevitably mislocalize for some failures, and it remains difficult to correctly localize the root cause instances of similar failures without feedback to help the method make corrections.

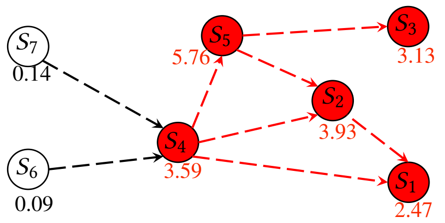


Fig. 4. An example of a partial SBG for a certain failure $\mathbb{F}$, where the truth root cause instance is S_4.

Failure	Root Cause Instances
$\mathbb{F}$	$\{S_4\}$

Fig. 5. The feedback label provided for failure $\mathbb{F}$.

Therefore, we expect the method can leverage labeled failure cases to enhance performance. For instance, in the failure case described in Fig. 4, providing the root cause instance label, such as Fig. 5, as supervisory information can help extract useful insights. Due to the small number of labeled historical failure cases in most scenarios, operators' feedback on root cause instances can help improve the method's performance.

4 DESIGN

4.1 Design Overview

To precisely and interpretably localize the root cause of the microservice system, we propose a multimodal-data based approach, *DeepHunt*. The framework of *DeepHunt*, as shown in Fig. 6, consists of four components: *SBG construction*, *offline training*, *interpretable online localization*, and *feedback*.

In SBG construction, *DeepHunt* unifies and fuses logs, metrics, and traces information, and then constructs SBGs by using the deployment topology. For interpretable online localization, *DeepHunt* proposes a root cause score to combine reconstruction error and failure propagation pattern (addressing challenge 1). In offline training, to solve the problem of insufficient labeled failure cases and normal training data (challenge 3), *DeepHunt* adopts GAE, a typical SSL method for graph data, and performs data augmentation. At last, *DeepHunt* achieves continuous learning and optimization through a feedback mechanism (addressing challenge 2).

4.2 SBG Construction

Current studies commonly employ two methods to fuse multimodal data: unifying them into standard events [55?, 56], or unifying them into vectors through feature extraction [22, 58]. Considering that events in various works have varying definitions and additional embedding operations are required before feeding them into the neural network, the latter approach is adopted in our work to ensure simplicity and generalization. In the *SBG construction* phase, *DeepHunt* commences with multimodal serialization to standardize data from diverse modalities into a time series, followed by a modal-wise feature extraction process. Finally, SBGs are constructed.

4.2.1 *Multimodal Serialization.*

Inspired by the work [22], we serialize different data modalities with the following rules.

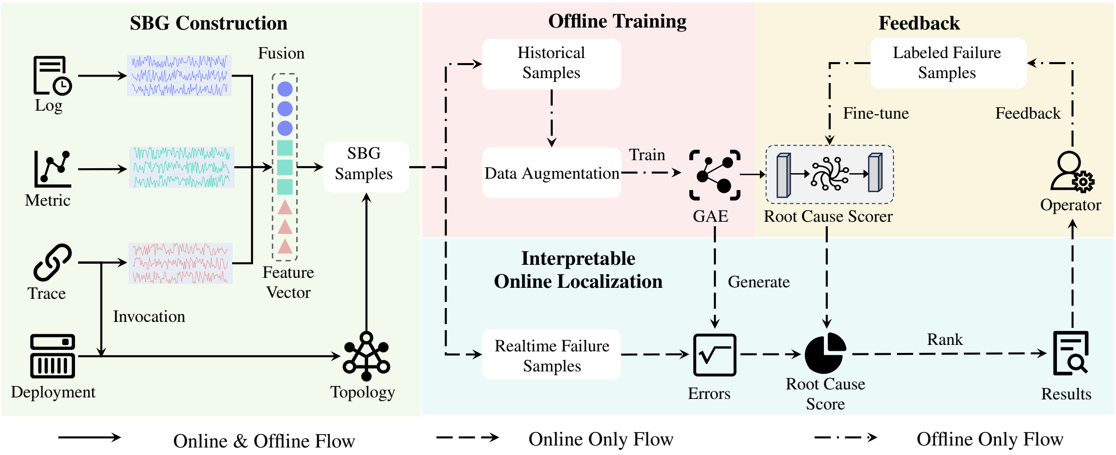


Fig. 6. The framework of *DeepHunt*.

Traces. The trace data consists of chain-structured records of user request paths, including details like latency and status codes. Inspired by previous work [30, 51], we extract features including latency, request count, and status codes for each callee instance. The structural information of the traces is extracted as dependencies that are used as partial edges of the SBG. These data are transformed into multivariate time series by computing metrics such as the average latency per minute, the total number of requests per minute, and the frequency of various status codes per minute. We extract the trace multivariate time series $H_{\text{trace}}^{(i)}$ for each instance i.

Logs. The logging behavior of microservices can be highly variable and dependent on developers' expertise, presenting challenges in ensuring consistent log semantics [14]. Moreover, extracting log semantics often requires computationally intensive natural language processing, which may hinder real-world applicability [22]. To maintain lightweight preprocessing, we focus on modeling log template occurrences rather than semantics. We utilize Drain [13] to parse logs into templates, removing variables from logs. To avoid excessive templates and sparse features, we group log templates based on frequency and fluctuation. Templates with low frequency and minimal fluctuations are consolidated, while high-frequency or highly fluctuating ones remain separate. For log templates that did not appear in the historical records, we handle them as follows. Given the infrequency of new templates in most scenarios, we pre-define a time series to track the frequency of occurrences of new log templates. However, if an abundance of new templates arises, it may necessitate retraining to update the feature engineering component accordingly. We then treat the occurrences per minute of a template group as a time series to construct a multivariate log time series. For each instance i, we extract the log time series $H_{\text{log}}^{(i)}$. This provides a compact representation capturing log-based temporal patterns and dynamics without heavy natural language processing.

Metrics. The metric data is inherently represented as a time series of performance indicators. We employ resampling and nearest-neighbor interpolation to standardize all metric intervals to one minute. Constructing multivariate time series simply involves aligning all metric data by timestamp. We extract the metric multivariate time series $H_{\text{metric}}^{(i)}$ for each instance i.

4.2.2 Modal-wise Feature Extraction. We perform z-score standardization [46] using a sliding historical window on multivariate time series $H_{\text{modal}}^{(i)}$, where modal $\in \{\text{trace}, \text{log}, \text{metric}\}$, to normalize different magnitudes. To simplify feature extraction and avoid excessive overhead, we directly treat the standardized data $\hat{H}_{\text{modal}}^{(i)}$ as features. Specifically, for each instance i at time t, the feature vector $F^{(i)}_{\text{modal}} = \hat{H}^{(i)}_{\text{modal}}(t)$. We take the union of all features. Next, we concatenate the features into a fused feature vector $F^{(i)} = \left(F^{(i)}_{\text{trace}} \middle\| F^{(i)}_{\text{log}} \middle\| F^{(i)}_{\text{metric}}\right)$. This achieves preliminary fusion across metrics, logs, and traces. Further inter-modality relationships are learned through the GAE model.

4.2.3 SBG Construction. As shown in Fig. 6, we extract the topology from the deployment relationships within the deployment data and the invocation relationships within the trace data to form the nodes and edges of the SBG. The feature vectors extracted from multimodal monitoring data for each instance are utilized as node attributes in the SBG. The SBG represents the state of a microservices system within a short period and evolves dynamically.

4.3 Offline Training

Consequently, GAE is designed to learn the system's normal patterns, generating elevated reconstruction errors as indicative features for root cause localization. During the *offline training* phase, *DeepHunt* trains a GAE using SBGs constructed from historical data.

4.3.1 Model Structure. The GAE in *DeepHunt* consists of an encoder and a decoder, each comprising several layers of graph neural networks. The operation of the k-th layer ($k = 1, 2, \dots, N$) of the encoder or decoder is formulated as

$$\begin{aligned} h^{(k)}_{\mathcal{N}(i)} &= agg\left(h^{(k-1)}_j, \forall j \in \mathcal{N}(i)\right), \\ h^{(k)}_i &= norm\left(\sigma\left(W^{(k)} \cdot concat\left(h^{(k-1)}_i, h^{(k)}_{\mathcal{N}(i)}\right)\right)\right). \end{aligned} \tag{1}$$

where $h^{(k)}_i$ is the k-th layer's representation of instance i, $\mathcal{N}(i)$ is the set of neighbors of instance i, agg is the operation of aggregating the features of the neighbors (*e.g.*, calculating the mean value), $concat$ is the operation of concatenating the feature vectors of the current instance, and $norm$ is the normalization operation, $W^{(k)}$ is the weight matrix of the k-th layer, and σ is the LeakyReLU activation function [35].

