Graph based Incident Extraction and Diagnosis in Large-Scale Online Systems

Zilong He, Pengfei Chen*
Sun Yat-sen University, China
hezlong@mail2.sysu.edu.cn,chenpf7@mail.sysu.edu.cn

Yu Luo, Qiuyu Yan
Tencent, China
{zekaluo,ireneyan}@tencent.com

Hongyang Chen, Guangba Yu
Sun Yat-sen University, China
{chenhy95,yugb5}@mail2.sysu.edu.cn

Fangyuan Li
Tencent, China
leiffyli@tencent.com

ABSTRACT

With the ever increasing scale and complexity of online systems, incidents are gradually becoming commonplace. Without appropriate handling, they can seriously harm the system availability. However, in large-scale online systems, these incidents are usually drowning in a slew of issues (i.e., something abnormal, while not necessarily an incident), rendering them difficult to handle. Typically, these issues will result in a cascading effect across the system, and a proper management of the incidents depends heavily on a thorough analysis of this effect. Therefore, in this paper, we propose a method to automatically analyze the cascading effect of availability issues in online systems and extract the corresponding graph based issue representations incorporating both of the issue symptoms and affected service attributes. With the extracted representations, we train and utilize a graph neural networks based model to perform incident detection. Then, for the detected incident, we leverage the PageRank algorithm with a flexible transition matrix design to locate its root cause. We evaluate our approach using real-world data collected from the WeChat® online service system, the largest instant message system in China. The results confirm the effectiveness of our approach. Moreover, our approach is successfully deployed in the company and eases the burden of operators in the face of a flood of issues and related alert signals.

CCS CONCEPTS

• **Software and its engineering** → **Maintaining software**.

KEYWORDS

Incident detection, incident diagnosis, online systems

ACM Reference Format:
Zilong He, Pengfei Chen*, Yu Luo, Qiuyu Yan, Hongyang Chen, Guangba Yu, and Fangyuan Li. 2022. Graph based Incident Extraction and Diagnosis in Large-Scale Online Systems. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3551349.3556904

ASE '22, October 10–14, 2022, Rochester, MI, USA

ACM ISBN 978-1-4503-9475-8/22/10…$15.00
https://doi.org/10.1145/3551349.3556904

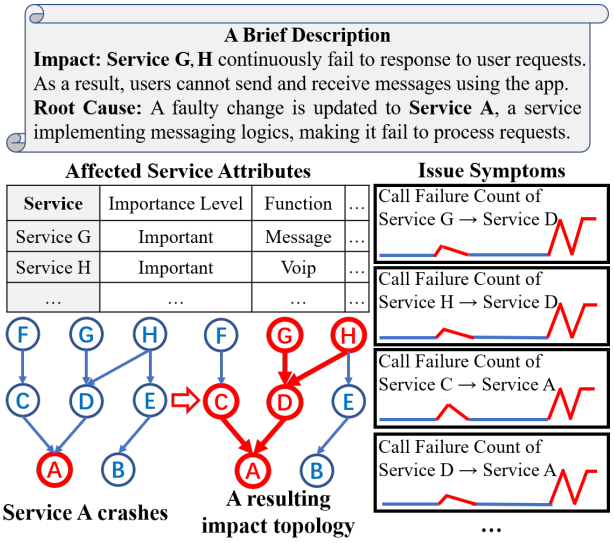


Figure 1: An issue impact topology, where red nodes denote abnormal services, and red edges denote abnormal call-relationships. In this case, a fault occurs at Service A and further cascades to Service C, D, G and H, while Service F is not seriously affected because a degrade mechanism is implemented for it for fault tolerance.

1 INTRODUCTION

Online systems are becoming increasingly popular and indispensable in modern daily life. With the rapid evolution of functionalities and the growth of users, the scale and complexity of these systems are becoming larger and larger. Taking the examined real-world system in this study as an example, the number of services is over 20K and the number of call-relationships among these services is over 80K. As a result, maintaining such a large-scale system becomes extremely difficult. Despite a quantity of effort devoted to service availability assurance, incidents are still inevitable. Without timely and appropriate management, they can quickly result in a great economic loss and a serious decrease of user experience. For example, it is estimated that Amazon.com has lost over $100 million for a single hour of downtime on Prime Day in 2018 [8]. Therefore, it is a critical task to handle an incident quickly and properly.

This study targets at managing the incidents that affect the system availability. Typically, these incidents are drowning in numerous issues. Specifically, an issue denotes something abnormal that is happening in a system, while an incident is an emergent issue. Detailed definitions of these two concepts and examples illustrating their differences will be presented in Section 2.1. Incident detection is a task to identify incidents from issues. This task is important yet challenging since various information need to be accumulated and considered together to arrive at a judgment. Generally, an issue occurring at some service will result in a cascading effect, which can be represented as an impact topology. Figure 1 presents an example of an issue and its impact topology. Various information (e.g., the affected services and the issue symptoms) can be attached to this impact topology to help incident detection and diagnosis. Extracting the impact topology of an issue is to find its affected services and call-relationships as well as collect this information. We also name this process as issue extraction.