In each layer of the encoder and decoder, information about the neighbors of each instance is obtained through neighbor sampling and aggregation. The encoder takes the SBG as input and performs graph convolution operations to capture the dependencies within and between instances. The encoder gradually reduces the dimensionality of the instance representation and finally maps it to a low-dimensional latent space. The decoder takes the feature representations from the latent space and the structural information of the SBG as input. It then applies graph convolution operations to reconstruct the features of each instance.

4.3.2 Data Augmentation. Usually, increasing the number of training samples for GAE makes the reconstruction error more helpful in determining anomalous behaviors [3]. However, in certain scenarios, like newly deploying a system or undergoing significant changes, it is challenging to acquire adequate historical data for training in the short term. Enhancing the model's performance with limited training data is an important consideration, and data augmentation emerges as a common practice in this scenario.

Unlike traditional augmentation techniques used for images, such as rotation or cropping, SBGs, being graph data, require consideration of structural and feature modifications within the graph. In *DeepHunt*, GAE primarily focuses on instance features, thus leading towards augmenting the information within these instance features. Additionally, we conduct data augmentation aiming to introduce scenarios that might occur but are not included in the training set. However, altering the graph structure might introduce improbable scenarios that could mislead the model, like establishing invocation dependencies between entirely unrelated instances. Consequently, our choice learns towards augmenting changes in instance features.

To conduct data augmentation, we randomly mask features of each instance in the SBGs before feeding them into GAE. The data agumentation process is controlled by a probability called the masking rate. Obviously, the masking rate is an important parameter, and it should be neither too low nor too high. So, we demonstrate the effect of different masking rates on the model's performance in Fig. 10. Our core idea for augmentation is to mimic data absence by masking input features, reducing the possibility of the model overly relying on specific features during training, which compels the model to learn more robust and generalized features. The evaluation experiments in Section 5.3 validated the effectiveness of data augmentation.

4.3.3 Training. The training objective of GAE is to reconstruct feature vectors of instances in SBGs while minimizing reconstruction error. We use mean squared error (MSE) to measure the reconstruction error. Because microservice systems are typically stable most time, GAE learns the normal patterns of the systems through training.

4.4 Interpretable Online Localization

Even though we recognize that reconstruction error and failure propagation pattern can serve as features indicating the root cause (see in Section 3.2), quantifying how these two types of features relate to the root cause remains unknown. In this section, we introduce how *DeepHunt* provides an interpretable approach to localize the root cause. Our core idea is to calculate a root cause score for each instance and subsequently rank the instances based on it. This process consists of two parts: calculating reconstruction error and quantifying failure propagation pattern.

4.4.1 Calculate Reconstruction Error. When a failure occurs, the entire system has likely undergone some minor changes. Therefore, we typically analyze root causes not only at the moment of failure but also consider the data preceding the failure. This allows us to obtain reconstruction errors for multiple time intervals. Therefore, our initial consideration is to aggregate the reconstruction errors from multiple time intervals to obtain local features reflecting the state of instances. A simple and common method is to take the average of the reconstruction errors across all time intervals. However, the contribution of features from different time intervals to root cause determination may not be equal. Therefore, a more effective method is to apply weighted averaging to the reconstruction errors of each time interval using different importance weights. This can be achieved by employing a fully-connected layer (denoted as FC_1) in the process.

4.4.2 Quantify Failure Propagation Pattern. As mentioned in Section Section 3.2.2, the failure propagation pattern contributes to achieving more accurate root cause instance localization. Fortunately, regardless of the SBG's structural complexity, any anomalous instance has only four potential first-order upstream and downstream conditions, as depicted in Fig. 7. We remove the self-loop in the SBG because it does not significantly contribute to failure propagation analysis. These four conditions also apply to certain special scenarios of anomalous instances:

1) Instances lacking upstream or downstream instances can be regarded as having an upstream or downstream instance with a reconstruction error of 0;
2) Instances with bidirectional dependencies (such as S_7 and H_3 in Fig. 7) can be considered as having the same instance for both upstream and downstream;

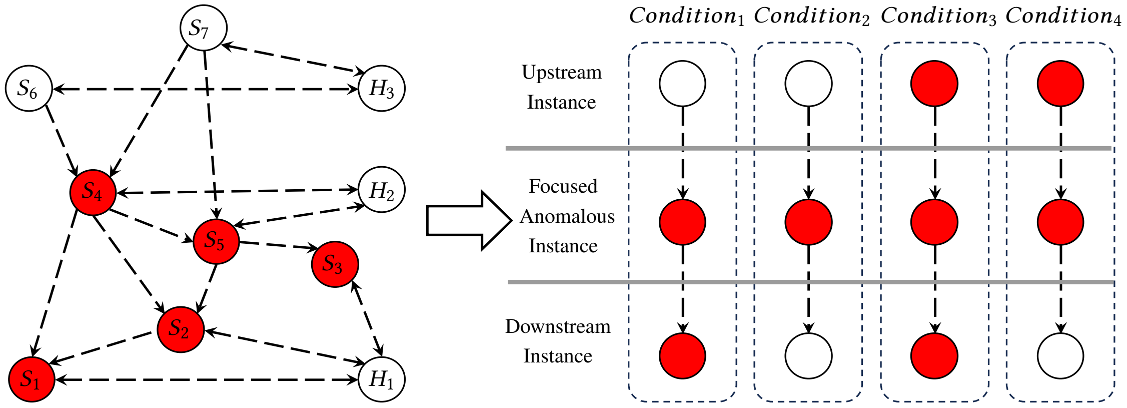


Fig. 7. The four possible states for any anomalous instance in an SBG. Operators consider that the focused anomalous instances in $State_1$ and $State_2$ are more likely to be the root cause instances.

3) Instances with multiple upstream or downstream instances (such as S_5, S_2 in Fig. 7) can be consolidated into one, with the reconstruction error being the maximum value, indicating the most anomalous part of the upstream or downstream instances.

After operators' confirmation, anomalous instances in $Condition_1$ and $Condition_2$ are more likely to be the root cause than those in $Condition_3$ and $Condition_4$, as the anomalies in the former two conditions are not caused by failure propagation.

Consequently, we can quantify the failure propagation pattern through the anomaly degrees (reconstruction errors) of each instance itself, its first-order upstream and first-order downstream instances, enhancing the root cause probabilities of the anomalous instances in $Condition_1$ and $Condition_2$. We combine the local features of each instance itself, its first-order upstream instance, and its first-order downstream instance into a three-dimensional vector. A graph aggregation layer (denoted as GA) is designed to quantify the failure propagation pattern of the SBG. Subsequently, we use another fully-connected layer (denoted as FC_2) to combine each dimension of GA's output with different importance weights, calculating the root cause score. FC_1, GA, and FC_2 mentioned above collectively form the root cause scorer.

4.4.3 Root Cause Score. As mentioned earlier, we require an interpretable approach to integrate reconstruction errors and failure propagation patterns for improved localization accuracy. Consequently, we employ a root cause scorer (comprising layers FC_1, GA, FC_2) to calculate the root cause score for each instance, facilitating later interpretability in the localization process. Throughout the learning procedure, *DeepHunt* ensures that the root cause score is positively associated with the root cause probability of each instance. Finally, *DeepHunt* performs online localization by sorting each instance in descending order based on their root cause scores. Specifically, the calculation of the root cause score

Table 1. A toy example of the initialization of W_1 and W_2.

Window_Size	W_1	W_2
10	$[0.1, 0.1, 0.1, \ldots, 0.1, 0.1]_{10}$	$[1, 0, 0]$

$RCS^{(i)}$ of instance i is as follows:

$$F^{(i)} = W_1 \cdot Errorswindow^{(i)},$$
$$F^{(i)}_{propagation} = \begin{pmatrix} F^{(i)} \\ F^{(i)}_{down} \\ F^{(i)}_{up} \end{pmatrix} = \begin{pmatrix} F^{(i)} \\ Agg\left(L^{(j)}, \forall j \in V^{(i)}_{down}\right) \\ Agg\left(L^{(k)}, \forall k \in V^{(i)}_{up}\right) \end{pmatrix}, \tag{2}$$
$$RCS^{(i)} = softmax(W_2 \cdot F^{(i)}_{propagation}).$$

where W_1 and W_2 represent the weight matrix of FC_1 and FC_2 respectively. $Errorswindow^{(i)}$ and $F^{(i)}$ denote the sequence of reconstruction errors in the time window we take and the overall reconstruction error of instance i, respectively. Agg denotes the method of aggregating the features of the upstream or downstream instances. In this study, we select the max function as we focus on the most anomalous parts during the feature aggregation process. $V^{(i)}_{down}$ and $V^{(i)}_{up}$ denote the sets of first-order downstream and upstream instances in the SBG. $F^{(i)}_{down}$ and $F^{(i)}_{up}$ represent the features aggregated from the features of instances in $V^{(i)}_{down}$ and $V^{(i)}_{up}$, respectively. $F^{(i)}_{propagation}$ denotes the quantified failure propagation pattern.

4.4.4 Interpretability of Online Localization. We elucidate the significance of each parameter within the root cause scorer to render the process of root cause score calculation transparent. FC_1 is employed to compute the reconstruction errors across multiple time intervals, reflecting the state of an instance within a time window. Consequently, the number of parameters in the weight matrix W_1 of FC_1 is equal to the length of the window (a hyperparameter Window_Size), with each parameter individually denoting the importance weight of different time intervals within that window. The role of FC_2 is to calculate the root cause score from the failure propagation pattern. Thus, the weight matrix W_2 of FC_2 contains three parameters, representing the importance weights of the reconstruction error of each instance, its first-order upstream instance, and its first-order downstream instance when calculating root cause score.

It is worth noting that for the calculation of the root cause score for each instance, W_1 and W_2 are shared. Under the initial conditions, we adopt a fixed initialization for W_1 and W_2 instead of random initialization, and a toy example is shown in Table 1. The fixed initialization serves as the foundation for *DeepHunt* to achieve a zero-label cold start. When no labels are available, *DeepHunt* computes root cause scores using initialized parameters. In this scenario, we assume uniform importance for each time interval within the time window and do not consider the upstream and downstream components in the failure propagation pattern when calculating root cause scores. At this stage, *DeepHunt* resembles an “inexperienced operator.” Subsequently, through feedback, it learns and adjusts weights from feedback samples provided by operators, gradually becoming “experienced.” After receiving the operator’s feedback, *DeepHunt* continues to update W_1 and W_2. The refined W_1 and W_2 after feedback fine-tuning are presented in Section 5.6.

4.5 Feedback

We implement a feedback mechanism that enables operators to interact with *DeepHunt*, providing valuable input to progress its performance. For a failure case, the operator can provide feedback information based on the output of

DeepHunt: confirming correctly localized cases and correcting wrongly localized ones. For cases where *DeepHunt* fails to localize the root cause, the operator can point out the real root cause instance(s) of the failure, as shown in Fig. 5. *DeepHunt* then translates the feedback information from operators into a label vector $Y = [0, 0, \ldots, 1, \ldots, 0]$, and the value of the i-th dimension indicates whether the i-th instance is a root cause (1 denotes root cause, and 0 denotes non-root cause). Note that the feedback phase can be periodically triggered or manually initiated.

Our training objective is to maximize the root cause score for root cause instances. However, in real systems, the number of non-root cause instances is significantly larger than the number of root cause instances. A typical cross-entropy loss function struggles to address such an imbalanced ratio of instance quantities. Although DéjàVu [26] introduced a weighted binary cross-entropy, it doesn't fully mitigate this issue. To address this, we propose a ranking-oriented loss function (referred to as *ranking loss*) that ignores the influence of irrelevant instances on the optimization direction:

$$L_s = -\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{K_j}\max\{RCS_j^{(i)} - RCS_j \cdot Y_j, 0\}. \tag{3}$$

where N denotes the total number of fine-tuned cases, K_j denotes the number of instances of the j-th case, Y_j denotes the true labels of the j-th case, RCS_j denotes the root cause score vector for all instances in the j-th case, and $RCS_j^{(i)}$ denotes the root cause score for the i-th instance within that case.

A loss function based on cross-entropy calculates the deviation between the output value and the true label (0 or 1) of each instance, resulting in the domination of loss values for non-root cause instances due to their overwhelming number. Consequently, the model tends to predict all instances as non-root causes. *The ranking loss* addresses this issue by calculating loss values only for instances ranked before the true root cause instances, and not for instances ranked after the true root cause instances, thus mitigating the impact of non-root instances that are numerically dominant. By minimizing *the ranking loss*, the model gradually optimizes towards prioritizing the ranking of root cause instances before non-root cause instances, aligning with the objective of our work.

5 EVALUATION

In this section, we evaluate the performance of *DeepHunt* using the datasets collected from two microservice systems. We aim to answer the following research questions (RQs):

RQ1: How effective is *DeepHunt* in failure root cause instance localization?
RQ2: Does each component of *DeepHunt* have significant contributions to *DeepHunt*'s performance?
RQ3: Is the computational efficiency of *DeepHunt* sufficient for failure diagnosis in the real world?
RQ4: What is the impact of different hyperparameter settings?
RQ5: How do the parameters of *DeepHunt*'s interpretability module change after fine-tuning with feedback?

5.1 Experimental Setup

5.1.1 Dataset. To evaluate the performance of *DeepHunt*, we conduct extensive experiments on two datasets D1 and D2 collected from two microservice systems under different business backgrounds and architectures. Detailed information is listed in Table 2. The systems that produce D1 and D2 are as follows:

(1) D1. D1 is collected from a simulated e-commerce system with microservice architecture. The system comprises 46 system instances, including 40 microservice instances and 6 virtual machines. Its pattern of user requests is consistent with that of a real-world e-commerce system. Additionally, the failure cases in this dataset are derived

Table 2. Detailed information of datasets.

Dataset	# Instances	# Normal	# Failure	# Records		Failure Types
D1	46	3,714	210	trace log metric	44,858,388 66,648,685 20,917,746	Container hardware failure Container network failure Node CPU failure Node disk failure Node memory failure
D2	18	12,297	133	trace log metric	214,337,882 21,356,870 12,871,809	JVM memory failure JVM CPU failure Container memory failure Container CPU failure Container network failure Container disk failure

from real-world failures and are replayed in batches. The recorded failures were then labeled with their respective root cause instances. We have opened source the raw data and root cause labels of failures for D1[3].

(2) D2. D2 is collected from the management system of a top-tier commercial bank. The system comprises 18 system instances, including web servers, application servers, databases, and dockers. Due to the non-disclosure agreement, we cannot make this dataset publicly available. Two experienced operators examined the failure records from January 2021 to June 2021 and labeled the root cause instances of each failure. The labeling process was conducted separately by each operator, and they cross-checked their labels with each other to ensure consensus. This dataset has been used in the International AIOps Challenge 2022 [4].

5.1.2 Baseline Methods. We select nine advanced methods as the baseline methods, including non-deep learning-based methods (*i.e.*, MicroHECL [29], MicroRank [52], AutoMAP [34], TraceRCA [24], Microscope [27], RCD [17]), which employ techniques such as traditional machine learning, statistical models, or graph algorithms; and three supervised deep learning-based methods (*i.e.*, DéjàVu [26], Eadro [22], DiagFusion [56]). More details can be found in Section 7. Among the baseline methods, MicroHECL, MicroRank, and TraceRCA utilize trace, AutoMAP, RCD, Microscope, and DéjàVu utilize metric, and Eadro and DiagFusion utilize the three modalities of data including trace, log, and metric. We configure the parameters (*e.g.*, significance level, feature dimension) of all these methods according to their original settings depicted in the above works.

5.1.3 Evaluation Metrics. As stated in Section 2.3, *DeepHunt* aims to localize the root cause instances for failures. We carefully choose evaluation metrics to better reflect the comprehensive performance of all selected methods. More specifically, we employ *Top-k accuracy* (A@k) and *Top-5 average accuracy* (Avg@5) as the evaluation metrics. A@k quantifies the probability that the top-k instances output by each method indeed contain the root cause instance. Formally, given A as the test set of failures, $|A|$ as the size of the test set, RC_t^a as the ground truth root cause instance of failure a, $RC_p^a[k]$ as the top-k root cause instances set of failure a generated by a method, A@k is defined as:

$$A@k = \frac{1}{|A|}\sum_{a\in A}\begin{cases}1, & \text{if } RC_t^a \in RC_p^a[k], \\ 0, & \text{otherwise.}\end{cases} \tag{4}$$

[3] https://github.com/bbyldebb/Aiops-Dataset

[4] https://aiops-challenge.com/

Avg@5 evaluates a method's overall capability in localizing the root cause instance. In practice, operators often examine the top five results. Avg@5 is calculated by:

$$Avg@5 = \frac{1}{5} \sum_{1 \le k \le 5} A@k. \tag{5}$$

5.1.4 Implementation. We implement *DeepHunt* and baselines with Python 3.9.13, PyTorch 1.12.1, scikit-learn 1.1.2, and DGL 0.9.0 respectively. We run the experiments on a server with 12 × Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz and 128G RAM (without GPUs). We repeat every experiment five times and take the average result to reduce the effect of randomness.

5.2 Overall Performance (RQ1)

In *DeepHunt*, we utilize the data from non-failure periods to train the GAE. The failure cases with root cause labels are utilized to evaluate the effectiveness (test set) and simulate the feedback information provided by operators (training set). We split the failure cases into training and test sets chronologically. Specifically, for the evaluation in Table 3, we allocate the first 30% of failure cases as the training set and the remaining 70% as the test set. To demonstrate the effectiveness of *DeepHunt*, we compare its performance on both D1 and D2 with the baseline methods. The performance comparison result is shown in Table 3.