Most of the time, issue extraction is manually performed by operators during incident diagnosis and reporting. However, such a routine usually gives rise to an overdue and incomplete analysis of the issue impact scope. As a result, some problems may arise. The first problem is that operators can be exhausted to individually examine the overwhelming amount of alerts triggered by all services in an issue impact topology. Even worse, since a full view of the ongoing issue is not presented in advance, it might be not until operators have wasted a lot of time that they eventually find all the examined alerts are actually false alerts. This problem is quite common especially in the management of a large-scale online system. Another problem is that a manual process generally results in a scrappy form of the extracted issue impact topology due to the lack of a unified standard for issue extraction. As a result, the fusion of incident-indicating information from different sources can be difficult to perform, hindering a re-utilization of the wisdom recorded in historical incident reports.

Faced with these problems, we argue that issue extraction needs to be performed automatically and as early as possible, even earlier than incident detection. Specifically, the combination of issue extraction and incident detection is denoted as *incident extraction* in this study. The extracted impact topology of an incident is treated as an atomic unit over the subsequent incident management procedures. This introduces a new paradigm of incident detection since prior data-driven approaches [14, 35, 60] usually do not perform issue extraction first and they treat the snapshot of the whole system as an atomic unit for incident detection. This paradigm poses several challenges, including: (i) How to better organize various information of ongoing issues and represent them in a comprehensive way? (ii) How to process and make full use of the obtained issue representations to perform incident detection and diagnosis?

To address these challenges, we propose a **G**raph based **I**ncident **E**xtraction and **D**iagnosis approach, namely **GIED**. To sum up, the contributions of this paper are four-fold.

- We propose a method to extract the issue impact topology at an early stage of incident management. The extracted representation takes both the issue symptoms and the affected services into account, which can help evaluate the impact of the issue more accurately.

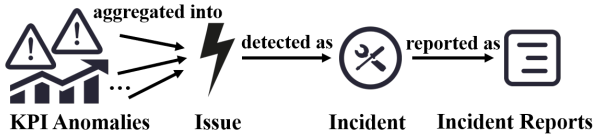


Figure 2: The relationships between some basic concepts

- To perform incident detection, we design a graph neural networks based model, which can well handle various information contained in an issue impact topology.
- We propose a flexible method to locate the root cause service of an issue based on PageRank and the extracted impact topology. With a flexible transition matrix design, the method can concurrently consider various clues about the root cause and perform root cause localization accurately.
- We conduct experiments to evaluate our approach based on real-world data collected from the WeChat® online service system, a very large-scale online system providing complex functionalities such as messaging, banking and video playing. The results confirm the effectiveness of our approach for both incident detection and root cause service localization. The proposed approach has been merged to the incident management in practice. Moreover, to facilitate reproduction, we implement an artifact on a benchmark system. The artifact is available at https://github.com/IntelligentDDS/GIED.

2 TERMINOLOGY AND RELATED WORK

We will first introduce some basic concepts about incident management for large-scale online systems in Section 2.1. Then Section 3.1-3.2 will introduce the background motivating the proposed incident extraction method, and Section 3.3 will introduce the background motivating the proposed incident diagnosis method.

2.1 Basic Concepts in Incident Management

Figure 2 presents the relationships between some basic concepts in incident management. We shall elaborate these basic concepts in the following.

Definition 2.1. A *KPI anomaly* refers to an unusual KPI (Key Performance Indicator) observation spotted by an anomaly detector.

When a KPI anomaly is detected, a corresponding alert will be triggered and sent to on-call operators, who need to examine the KPI anomaly and resolve the corresponding issue if needed.

Definition 2.2. An *issue* denotes a spatio-temporal aggregation of multiple KPI anomalies caused by the same root cause.

Definition 2.3. An *incident* is an emergent issue that is or will be affecting the user experience heavily.

Note that an issue does not necessarily imply an incident. For example, a cron job which processes data once a day can result in a sudden change in the CPU utilization of a corresponding server at fixed time; the instability of the network can cause a momentary fluctuation in the latency of some modules; an updated version of the software can alter patterns in some related KPIs even it is harmless. These issues are not considered as incidents. Generally, on-call operators need to pay more attention to incidents.

If an incident is hard to resolve in time, or it has a serious impact, a corresponding incident report should be created.

Definition 2.4. An *incident report* is a detailed description of an incident for continuous follow-up and postmortem analysis.

It should be written as clearly as possible, including a clear timeline, a detailed diagnosis for the incident, a thorough analysis of the impact scope and a subsequent optimization plan. As a result, the quantity of generated incident reports should be carefully controlled. Otherwise, an overwhelming amount of mutual related or unnecessary reports will exhaust operators and cover up real important problems. In practice, the detailed rules about whether to generate an incident report might vary across different systems.