The *labeling ratio* in the table indicates the percentage of samples used for supervised learning. *DeepHunt* achieves the best performance overall. Without the utilization of labels (the labeling ratio is 0%), *DeepHunt* has already achieved good performance. With a labeling ratio as low as 1%, *DeepHunt* performs closely rivals or even surpasses most baseline methods. As mentioned earlier, *DeepHunt* does not necessarily require a large amount of labeled data to start and even can initiate with a zero-label cold start (see in Section 4.4.4). As the number of feedback samples increases, the localization accuracy of *DeepHunt* gradually improves. Take 30% labeling ratio as an example, supervised methods begin to exhibit a certain level of root cause instance localization capability. *DeepHunt* outperforms all baseline methods, demonstrating an improvement in A@5 ranging between 16% to 455%.

Compared to baseline methods that do not utilize supervised information, *DeepHunt* learns from historical runtime data and failure cases to enhance the accuracy of root cause localization. Additionally, the limitations in robustness to noise restrict the accuracy of these unsupervised methods. In scenarios with limited labels, *DeepHunt* offers the following two advantages compared to supervised baseline methods: 1) it utilizes SSL to learn normal patterns from historical runtime data and extracts reconstruction errors as effective features for root cause instance localization; 2) its parameters of both the GAE (trained through SSL) and the root cause scorer are well-initialized, relying less on supervised manual labels of historical failure cases.

Since *DeepHunt* is a deep learning-based method, we pay extra attention to how it compares with other deep learning-based methods. We focus on two main points:

1) How does each method perform under different labeling ratios? We conduct experiments using supervised samples ranging from 0% to 50% and present the results for Avg@5 in Fig. 8. The experimental results reveal that *DeepHunt* achieves remarkable performance with limited supervised information. As the labeling ratio increases, *DeepHunt* shows an upward trend in its performance. However, the improvement becomes less significant once the labeling ratio reaches a certain threshold, such as 1% in D1 and 25% in D2. This suggests that *DeepHunt* does not necessarily require a large number of supervised information for optimal performance. Moreover, *DeepHunt* consistently delivers higher accuracy with the same labeling ratio compared to other methods. This indicates that *DeepHunt* is highly

Table 3. Effectiveness of root cause instance localization. ("-" means this method does not need labeled samples for training.)

Method	D1					D2				
	Labeling ratio	A@1	A@3	A@5	Avg@5	Labeling ratio	A@1	A@3	A@5	Avg@5
DeepHunt	0%	0.780	0.898	0.959	0.889	0%	0.445	0.772	0.903	0.716
	1%	0.795	0.905	**0.966**	0.894	1%	0.498	0.781	0.910	0.741
	25%	0.797	0.902	**0.966**	0.895	25%	0.783	0.935	0.944	0.900
	30%	**0.803**	**0.912**	**0.966**	**0.898**	30%	**0.785**	**0.936**	**0.946**	**0.901**
DéjàVu	30%	0.473	0.701	0.793	0.670	30%	0.583	0.733	0.817	0.714
Eadro	30%	0.310	0.446	0.484	0.413	30%	0.214	0.386	0.454	0.361
DiagFusion	30%	0.333	0.500	0.648	0.493	30%	0.398	0.552	0.750	0.532
MicroHECL	-	0.091	0.232	0.386	0.236	-	0.068	0.240	0.414	0.242
MicroRank	-	0.144	0.218	0.259	0.209	-	0.208	0.365	0.541	0.369
AutoMAP	-	0.279	0.574	0.729	0.531	-	0.128	0.271	0.421	0.283
TraceRCA	-	0.243	0.310	0.338	0.302	-	0.241	0.368	0.459	0.362
Microscope	-	0.074	0.113	0.227	0.127	-	0.030	0.078	0.241	0.117
RCD	-	0.095	0.124	0.174	0.128	-	0.106	0.167	0.220	0.170

effective in failure root cause instance localization, making it a valuable option for practical deployment, particularly in scenarios where obtaining a large amount of labeled data is challenging or time-consuming.

2) **How stable is each method's performance in multiple experiments on the same training data?** We repeat the experiments for each method five times without setting the random seed. To ensure the effectiveness of supervised baseline methods, we set the supervision rate to 30%. We then visualize the results using box plots in Fig. 9. It clearly indicates that the stability of *DeepHunt* is significantly higher than that of the other methods. We attribute this higher stability to the fact that, unlike other methods that rely on random initialization, the large number of uptime data used to train the GAE provides good initialization parameters for *DeepHunt*. As a result, when the training sample size is limited, the model experiences notably less uncertainty from stochastic operations like stochastic gradient (SGD) compared to other methods. This robustness and stability further highlight the effectiveness and reliability of *DeepHunt* in failure root cause instance localization.

Table 4. Contributions of components.

	Method	A@1	A@3	A@5	Avg@5
D1	*DeepHunt*	**0.795**	**0.905**	**0.966**	**0.894**
	C1	0.759	0.901	0.961	0.882
	C2	0.488	0.770	0.814	0.706
	C3	0.780	0.898	0.959	0.889
	C4	0.544	0.829	0.891	0.776
	C5	0.776	0.898	0.959	0.888
D2	*DeepHunt*	**0.498**	**0.781**	**0.910**	**0.741**
	C1	0.426	0.774	0.871	0.699
	C2	0.447	0.726	0.873	0.687
	C3	0.445	0.772	0.903	0.716
	C4	0.138	0.436	0.776	0.457
	C5	0.432	0.765	0.896	0.706

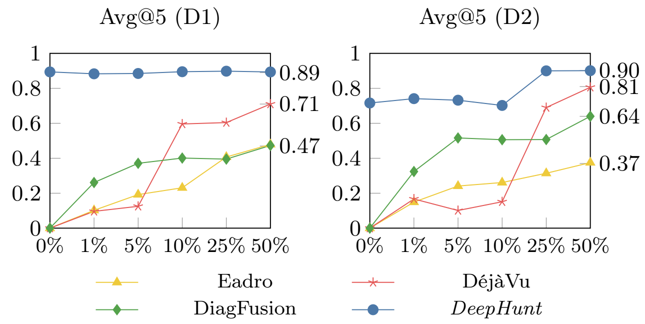


Fig. 8. Performance of the deep learning-based methods with different labeling ratios.

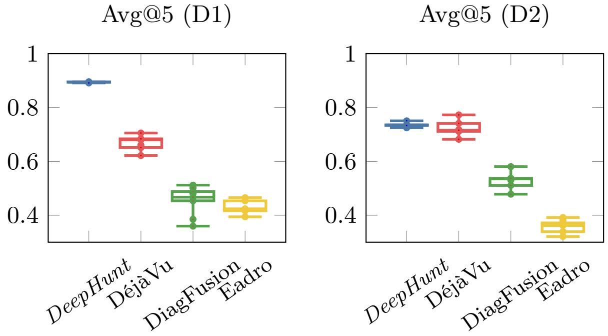


Fig. 9. Stability of the deep learning-based methods.

5.3 Ablation Study (RQ2)

To evaluate the effects of the five key technique contributions of *DeepHunt*: 1) the data augmentation module; 2) the GAE; 3) the feedback phase; 4) the root cause scorer; 5) the ranking loss, we create five variants of *DeepHunt*. **C1:** Remove the data augmentation module. **C2:** Replace GAE with an autoencoder built upon non-graph neural networks. **C3:** Remove the feedback phase. **C4:** Replace our root cause scorer with random forest regression [5]. **C5:** Replace our *ranking loss* with loss function proposed in DéjàVu [26].

Table 4 lists that *DeepHunt* outperforms all the variants on D1 and D2, demonstrating each component's significance. In **C1**, the decrease in accuracy highlights the effectiveness of the data augmentation module. In **C2**, the replaced autoencoder disregards the inter-instance dependency information while learning the system's normal pattern, resulting in a decline in feature quality and ultimately impacting *DeepHunt*'s performance. **C3** demonstrates the continuous learning capability of *DeepHunt*, enabling it to fine-tune itself through feedback from operators continually. **C4** shows that the necessity of providing interpretability to *DeepHunt* is affirmed, as other interpretable traditional methods (such as random forest) fail to deliver satisfactory performance. In **C5**, our proposed ranking-oriented loss function exhibits superior advantages in handling the imbalance between root cause instances and non-root cause instances compared to the loss function proposed in DéjàVu.