2.2 Related Work

Issue Extraction. Recently, some researchers have found that directly analyzing enormous alert signals in a large-scale online system is exhausting, so they explore ways to ease this burden [13, 16, 41, 59]. For example, AlertSummary [59] handles alert storms through a clustering based analysis and recommends the centroids of alert clusters as representative alerts to operators. GRLIA [16] aggregates service incidents to analyze their cascading impact through graph representation learning. The objective of GRLIA is similar with that of the issue extraction stage in GIED, but GRLIA stops after the issue impact topologies have been extracted, while GIED focuses on more downstream tasks, including incident detection and diagnosis with the use of the extracted issue impact topologies.

Incident Detection. In recent years, we have witnessed a dozen of researches [10, 14, 23, 26, 35, 40, 60, 61] investigating the problem of incident detection in large-scale online systems[1]. Among them, some work [14, 35] detects incident using metrics, which are presented as time series data, while other work [10, 23, 26, 40, 60, 61] detects incident using logs, alerts, reports or user feedback, which are mainly textual data. Since the storing and processing of metrics can be more light-weight, GIED follow the metric-based work and use metrics to perform incident detection. Warden [35] is a state-of-the-art metric-based incident detection method, which first prioritizes alert signals through Weighted Mutual Information analysis and perform incident detection using a Balanced Random Forest based classifier. Compared with it, GIED takes the advantage of the extracted issue impact topologies to prioritize incident-indicating alert signals and considers relevant service attribute information, allowing more accurate incident detection for service availability assurance. However, it should still be noted that methods from different perspectives are necessary and can be applied together to ensure high system availability.

Incident Diagnosis. A great deal of effort [9, 11, 24, 29, 30, 38, 42, 45, 51–53, 55, 56, 58, 62–65] has been devoted into incident diagnosis for online systems. Among them, MonitorRank [30], MicroRCA [53], AutoMAP [45] and MicroRank [55] employ the PageRank algorithm to analyze metric data to perform root cause localization, and MicroHECL [42] locates root cause using extracted impact topologies as inputs. However, these methods are not flexible enough to take various clues into consideration when performing root cause localization, which will be discussed in details in Section 3.3. Therefore, GIED is proposed as an extension of these methods.

[1]Some previous work uses the term "incident prediction", and thus the start time of every incident needs to be precisely specified to claim some detection results as predictions. This is hard to achieve for all incidents yet, so we focus on incident detection, which is also of vital importance. However, since a prediction can be considered as an early detection, we also include the related work of incident prediction in this section.

3 BACKGROUND AND MOTIVATION

In this section, we discuss the background that motivates GIED. Section 3.1-3.2 will introduce the background motivating the proposed incident extraction method, and Section 3.3 will introduce the background motivating the proposed incident diagnosis method.

3.1 Service Call Failures as Issue Symptoms

In an online system, services need to cooperate to process user requests. The cooperation is achieved through service calls (e.g., Remote Procedure Calls). A call failure can be defined as the case that a service calls another service but it cannot receive a correct response. For example, if gRPC (google Remote Procedure Call) is used, a response whose status code is not equal to 0 can be considered as a call failure. Issues usually manifests themselves as call failures across services. For example, a software bug that crashes a service will abort a service call and increase the corresponding call failure count. Another example is the performance issues which affect the request time. Since modern online systems are usually armed with a timeout mechanism, a severe increase in request time will also result in an increase in call failures. As a result, call failure count is also known as a Key Performance Indicator (KPI) or golden metric [7]. In the following, we use the term FC to denote the call Failure Count per minute for simplicity. Though this metric is not a silver bullet yet (more discussions are detailed in Section 5.4.3), it is useful enough for the assessment of many issue symptoms, so we extract issue impact topologies with its help. Specifically, we use FC in the granularity of call-relationships, so each service caller-callee pair is associated with a corresponding FC. This metric can be reported by each service when it calls another service, and then aggregated by a monitoring agent. If services do not provide APIs to collect this information, techniques in [21, 28] might help.

3.2 Service Attribute Management

In order to manage services appropriately in a large-scale online system, operators usually sort out and store abundant service information in the Configuration Management Database (CMDB). Service attributes are an important class of service information. The service attributes such as the online status, the functionality provided, the importance level, are very helpful when operators are evaluating what is happening in the system and how urgent it is. For example, an incident affecting the banking service tends to be more urgent than one which impacts the mail service; an incident that takes place at a stable service which provides basic functions is inclined to receive more attention than one occurring at a fast-evolving service which targets at delivering new functions.

However, this information has not been fully utilized when deciding whether to trigger an alert. Typically, operators use the service attributes along with their understanding about the system to manually set alert thresholds. In this way, the alert rules are complex and there is no a unified standard to set thresholds. As a result, operators tend to set a conservative threshold in most situations,

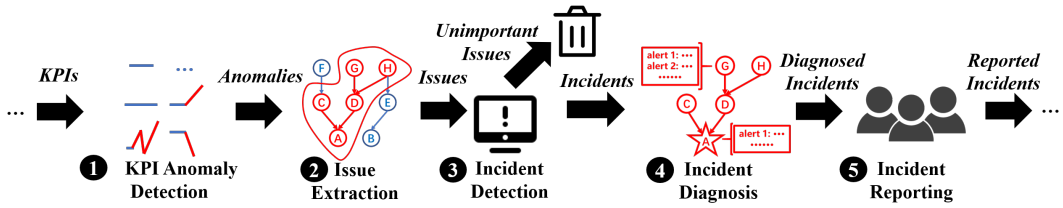


Figure 3: The overview of GIED

which might lead to a large number of false positive alerts in practice. By contrast, this study proposes to use the affected service attributes as features when training an incident detection model. To the best of our knowledge, we are the first to explicitly take the affected service attributes into consideration when applying a learning based method to cope with incident detection.