Furthermore, *DeepHunt* can adapt to the dynamic addition and removal of instances. To verify this, we conduct additional experiments in which we manually introduce changes in the number of instances. Specifically, we randomly remove 20% of the instances in the training set to simulate the addition of instances in the test set and randomly remove 20% of the non-root cause instances in the test set to simulate the removal of instances in the test set. We create other four variants of *DeepHunt*. **C6:** Randomly remove 20% of instances in the training set. **C7:** Remove the feedback phase in the context of **C6**. **C8:** Randomly remove 20% of instances in the test set. **C9:** Remove the feedback phase in the context of **C8**. For each variant, we repeat the experiment five times and average the results.

Table 5. Performance under the dynamic addition and removal of instances.

	Method	A@1	A@3	A@5	Avg@5
D1	*DeepHunt*	0.795	0.905	0.966	0.894
	C6	0.788	0.904	0.966	0.892
	C7	0.781	0.898	0.959	0.890
	C8	**0.806**	**0.918**	**0.978**	**0.908**
	C9	0.801	0.913	0.966	0.904
D2	*DeepHunt*	0.498	0.781	0.910	0.741
	C6	0.473	0.766	0.886	0.721
	C7	0.439	0.768	0.901	0.715
	C8	**0.523**	**0.815**	**0.916**	**0.763**
	C9	0.482	0.807	0.925	0.757

The results are shown in Table 5. The outcomes of C6 and C8 indicate that the dynamic addition and removal of instances have little impact on the accuracy of *DeepHunt*. Notably, removing instances in the test set (C8) reduces the number of candidate instances, thereby decreasing the difficulty of localization, which increases accuracy instead. The accuracy of C7 and C9 is lower than that of C6 and C8, respectively, suggesting that the feedback phase positively affects *DeepHunt*'s adaptation to the dynamic deletion of instances.

5.4 Efficiency (RQ3)

We record the running time of all methods and compare them in Table 6. It shows that *DeepHunt* can localize the root cause instances of a failure within 1 second on average online. This demonstrates that *DeepHunt* can meet the needs of online diagnosis.

Offline training time is not sensitive because it does not need to be retrained frequently. However, we note a significant difference in *DeepHunt*'s offline training time between the two datasets. Offline training time is typically affected by feature engineering, model structure, optimization algorithms, and hyperparameter settings. We use the same model structure, optimization algorithm, and similar hyperparameter settings on datasets D1 and D2, so they cannot be the key factors. Specifically, we use the same model structure for both datasets to ensure consistent model complexity. We choose Adaptive Moment Estimation (Adam) [20] as the optimization algorithm for its adaptive learning rate, reducing the need for hyperparameter tuning and ensuring efficient and stable model convergence. Feature engineering time mainly depends on data volume and complexity, which differs between datasets. As presented in Table 2, there is a significant difference in the number of samples used for GAE training (# Normal) between the two datasets. Additionally, the amount of trace data in D2 is an order of magnitude larger than that in D1. These result in a greater time overhead for constructing SBG samples on the D2 dataset. In summary, feature engineering is the key factor influencing *DeepHunt*'s offline training time.

Table 6. Training time (Offline) and average time to diagnose a failure case (Online). ("-" means this method does not need training)

Method	D1		D2	
	Offline(s)	Online(s)	Offline(s)	Online(s)
DeepHunt	629.892	0.169	1961.616	0.262
DéjàVu	429.048	0.318	381.421	0.192
Eadro	1126.162	5.370	399.251	0.432
DiagFusion	613.919	4.145	308.020	3.297
MicroHECL	-	12.233	-	4.193
MicroRank	-	42.540	-	28.877
AutoMAP	-	3.845	-	0.667
TraceRCA	-	34.731	-	92.956
Microscope	-	26.685	-	8.548
RCD	-	27.072	-	19.283

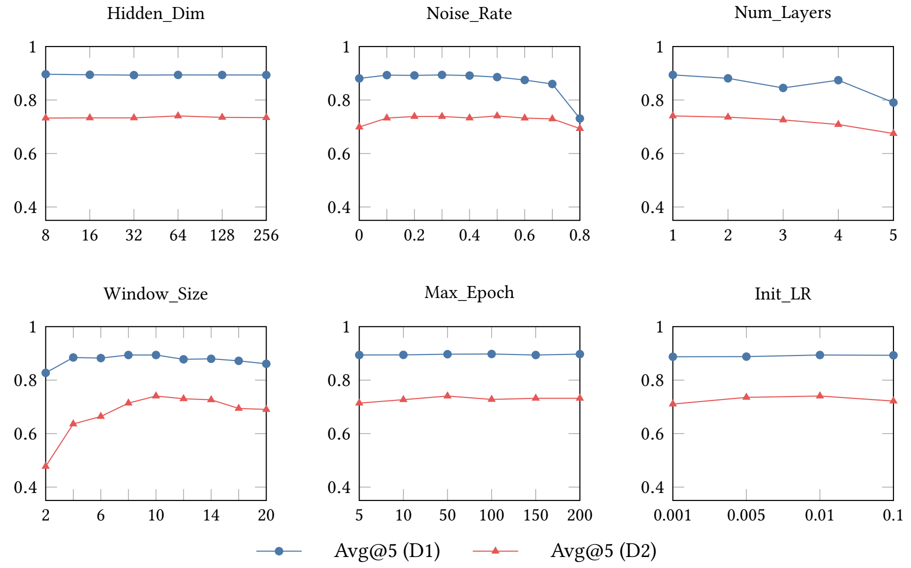


Fig. 10. The effectiveness of *DeepHunt* under different hyperparameters.

It's worth noting that Microscope has been analyzing online for longer than *DeepHunt*, even though it's a simple metric-based approach. The computational complexity of the PC algorithm used by Microscope is exponentially related to the number of nodes. When the number of metrics increases, the PC algorithm needs to perform more conditional independence tests and graph searches, which increases the computational burden. There are quite a number of metrics in datasets D1 and D2, and the possible combinations of graph structures increase with them, which increases the complexity of the search and causes the algorithm to take more time to find the optimal graph structure. So the online time of Microscope is longer compared to *DeepHunt*.

5.5 Hyperparameter Sensitivity (RQ4)

We discuss the effect of six hyperparameters of *DeepHunt*. Fig. 10 shows how Avg@5 changes with different hyperparameters.

The number of neurons in the hidden layer (Hidden_Dim). The performance of *DeepHunt* demonstrates relative stability when varying the number of neurons in the hidden layer. It is not a sensitive parameter for *DeepHunt*.

The ratio of masked features (Noise_Rate). Randomly masking a certain proportion of features indeed leads to an improvement in the performance of *DeepHunt*. However, when the proportion of injected noise is excessively high, it can compromise the characteristics of the original samples, resulting in a notable decline in performance.

The number of hidden layers in encoder/decoder (Num_layers). As the number of hidden layers increases, the model becomes more complex, resulting in overfitting with limited samples. *DeepHunt* experiences a degradation in performance when this parameter becomes excessive. This parameter needs to be set based on the specific sample conditions, and we set it to 1 in our study.

The size of the data time window around the failure occurrence used for root cause localization (Window_Size). *DeepHunt* exhibits an overall trend of performance improvement followed by a decline with varying window sizes. Clearly, a window that is too small fails to encompass complete failure information, while an excessively large window contains too much irrelevant data. In our study, setting it to 10 proves to be a suitable choice.

The maximum number of epochs for fine-tuning during feedback (Max_Epoch). We implement an early stop strategy during fine-tuning, which might lead to the performance of *DeepHunt* being insensitive to Feedback_Epoch. Nevertheless, we still advise against setting this parameter excessively high, especially when dealing with small sample sizes.

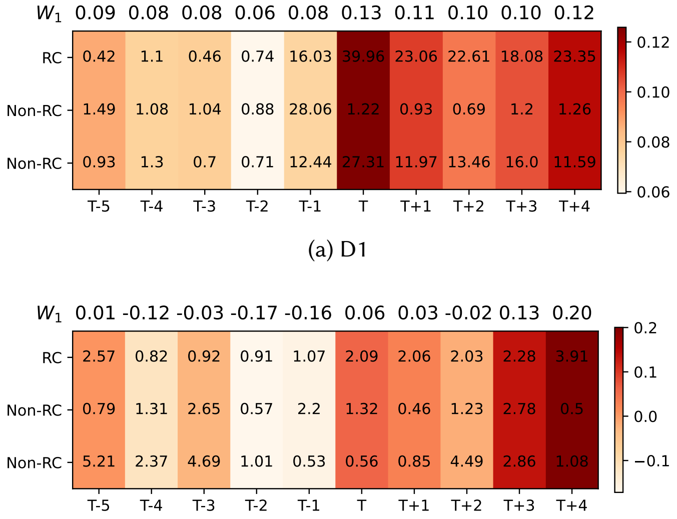
The initialization learning rate for fine-tuning during feedback (Init_LR). We employ the adaptive learning rate algorithm Adam [20] during fine-tuning. Nonetheless, the init_LR remains a critical hyperparameter, influencing the speed and effectiveness of convergence in fine-tuning. In our study, a learning rate setting of 0.01 is deemed an appropriate choice.