3.3 Drawbacks of Previous Root Cause Localization Methods

It is difficult to locate the root cause of an issue. Typically, operators need to find and examine clues according to their domain knowledge to complete this task. There mainly exist three representative types of clues which can be used to infer where the root cause is, i.e., (i) **Similarity-based clues:** Assuming that the root cause service is one whose metrics exhibit a similar abnormal pattern with a metric that trigger the diagnosis process (e.g., business metric or frontend sensor). (ii) **Depth-based clues:** Assuming that the root cause service is one that resides in the bottom of an issue impact topology. (iii) **Strength-based clues:** Assuming that the root cause service is one that bears the strongest alert signal.

Though these clues are reasonable to some extent and have inspired plenty of work (e.g., similarity-based clues for [30, 42, 53] and depth-based clues for [11, 30, 38, 53]), they expose some drawbacks if they are used alone. Taking similarity-based clues as an example, when an issue occurs, there may be several abnormal services that exhibit a very similar pattern due to the cascading effect. As a result, if one just assume that the root cause is the service whose metrics exhibit the most similar abnormal pattern with the abnormal frontend sensor, the localization result is very likely to be false because the examined metrics in an issue impact topology are usually similar with each other and the highest ranked one might actually be not significant enough. However, prior work either only uses one type of clues [11, 38, 42] or combines clues in a rather hard-coded way [30, 53]. They are not flexible enough to combine different clues and not extensible for new clues, resulting in a not accurate localization result. Therefore, this study introduces a mathematical formulation for clues and proposes a root cause localization method based on this formulation to flexibly combine different clues, achieving more accurate root case localization.

4 THE PROPOSED APPROACH

In this study, we propose **GIED**, a novel technique for incident extraction and diagnosis in large-scale online systems. Figure 3 presents an overall workflow of GIED. It consists of five major phases, namely ❶ KPI anomaly detection, ❷ issue extraction, ❸ incident detection, ❹ incident diagnosis and ❺ incident reporting. The last phase is conducted by operators manually with extracted information from the first four phases. In the following, we will elaborate the first four processes in detail.

4.1 KPI Anomaly Detection

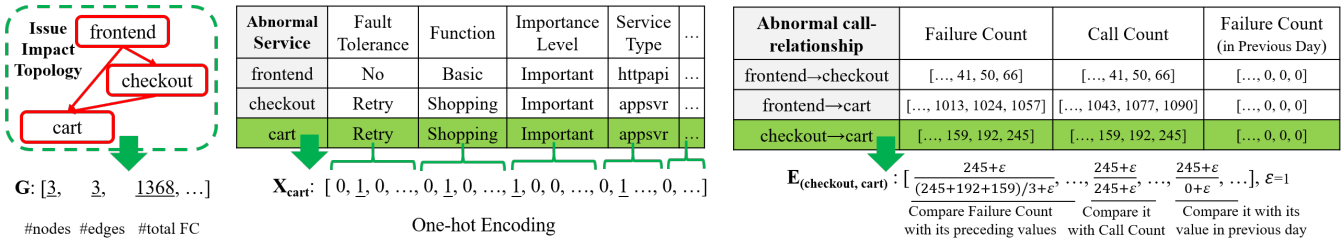
4.1.1 Problem Formulation. Abnormal call-relationships need to be picked out first in real time, acting as an entry point to other phases. As discussed in Section 3.1, FC is a useful issue symptom indicator. Therefore, GIED identifies abnormal call-relationships based on their FC. Specifically, we denote the latest window of a service caller-calee pair's FC as an FC curve and use it to perform anomaly detection. Given that the caller in a call-relationship is Service i and the callee is Service j, the FC curve of this service caller-callee pair (i, j) is formulated as a tuple $e_{i,j} = \langle (i, j), \mathbf{w}_{i,j} \rangle$, where $\mathbf{w}_{i,j} = [v_{t-l+1}, \cdots, v_{t-1}, v_t] \in \mathbb{R}^l$ contains the latest observations, with v_t as the observation at the latest time t and l as the window length. The objective of KPI anomaly detection is to find out the FC curves that the corresponding $\mathbf{w}_{i,j}$ behaves abnormally.

4.1.2 Chain Difference based Anomaly Detection. In practice, we employ a chain difference based method, a mature anomaly detector for KPIs, to perform anomaly detection on the collected FC data from all call-relationships of a system. Specifically, chain difference is defined as the difference between the current value of a KPI and its value at the same time in the previous day. If the chain difference of the FC of a call-relationship exceeds a preset threshold, an anomaly is detected. For detected abnormal call-relationships, their FC and call count in the latest time window, forming abnormal FC curves and call count curves, will be used for analysis in the future phases. One can also replace this anomaly detection algorithm with anomaly detectors from either classical anomaly detectors such as a 3-sigma algorithm [3] or novel techniques developed in recent years such as those mentioned in [32, 47, 48, 54, 57].