5.6 Interpretation of Localization Results (RQ5)

We have described the interpretability of *DeepHunt* for root cause instance localization in Section 4.4.4. In this section, we show in detail the parameters W_1 and W_2 obtained by fine-tuning the root cause scorer in D1 and D2 during feedback respectively, where $window\_size$ is set to 10.

The FC_1 layer of the root cause scorer, used to calculate the overall reconstruction error of each instance, initializes each parameter in W_1 to 0.1. In Fig. 11, we present the heatmaps of the fine-tuned parameters for the two datasets, each exhibiting a failure case. Intriguingly, the results differ: in D1, the largest weight appears at the first minute after the failure onset, whereas in D2, the largest one appears at the fifth minute the failure occurs. We analyze that this scenario relates to the observational characteristics inherent in the datasets. In D1, most root cause instances exhibit anomalous fluctuations earlier than the non-root cause instances. Conversely, in D2, most root cause instances tend to persist in anomalies for an extended period compared to the non-root cause ones.

The GA layer and the FC_1 layer quantify the failure propagation pattern for each instance. They utilize three parameters $W_2 = (\alpha, \beta, \gamma)$, representing the importance weight of self, downstream, and upstream dependencies, respectively. They are initialized as $(1, 0, 0)$, respectively, signifying no aggregation of dependencies by default. We froze the parameter α and exclusively fine-tune β and γ. Table 7 shows the fine-tuned parameters for the two datasets.



(a) D1

(b) D2

Fig. 11. The heatmap of weights W_1 of FC_1 layer within a window. Values within cells represent reconstruction errors, while color shades illustrate weight magnitude. T-/+i represents the i-th minute before/after the failure. “RC” denotes root cause instances, “Non-RC” denotes non-root cause instances.

Table 7. The parameters W_2 of FC_2 layer in the root cause scorer after fine-tuning.

Dataset	α	β	γ
D1	1.000	0.020	0.009
D2	1.000	0.133	-0.002

The outcomes indicate a higher significance of downstream dependencies over upstream dependencies, and the weight of the upstream ones even displays negativity in D2. This suggests that the local features of upstream instances are less important for quantifying root causes, to the extent that higher ones diminish the probability of being a root cause.

6 DISCUSSION

6.1 Limitations and Future Works

When a failure occurs, it is crucial to swiftly localize the instance of the culprit. Operators often require accurate and detailed information to pinpoint the root cause of the failure. This includes not only identifying the location of the root cause instance but also obtaining more specific results, such as the failure type. However, *DeepHunt* cannot currently determine the failure type. This limitation arises because the reconstruction errors extracted by the GAE reflect the anomalies of the instance as a whole, but it’s difficult to capture the nuanced failure details within the instance. Addressing this limitation would be an important area for future improvements in *DeepHunt*.

Based on the work of *DeepHunt*, a potential avenue for future work could involve training a failure-type classifier using a smaller amount of labeled data. The accurate localization provided by *DeepHunt* helps narrow down the scope of determining the root cause, ideally requiring only the data from the root cause instance to train the classification model rather than the data of the entire system. Additionally, the GAE serves another purpose of performing feature dimensionality reduction, allowing for the extraction of a high-quality, low-dimensional representation of the initial features. This dimensionality reduction can help reduce the number of parameters required for the failure type classifier.

6.2 Concerns about Deployment and Validity

Deploying *DeepHunt* in real-world microservice systems may encounter some concerts: (1) *DeepHunt* needs to adapt to dynamic microservice architectures. *DeepHunt* utilizes GraphSage layers in the GAE model that can learn the aggregation of neighboring nodes. GraphSage enables individual nodes to update their representations by leveraging information from neighboring nodes while facilitating the model's adaptation to diverse neighbor structures and characteristics across nodes. This flexibility enables *DeepHunt* to handle the dynamic increase and decrease of instances in real-world deployments. (2) Incomplete monitoring of modalities. Some production systems may not monitor all three modalities (trace, log, and metric) simultaneously. *DeepHunt* integrates the various modalities into a unified time-series data representation and subsequently extracts features for fusion. This approach ensures that *DeepHunt* is not reliant on any specific modal data and can accommodate any combination of the three modalities. However, it is important to note that the lack of monitoring data from certain modalities could compromise the observability of failures and subsequently reduce the accuracy of localization.

This study faces two main threats. Firstly, the limited size of the D1 and D2 datasets used in this study. These datasets may be less complex and dynamic compared to real-world industrial microservice systems. Secondly, we evaluate *DeepHunt* on two datasets, which cannot represent all microservice systems. However, it is important to note that the two datasets are still valuable for evaluation. The datasets are sourced from different systems with diverse architectures and business operations. The validity and generalizability of *DeepHunt* are supported by the successful results obtained in our experiments. So, we believe that *DeepHunt* holds promise for application in larger industrial microservice systems with more complex failure scenarios.

7 RELATED WORK

Table 8. Comparison of existing representative methods. DL-based is short for deep learning-based. M, L, and T are short for Metric, Log, and Trace.

Method	DL-based	Modality	Pros	Cons
DéjàVu	✓	M	• Providing fine-grained failure diagnosis for recurring failures; • An interpretable module is provided.	• Requiring a large number of labeled failure cases for training. • Not fusing the multimodal data.
Eadro	✓	M, L, T	• Fusing multimodal data; • Investigating the close connection between detection and localization.	• Requiring a large number of labeled failure cases for training. • The number of output layer neurons must be equal to that of system instances.
DiagFusion	✓	M, L, T	• Fusing the multimodal data; • Overcoming the challenge of unbalanced types of failures.	• Requiring a large number of labeled failure cases for training.
MicroHECL	×	T	• Not requiring labeled failure cases for training. • Based on interpretable methodologies.	• Lack of a learning process from historical data, limited accuracy; • Not fusing the multimodal data.
MicroRank	×	T		
AutoMAP	×	M		
TraceRCA	×	M		
Microscope	×	M		
RCD	×	M		

Non-deep learning-based methods. Many studies aim to capture the interactions between system components during failures by proposing dependency graphs. Examples of such works include MicroRCA [49], MS-Rank [33], and its extension AutoMAP [34]. Some works construct more fine-grained graphs to capture causal relationships between metrics, *e.g.*, MicroCause [39], Microscope [27], and RCD [17]. However, the effectiveness of these approaches heavily relies on the accuracy of the relational graphs and the appropriate setting of parameters. This reliance on graph accuracy and parameter tuning reduces their applicability and limits their effectiveness in real-world scenarios. MEPFL [59], TraceRCA [24], MicroHECL [29], and MicroRank [52] utilize trace information to localize the root cause service. However, these approaches often focus more on the global characteristics of the system and may overlook the local characteristics of individual service instances. PDiagnosis [15] combines metrics, logs, and traces to identify root causes. It employs lightweight anomaly detection in all three modalities to detect anomalous patterns. Based on a voting strategy, the most severe component is selected as the root cause. However, PDiagnosis does not take into account the topological characteristics of the microservice system. Nezha [53] converts multimodal data into a unified event representation and extracts event patterns by constructing and mining the event graph. It then compares event patterns between failure-free and failure-occurrence phases to localize the root cause interpretively. Nezha primarily localizes root causes in code regions and resource types, differing somewhat from the instance-level localization approach in this paper. ShapleyIQ [23] employs multimodal data to build a causal graph for root cause localization via counterfactual evaluation and Shapley values. It utilizes a first principles model based on physical laws and historical observations to evaluate counterfactual effects. However, this method relies on constructing physical law-based models, whose accuracy hinges on precise assumptions about system behavior. Deviations from these assumptions may result in inaccurate estimations of causal relationships.

Deep learning-based methods. In recent years, there has been a growing trend in using graph neural networks (GNNs) to capture and learn the topological features of microservices. DéjàVu [26] learns metrics features and topological features of microservice systems using Gated Recurrent Unit (GRU) [7] and Graph Attention Networks (GAT) [47] for fine-grained diagnosis of recurring failures. Eadro [22]unifies data of different modalities into vectors and performs joint training for anomaly detection and root cause localization. DiagFusion [56] unifies data from different modalities into events, performs unified embedding representation, and learns from historical failure cases to identify root cause instances and failure types. However, all these methods have a limitation in that they require a large number of high-quality labeled failure cases for method training; otherwise, it is difficult to achieve good performance. Furthermore, Eadro and DiagFusion have a specific requirement where the number of output neurons should equal the number of instances in the system. This constraint limits their applicability in scenarios where the number of nodes dynamically changes, such as in systems with dynamic scaling or evolving architectures.