4.2 Issue Extraction

4.2.1 Problem Formulation. Given that a set of abnormal FC curves $\widehat{\mathcal{E}} = \{e_{i,j}, e_{p,q}, \cdots\}$ have been detected, the objective of issue extraction is to divide them into several disjoint subsets, each of which is caused by the same issue. The FC curves in each subset constitute an issue impact topology $T = \langle \mathcal{V}, \mathcal{E} \rangle$, with $\mathcal{V}$ representing a set of nodes, each corresponding to an affected service, and $\mathcal{E} \subseteq \widehat{\mathcal{E}}$ representing a set of edges over $\mathcal{V}$, each corresponding to an abnormal call-relationship. Finally, the output of issue extraction is a set of issue impact topology $\mathcal{T} = \{T_0, T_1, \cdots\}$.

4.2.2 Clustering Based Issue Impact Topologies Extraction. There might be multiple ongoing issues in a large-scale online system. If



(a) The issue impact topology and its global level features (b) An example of the node features presenting the affected service attribute information of the issue (c) An example of the edge features presenting the issue symptom information, which is constructed using the latest windows of KPIs

Figure 4: An example of feature engineering to the issue impact topology of a fault injected to the simulation environment

the detected FC anomalies are caused by different issues, we need to separate these anomalies to accurately assess the impact scope of each issue. Inspired by [43, 53], we assume that if two abnormal FC curves satisfy the following two conditions, they are caused by the same root cause: (i) they have similar abnormal temporal patterns (e.g., curve shapes) over a sliding window ending at the current time; (ii) they are in one of the connected components in a graph composed of the detected abnormal call-relationships.

To meet the condition (i), we first perform shape-based clustering on the FC curves $\widehat{\mathcal{E}}$ to separate call-relationships with different abnormal patterns. The shape-based distance between each two abnormal FC curves, e.g., $e_{i,j}$ and $e_{p,q}$, can be calculated as follows,

$$\mathbf{Dist}(e_{i,j}, e_{p,q}) = \begin{cases} 1 & \text{if } |\rho(\mathbf{w}_{i,j}, \mathbf{w}_{p,q})| >= 0.9 \\ 2 & \text{otherwise} \end{cases}. \quad (1)$$

Specifically, if the absolute Pearson Correlation $|\rho(w_{i,j}, w_{p,q})|$ between two FC curves is higher than 0.9, we consider them as similar in shape and assign them a smaller distance value as shown in equation (1). Based on this distance measure, we apply DBSCAN algorithm [20] to group the FC curves into multiple clusters. DBSCAN is selected here because it does not require the number of clusters to be specified in advance, which is more suitable in our scenario. Then, to meet the condition (ii), connected components are assembled from each cluster based on the call-relationships and form issue impact topologies. Through the above steps, each extracted issue impact topologies are connected graph with all call-relationships exhibiting similar abnormal patterns. In practice, we can filter out impact topologies with few edges (e.g., ≤ 3) to strike a balance between efficiency and accuracy. This is because there might be frequent fluctuations in a real-world online system, which usually affect few services and are mostly transient. Since such a fault does not cascade to many services and only a few call-relationship behave abnormally for a while, excluding the corresponding issues from the successive phases can greatly alleviate the analysis overhead.

4.3 Incident Detection

4.3.1 Problem Formulation. Given an issue impact topology $T = \langle \mathcal{V}, \mathcal{E} \rangle$, the objective of incident detection is to determine whether T implies an incident. This is accomplished through two processes, namely an offline training process and an online inference process. In the offline training process, we use historical data to train a function that takes T as an input and then calculates an output $y \in \{0, 1\}$, which indicates whether an incident is detected. This function can be parameterized with a couple of trainable model parameters $\boldsymbol{\Theta}$. Then in the online inference process, we use the trained function to detect incidents from the extracted issues $\mathcal{T}$.

4.3.2 Preparation of Training Data. To train an incident detection model, we require training data with labels. Historical issue impact topologies can be indexed with their diagnosed root cause services (more details in Section 4.4) and their occurrence time, so labelled positive samples (impact topologies of incidents) can be acquired and extracted based upon the occurrence time and the root cause service analyzed in historical incident reports. In addition, new issue impact topologies with alerts can be labelled by operators. By default, if an issue lasts for a period of time and results in a series of extracted impact topologies in successive timestamps, we only keep the one that appears earliest to slim the dataset in practice. If an impact topology is not associated with an incident report and not labeled by operators as an incident, it will be treated as a negative sample (i.e., an unimportant issue).

4.3.3 Feature Engineering. The seriousness of an issue can be assessed mainly from two perspectives, namely the affected service attributes and the issue symptoms. Since each service affected by an issue can be considered as a node in the issue impact topology, we denote the affected service attributes as the node features $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times n}$ with n as its feature dimension. Node features consist of the importance level (e.g., important, ordinary), the function (e.g., messaging, banking), the online status (e.g., online, offline), the service type (e.g., proxy service, key-value cache, database) and the configured fault tolerance strategy (e.g., different degrade or retry policies) of a service. These features can reveal the location of an issue and reflects whether it is emergent. All the selected node features are categorical features, so we process them using one-hot encoding before feeding them to the model. As for the issue symptoms, we construct features using KPIs of each abnormal call-relationship in the issue impact topology. Since each call-relationship can be represented as an edge in the issue impact topology, the constructed features are denoted as edge features $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times s}$, with s as its feature dimension, including a binning of FC values, a division of FC by the corresponding call count per minute and a division of current FC by its historical values. These features can describe the degree of abnormality in the KPI values. Another type of features that can describe the issue symptoms is the graph-level features of the issue impact topology, which can be denoted as global features

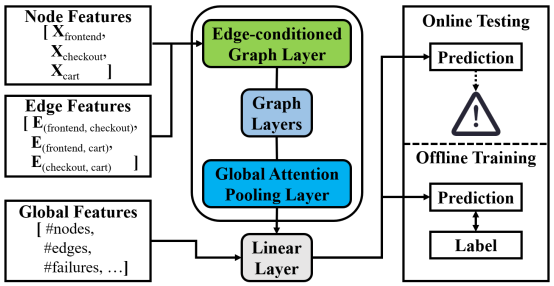


Figure 5: A demonstration of the graph based model

$\mathbf{G} \in \mathbb{R}^m$, with m as its feature dimension. Global features consist of the topology size, the important service count and the total FC count of an issue. The constructed global features and edge features are numerical features and will be processed with a logarithmic transformation to avoid a negative effect from the long-tailed feature distribution and a Z-score normalization to ensure no specific dimension will dominate the statistics. An example illustrating the feature engineering process is in Figure 4.

4.3.4 Offline Model Training. We use a graph neural networks based model to learn to detect incidents using the extracted features.

Why do we choose Graph Neural Networks? To support an end-to-end learning, especially considering the complex interactions between nodes and edges as well as the uncertain dimension of the input (i.e., $|\mathcal{V}|$ and $|\mathcal{E}|$), we design the graph neural networks based model. Vanilla machine learning models cannot handle this complexity and thus may not achieve a good efficacy.

As in Figure 5, this model includes an edge-conditioned graph layer [49] to aggregate both node features and edge features, a couple of optional graph layers [50] to further process the calculated hidden representations, a global attention pooling layer [36] to calculate a summarized representation of the whole impact topology and a linear layer to obtain the output based upon the summarized representation and global features.

Suppose the node features of nodes i, j are represented as $\mathbf{X}_i$, $\mathbf{X}_j \in \mathbb{R}^n$, respectively, and the edge features between them are represented as $\mathbf{E}_{i,j} \in \mathbb{R}^s$, the feed-forward process of edge-conditioned graph layer [49] to calculate node i's hidden representation $\mathbf{X}'_i \in \mathbb{R}^d$ with d representing the hidden dimension, is as follows:

$$\mathbf{X}'_i = \mathbf{X}_i \mathbf{\Theta}_1 + \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \mathbf{X}_j h_{\mathbf{\Theta}_2}(\mathbf{E}_{i,j}), \tag{2}$$

where $\mathcal{N}(i)$ denotes node i's neighborhood derived from $\mathcal{V}$ and $\mathcal{E}$, $\mathbf{\Theta}_1 \in \mathbb{R}^{n \times d}$ refers to a trainable weight, and $h_{\mathbf{\Theta}_2} : \mathbb{R}^s \to \mathbb{R}^{n \times d}$ refers to a trainable filter which first applies a trainable weight $\mathbf{\Theta}_2 \in \mathbb{R}^{s \times u}$, with $u = n \times d$, to edge features $\mathbf{E}_{i,j}$ and then reshape[2] the result $\mathbf{E}_{i,j}\mathbf{\Theta}_2 \in \mathbb{R}^u$ into a matrix of the form $h_{\mathbf{\Theta}_2}(\mathbf{E}_{i,j}) \in \mathbb{R}^{n \times d}$.

With the edge-conditioned graph layer, we obtain a synthesized representation for each service considering both of its service information and KPI information. Afterwards, optional graph layers such as that in [50] can also be added to construct a deeper network to produce a more comprehensive representation for each service.

[2] An example of reshape: https://pytorch.org/docs/stable/generated/torch.reshape.html.

When the final hidden representation of each service is determined, the next step is to calculate a summarized representation S for the whole impact topology through a pooling operation on hidden representations of all the nodes. Suppose the calculated representation $\mathbf{X}'_i$ represents the final hidden representation of node i, a global attention pooling layer [36] is formulated as:

$$\mathbf{S} = \sum_{i \in \mathcal{V}} \text{softmax}\left(\mathbf{X}'_i \mathbf{\Theta}_3\right) \odot \mathbf{X}'_i, \tag{3}$$

where $\mathbf{\Theta}_3 \in \mathbb{R}^{d \times 1}$ is a trainable weight to assign a score for each node, softmax$(\cdot)$ is applied over node scores to decide which nodes are relevant to the task and $\odot$ denotes element-wise multiplication.