We compare existing representative methods in Table 8, summarizing their classification, data modalities used, pros, and cons. *DeepHunt* refines the cons of these methods, summarized as 1) learning from historical unlabeled data and feedback from failure cases; 2) reducing the requirement for large amounts of labeled data; 3) adapt to the dynamic increase and decrease of instances; 4) providing interpretability for results.

8 CONCLUSION

In this work, we conduct an extensive study aiming to enhance the effectiveness of failure root cause instance localization while reducing reliance on heavily labeled data. Leveraging the principles of self-supervised learning, particularly Graph Autoencoder (GAE), we propose *DeepHunt*. By integrating reconstruction errors and failure propagation patterns (upstream-downstream relationships), *DeepHunt* introduces the root cause score to measure root causes interpretably. Furthermore, *DeepHunt* achieves zero-label cold start and continuous ongoing refinement through a feedback mechanism we designed. Experimental results on two datasets demonstrate that *DeepHunt* is more effective, stable, and less reliant on labeled failure cases than prevailing deep learning-based methods.

REFERENCES

[1] Ahmed Abdulaal, Zhuanghua Liu, and Tomer Lancewicki. 2021. Practical Approach to Asynchronous Multivariate Time Series Anomaly Detection and Localization. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (Virtual Event, Singapore) *(KDD '21)*. Association for Computing Machinery, New York, NY, USA, 2485–2494. https://doi.org/10.1145/3447548.3467174

[2] Jinwon An and Sungzoon Cho. 2015. Variational autoencoder based anomaly detection using reconstruction probability. *Special lecture on IE* 2, 1 (Dec. 2015), 1–18. https://api.semanticscholar.org/CorpusID:36663713

[3] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A Zuluaga. 2020. Usad: Unsupervised anomaly detection on multivariate time series. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 3395–3404.

[4] AWS. 2021. Summary of the AWS Service Event in the Northern Virginia (US-EAST-1) Region. https://aws.amazon.com/cn/message/11201/

[5] Leo Breiman. 2001. Random forests. *Machine learning* 45 (2001), 5–32.

[6] Yiran Cheng, Bo Cheng, Pengxiang Jin, Yongqian Sun, Xiaohui Nie, Nengwen Zhao, Shenglin Zhang, and Dan Pei. 2022. Effective Attribute Selection for Multi-dimensional Root Cause Analysis. In *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 321–331.

[7] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. (2014). https://doi.org/10.48550/arXiv.1406.1078 arXiv:arXiv:1406.1078

[8] Liang Dai, Tao Lin, Chang Liu, Bo Jiang, Yanwei Liu, Zhen Xu, and Zhi-Li Zhang. 2021. SDFVAE: Static and Dynamic Factorized VAE for Anomaly Detection of Multivariate CDN KPIs. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) *(WWW '21)*. Association for Computing Machinery, New York, NY, USA, 3076–3086. https://doi.org/10.1145/3442381.3450013

[9] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 1285–1298. https://doi.org/10.1145/3133956.3134015

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative Adversarial Networks. *Commun. ACM* 63, 11 (oct 2020), 139–144. https://doi.org/10.1145/3422622

[11] Xiaofeng Guo, Xin Peng, Hanzhang Wang, Wanxue Li, Huai Jiang, Dan Ding, Tao Xie, and Liangfei Su. 2020. 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* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1387–1397. https://doi.org/10.1145/3368089.3417066

[12] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 1025–1035.

[13] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. 2017. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In *2017 IEEE International Conference on Web Services, ICWS 2017, Honolulu, HI, USA, June 25-30, 2017*, Ilkay Altintas and Shiping Chen (Eds.). IEEE, Los Alamitos, CA, 33–40.

[14] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R. Lyu. 2021. A Survey on Automated Log Analysis for Reliability Engineering. *ACM Comput. Surv.* 54, 6, Article 130 (jul 2021). https://doi.org/10.1145/3460345

[15] Chuanjia Hou, Tong Jia, Yifan Wu, Ying Li, and Jing Han. 2021. Diagnosing Performance Issues in Microservices with Heterogeneous Data Source. In *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, Los Alamitos, CA, 493–500. https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00074

[16] Tao Huang, Pengfei Chen, and Ruipeng Li. 2022. A Semi-Supervised VAE Based Active Anomaly Detection Framework in Multivariate Time Series for Online Systems. In *Proceedings of the ACM Web Conference 2022* (Virtual Event, Lyon, France) *(WWW '22)*. Association for Computing Machinery, New York, NY, USA, 1797–1806. https://doi.org/10.1145/3485447.3511984

[17] Azam Ikram, Sarthak Chakraborty, Subrata Mitra, Shiv Saini, Saurabh Bagchi, and Murat Kocaoglu. 2022. Root Cause Analysis of Failures in Microservices through Causal Discovery. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 31158–31170. https://proceedings.neurips.cc/paper_files/paper/2022/file/c9fcd02e6445c7dfbad6986abee53d0d-Paper-Conference.pdf

[18] Ashish Jaiswal, Ashwin Ramesh Babu, Mohammad Zaki Zadeh, Debapriya Banerjee, and Fillia Makedon. 2021. A Survey on Contrastive Self-Supervised Learning. *Technologies* 9, 1 (2021). https://doi.org/10.3390/technologies9010002

[19] Longlong Jing and Yingli Tian. 2021. Self-Supervised Visual Feature Learning With Deep Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43, 11 (2021), 4037–4058. https://doi.org/10.1109/TPAMI.2020.2992393

[20] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[21] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. (2016). https://doi.org/10.48550/arXiv.1609.02907 arXiv:arXiv:1609.02907

[22] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R. Lyu. 2023. Eadro: An End-to-End Troubleshooting Framework for Microservices on Multi-Source Data. In *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE '23)*. IEEE Press, New Jersey, USA, 1750–1762. https://doi.org/10.1109/ICSE48619.2023.00150

[23] Ye Li, Jian Tan, Bin Wu, Xiao He, and Feifei Li. 2023. ShapleyIQ: Influence Quantification by Shapley Values for Performance Debugging of Microservices. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*. 287–323.

[24] Zeyan Li, Junjie Chen, Rui Jiao, Nengwen Zhao, Zhijun Wang, Shuwei Zhang, Yanjun Wu, Long Jiang, Leiqin Yan, Zikai Wang, Xiaohui Nie, Kaixin Sui, and Dan Pei. 2021. Practical root cause localization for microservice systems via trace analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS) (IWQOS '21)*. IEEE, Los Alamitos, CA, 1–10. https://doi.org/10.1109/IWQOS52092.2021.9521340

[25] Zeyan Li, Wenxiao Chen, and Dan Pei. 2018. Robust and unsupervised kpi anomaly detection based on conditional variational autoencoder. In *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*. IEEE, Los Alamitos, CA, 1–9. https://doi.org/10.1109/PCCC.2018.8710885

[26] Zeyan Li, Nengwen Zhao, Mingjie Li, Xianglin Lu, Lixin Wang, Dongdong Chang, Xiaohui Nie, Li Cao, Wenchi Zhang, Kaixin Sui, Yanhua Wang, Xu Du, Guoqiang Duan, and Dan Pei. 2022. Actionable and Interpretable Fault Localization for Recurring Failures in Online Service Systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Singapore, Singapore) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 996–1008. https://doi.org/10.1145/3540250.3549092

[27] Jinjin Lin, Pengfei Chen, and Zibin Zheng. 2018. Microscope: Pinpoint Performance Issues with Causal Graphs in Micro-service Environments. In *Service-Oriented Computing*. Springer International Publishing, Cham, 3–20. https://doi.org/10.1007/978-3-030-03596-9_1

[28] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log Clustering Based Problem Identification for Online Service Systems. In *Proceedings of the 38th International Conference on Software Engineering Companion* (Austin, Texas) *(ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 102–111. https://doi.org/10.1145/2889160.2889232

[29] Dewei Liu, Chuan He, Xin Peng, Fan Lin, Chenxi Zhang, Shengfang Gong, Ziang Li, Jiayu Ou, and Zheshun Wu. 2021. MicroHECL: High-Efficient Root Cause Localization in Large-Scale Microservice Systems. In *Proceedings of the 43rd International Conference on Software Engineering: Software Engineering in Practice* (Virtual Event, Spain) *(ICSE-SEIP '21)*. IEEE Press, Los Alamitos, CA, 338–347. https://doi.org/10.1109/ICSE-SEIP52600.2021.00043

[30] Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, and Dan Pei. 2020. Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE) (ISSRE '20)*. IEEE, Los Alamitos, CA, 48–58. https://doi.org/10.1109/ISSRE5003.2020.00014

[31] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. 2023. Self-Supervised Learning: Generative or Contrastive. *IEEE Transactions on Knowledge and Data Engineering* 35, 1 (Jan. 2023), 857–876. https://doi.org/10.1109/TKDE.2021.3090866