The calculated graph-level representation $\mathbf{S} \in \mathbb{R}^d$ is concatenated with the global features $\mathbf{G} \in \mathbb{R}^m$ and fed to a linear layer with parameters $\mathbf{\Theta}_4 \in \mathbb{R}^{(d+m) \times 1}$, producing the incident score $\hat{y}$:

$$\hat{y} = \text{concat}\left(\mathbf{S}, \mathbf{G}\right) \mathbf{\Theta}_4. \tag{4}$$

During the offline training phase, we adopt the cross-entropy as the loss function and Adam [31] as the optimizer to train the parameters including $\mathbf{\Theta}_1, \mathbf{\Theta}_2, \mathbf{\Theta}_3, \mathbf{\Theta}_4$ and possibly the parameters of the optional graph layers with the use of historical data. With y representing the label for the impact topology and $\sigma(\cdot)$ representing a sigmoid operation, the loss function is defined as:

$$L(y, \hat{y}) = -y \log\left(\sigma(\hat{y})\right) - (1 - y) \log\left(1 - \sigma(\hat{y})\right) \tag{5}$$

4.3.5 F1-score-sensitive Threshold Selection. Only a minority of impact topologies might imply an incident. To better handle the data imbalance problem, we do not directly apply the value 0 as the threshold to classify the samples. Instead, we select a threshold τ which can make detection results maximize the F1-score in training data. F1-score is a metric describing the effectiveness of a detection model. We will explain this metric more detailedly in Section 5.1.2.

4.3.6 Online Incident Detection. During the online inference phase, samples with an incident score $\hat{y}$ higher than τ will be detected as an incident and corresponding alerts will be sent to on-call operators.

4.4 Incident Diagnosis

4.4.1 Problem Formulation. Given an incident whose impact topology is $T = \langle \mathcal{V}, \mathcal{E} \rangle$, the objective of incident diagnosis is to locate its root cause service r, which is the culprit of the incident.

4.4.2 PageRank Algorithm for Root Cause Localization. Typically, when an operator is diagnosing an issue, he/she will investigate along the service topology and try to find out clues to approach the root cause service. This process can be simulated using PageRank algorithm [46], with the PageRank vector $\vec{v}$ representing the root cause scores of the services. The transition probability matrix $\mathbf{P}$ can be calculated with the use of pre-configured clues.

Definition 4.1. A *clue* is a function from (src, des) to p, where (src, des) represents a service pair with source src and destination des, and $p \in \mathbb{R}^+$ is calculated with an expression based on (src, des), whose result is correlated with the possibility that the destination des is close to the root cause.

For example, Table 1 shows clues configured in this study. Specifically, $(\cdot)^+$ denotes $max(\cdot, 0)$. The purpose of applying these clues

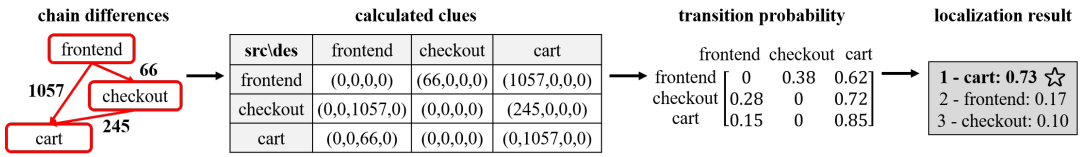


Figure 6: An example of the diagnosis process on the issue impact topology of a fault injected to the simulation environment

Table 1: Clues configured in this study, where $D_{i,j}$ represents the FC chain difference for the caller-callee pair (i, j), $\mathcal{B}$ stands for services in the bottom of an impact topology

Clue	Expression	Domain
$clue_0(i,j)$	$(D_{i,j})^+$	$\{(i,j)\|e_{i,j} \in \mathcal{E}\}$
$clue_1(i,i)$	$(\max_z D_{z,i} - \max_z D_{i,z})^+$	$\{(i,i)\|i \in \mathcal{V}\}$
$clue_2(j,i)$	$(\sum_z D_{i,z} - D_{i,j})^+$	$\{(i,j)\|e_{i,j} \in \mathcal{E}\}$
$clue_3(j,i)$	$(\max_z D_{z,i} - D_{i,j})^+$	$\{(i,j)\|e_{i,j} \in \mathcal{E} \wedge j \in \mathcal{B}\}$

is to make a service with a stronger alert signal more likely to be identified as the root cause. Here, $clue_0$ allows a callee with a more abnormal FC to be visited more often, and $clue_1$ allows the localization process stuck in a more abnormal service, and $clue_2$ and $clue_3$ allow a service's abnormal siblings to be visited more often. Such clues can be designed based on domain knowledge or observations from historical incident cases.