[32] Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and Philip S. Yu. 2022. Graph Self-Supervised Learning: A Survey. *IEEE Trans. on Knowl. and Data Eng.* 35, 6 (may 2022), 5879–5900. https://doi.org/10.1109/TKDE.2022.3172903

[33] Meng Ma, Weilan Lin, Disheng Pan, and Ping Wang. 2022. Self-Adaptive Root Cause Diagnosis for Large-Scale Microservice Architecture. *IEEE Transactions on Services Computing* 15, 3 (June 2022), 1399–1410. https://doi.org/10.1109/TSC.2020.2993251

[34] Meng Ma, Jingmin Xu, Yuan Wang, Pengfei Chen, Zonghua Zhang, and Ping Wang. 2020. AutoMAP: Diagnose Your Microservice-Based Web Applications Automatically. In *Proceedings of The Web Conference 2020* (Taipei, Taiwan) *(WWW '20)*. Association for Computing Machinery, New York, NY, USA, 246–258. https://doi.org/10.1145/3366423.3380111

[35] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, Vol. 30. Atlanta, GA, 3. https://api.semanticscholar.org/CorpusID:16489696

[36] Ajay Mahimkar, Carlos Eduardo de Andrade, Rakesh Sinha, and Giritharan Rana. 2021. A Composition Framework for Change Management. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (Virtual Event, USA) *(SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 788–806. https://doi.org/10.1145/3452296.3472901

[37] Ajay Mahimkar, Zihui Ge, Jia Wang, Jennifer Yates, Yin Zhang, Joanne Emmons, Brian Huntley, and Mark Stockert. 2011. Rapid Detection of Maintenance Induced Changes in Service Performance. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies* (Tokyo, Japan) *(CoNEXT '11)*. Association for Computing Machinery, New York, NY, USA, Article 13, 12 pages. https://doi.org/10.1145/2079296.2079309

[38] Ajay Anil Mahimkar, Han Hee Song, Zihui Ge, Aman Shaikh, Jia Wang, Jennifer Yates, Yin Zhang, and Joanne Emmons. 2010. Detecting the Performance Impact of Upgrades in Large Operational Networks. *SIGCOMM Comput. Commun. Rev.* 40, 4 (aug 2010), 303–314. https://doi.org/10.1145/1851275.1851219

[39] Yuan Meng, Shenglin Zhang, Yongqian Sun, Ruru Zhang, Zhilong Hu, Yiyin Zhang, Chenyang Jia, Zhaogang Wang, and Dan Pei. 2020. Localizing Failure Root Causes in a Microservice through Causality Inference. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*. IEEE, Los Alamitos, CA, 1–10. https://doi.org/10.1109/IWQoS49365.2020.9213058

[40] Yicheng Pan, Meng Ma, Xinrui Jiang, and Ping Wang. 2021. Faster, Deeper, Easier: Crowdsourcing Diagnosis of Microservice Kernel Failure from User Space. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual, Denmark) *(ISSTA 2021)*. Association for Computing Machinery, New York, NY, USA, 646–657. https://doi.org/10.1145/3460319.3464805

[41] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. 2016. Context Encoders: Feature Learning by Inpainting. (2016). https://doi.org/10.48550/arXiv.1604.07379 arXiv:1604.07379

[42] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. (2018).

[43] Yann LeCun Raia Hadsell, Sumit Chopra. 2006. Dimensionality Reduction by Learning an Invariant Mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, Vol. 2. IEEE, Los Alamitos, CA, 1735–1742. https://doi.org/10.1109/CVPR.2006.100

[44] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1988. Learning Representations by Back-Propagating Errors. In *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, MA, USA, 696–699.

[45] Divya Saxena and Jiannong Cao. 2021. Generative Adversarial Networks (GANs): Challenges, Solutions, and Future Directions. *ACM Comput. Surv.* 54, 3, Article 63 (may 2021), 42 pages. https://doi.org/10.1145/3446374

[46] Luai A. Shalabi, Zyad Shaaban, and Basel Kasasbeh. 2006. Data Mining: A Preprocessing Engine. *Journal of Computer Science* 2, 9 (Sept. 2006), 735–739. https://doi.org/10.3844/jcssp.2006.735.739

[47] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *stat* 1050, 20 (Feb. 2017), 10–48550.

[48] Yaohui Wang, Guozheng Li, Zijian Wang, Yu Kang, Yangfan Zhou, Hongyu Zhang, Feng Gao, Jeffrey Sun, Li Yang, Pochian Lee, Zhangwei Xu, Pu Zhao, Bo Qiao, Liqun Li, Xu Zhang, and Qingwei Lin. 2021. Fast Outage Analysis of Large-Scale Production Clouds with Service Correlation Mining. In *Proceedings of the 43rd International Conference on Software Engineering* (Madrid, Spain) *(ICSE '21)*. IEEE Press, Los Alamitos, CA, 885–896. https://doi.org/10.1109/ICSE43902.2021.00085

[49] Li Wu, Johan Tordsson, Erik Elmroth, and Odej Kao. 2020. MicroRCA: Root Cause Localization of Performance Issues in Microservices. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium* (Budapest, Hungary). IEEE Press, Los Alamitos, CA, 1–9. https://doi.org/10.1109/NOMS47738.2020.9110353

[50] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, Jie Chen, Zhaogang Wang, and Honglin Qiao. 2018. Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications. In *Proceedings of the 2018 World Wide Web Conference* (Lyon, France) *(WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 187–196. https://doi.org/10.1145/3178876.3185996

[51] Tianyi Yang, Jiacheng Shen, Yuxin Su, Xiao Ling, Yongqiang Yang, and Michael R. Lyu. 2022. AID: Efficient Prediction of Aggregated Intensity of Dependency in Large-Scale Cloud Systems. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering* (Melbourne, Australia) *(ASE '21)*. IEEE Press, Los Alamitos, CA, 653–665. https://doi.org/10.1109/ASE51524.2021.9678534

[52] Guangba Yu, Pengfei Chen, Hongyang Chen, Zijie Guan, Zicheng Huang, Linxiao Jing, Tianjun Weng, Xinmeng Sun, and Xiaoyun Li. 2021. MicroRank: End-to-End Latency Issue Localization with Extended Spectrum Analysis in Microservice Environments. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) *(WWW '21)*. Association for Computing Machinery, New York, NY, USA, 3087–3098. https://doi.org/10.1145/3442381.3449905

[53] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. 2023. Nezha: Interpretable fine-grained root causes analysis for microservices on multi-modal observability data. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 553–565.

[54] Yue Yuan, Wenchang Shi, Bin Liang, and Bo Qin. 2019. An approach to cloud execution failure diagnosis based on exception logs in openstack. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, Los Alamitos, CA, 124–131.

[55] Chenxi Zhang, Xin Peng, Chaofeng Sha, Ke Zhang, Zhenqing Fu, Xiya Wu, Qingwei Lin, and Dongmei Zhang. 2022. Deeptralog: Trace-log combined microservice anomaly detection through graph-based deep learning. In *Proceedings of the 44th International Conference on Software Engineering*. 623–634.

[56] Shenglin Zhang, Pengxiang Jin, Zihan Lin, Yongqian Sun, Bicheng Zhang, Sibo Xia, Zhengdan Li, Zhenyu Zhong, Minghua Ma, Wa Jin, Dai Zhang, Zhenyu Zhu, and Dan Pei. 2023. Robust Failure Diagnosis of Microservice System through Multimodal Data. *IEEE Transactions on Services Computing* (2023), 1–14. https://doi.org/10.1109/TSC.2023.3290018

[57] Xu Zhang, Yong Xu, Si Qin, Shilin He, Bo Qiao, Ze Li, Hongyu Zhang, Xukun Li, Yingnong Dang, Qingwei Lin, Murali Chintalapati, Saravanakumar Rajmohan, and Dongmei Zhang. 2021. Onion: Identifying Incident-Indicating Logs for Cloud Systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Athens, Greece) *(ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 1253–1263. https://doi.org/10.1145/3468264.3473919

[58] Chenyu Zhao, Minghua Ma, Zhenyu Zhong, Shenglin Zhang, Zhiyuan Tan, Xiao Xiong, LuLu Yu, Jiayi Feng, Yongqian Sun, Yuzhi Zhang, et al. 2023. Robust multimodal failure detection for microservice systems. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5639–5649.

[59] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. 2019. Latent Error Prediction and Fault Localization for Microservice Applications by Learning from System Trace Logs. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 683–694. https://doi.org/10.1145/3338906.3338961

[60] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. 2018. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International conference on learning representations*. ICLR, Kigali, Rwanda. https://openreview.net/forum?id=BJJLHbb0-