When multiple clues are considered, we can preprocess them with a logarithmic transformation to avoid the negative effect brought by the long-tailed distribution of some clues. Besides, a Z-score normalization can be applied across all the clues to ensure no specific clues will dominate the statistics, if needed.

Suppose the k^{th} clue is defined as $clue_k$ and its expression for a service pair (i, j) is represented as $clue_k(i, j)$, the transition probability matrix $\mathbf{P}$ can be formulated as:

$$\mathbf{A}_{i,j} = \begin{cases} \max_k(\alpha_k clue_k(i,j)) & \text{if } \exists k, (i,j) \in \text{dom}(clue_k) \\ 0 & \text{otherwise} \end{cases}, \quad (6)$$

$$\mathbf{P}_{i,j} = \frac{\mathbf{A}_{i,j}}{\sum_j \mathbf{A}_{i,j}}. \quad (7)$$

Here, $\max_k(\cdot)$ is a maximization operation applied over different clues, α_k stands for a strength parameter for $clue_k$, and $\text{dom}(clue_k)$ represents the domain of $clue_k$. Each α_k can be set based upon domain knowledge, or based on grid search according to the localization performance in historical incidents. Taking the service pair (frontend, checkout) in Figure 6 as an example, since it is also a call-relationship, it is in the domain of $clue_0$, and thus we can use the corresponding expression to calculate $clue_0$(frontend, checkout). However, it is not in the domain of $clue_1$, $clue_2$ and $clue_3$, so finally $\mathbf{A}_{\text{frontend,checkout}} = \alpha_0 clue_0$(frontend, checkout).

With the transition probability matrix $\mathbf{P}$ defined above, the PageRank equation can be formulated as:

$$\vec{v} = (1-c)\mathbf{P}\vec{v} + c\vec{u}, \quad (8)$$

where c denotes the damping parameter and $\vec{u}$ denotes the additional teleportation vector. By default, $\vec{u} = [\frac{1}{|\mathcal{V}|}, \frac{1}{|\mathcal{V}|}, \cdots, \frac{1}{|\mathcal{V}|}]^T$.

The root cause service localization result r is then calculated as:

$$r = \arg\max_i \vec{v}_i. \quad (9)$$

Table 2: Statistics of real-world datasets for RQ1~3

Sets	#Cases	#Incidents	#Test Cases	#Test Incidents
A	3222	297	1335	100
B	4174	372	1809	75
C	5168	453	1827	81

5 EVALUATION

In this section, we carry out experiments to evaluate our approach. We aim at answering the following research questions:

- **RQ1:** Can the extracted graph based issue representations help incident detection?
- **RQ2:** How effective is the graph neural networks based model for incident detection?
- **RQ3:** How useful is each type of features in the extracted graph based issue representations for incident detection?
- **RQ4:** How effective is the root cause service localization method for incident diagnosis?
- **RQ5:** Can the proposed approach help early incident management in industrial practice?

5.1 Experiment Setting

5.1.1 Datasets. We use both datasets from the real-world production environment and the simulation environment to evaluate GIED.

Datasets from the real-world production environment are collected from the company. As described in Section 1, this system is large-scale, managing thousands of services and serving billions of users. Monitoring metric data for call-relationships in this system can be over 500TB per day. We extract issues in real time based upon these monitoring data and store them in a database for analysis. The datasets for the incident detection evaluation (RQ1~3) are collected **for about 3 months**, and divided into 3 parts, namely A, B, and C, according to the time. Each dataset contains **over 3K** issues which have occurred in the system. For each of the datasets, the last 15-day period is used for testing, with the rest for training. Statistics of these datasets are displayed in Table 2. Then, among the incidents, **77** incidents containing detailed diagnosis information in corresponding incident reports are selected for the incident diagnosis evaluation (RQ4). To avoid the data leakage problem, these 77 cases are gathered after the design and tuning of the proposed root cause localization method have been completed.

The dataset from the simulation environment is collected from a benchmark system Online Boutique [6] consisting of 11 services (10 services are from the original benchmark, and we implement an additional database service for the benchmark to simulate real-world systems). We deploy it on a Kubernetes cluster with 13 nodes and collect monitoring data for 12 days. Specifically, to mimic the real-world workload pattern, we replace its load generator to one that is driven by the real-world user logs. Besides, we equip a timeout mechanism for services in this benchmark using

Istio [5]. Then, two types of issues are considered, including: (i) **Availability issues:** we implement multiple faulty versions for each service which can abort its processing for some types of requests, and randomly select one service and update it with one of its faulty versions. (ii) **Performance issues:** We use chaosblade [4] to randomly inject performance issues such as CPU exhaustion, network delay, database delay into an arbitrary service instance. For incident detection, since this task is closely related to the real user experience and a simulation benchmark cannot reproduce this complexity, this simulation dataset is not used for the comparison among different incident detection methods. Instead, we only define some simple rules to label some issues as incidents, and then we use this dataset as a showpiece of how GIED works in the open-sourced artifact. For incident diagnosis, a total of 200 availability issues and 202 performance issues are considered. This simulation dataset is available in the artifact.

5.1.2 Evaluation Metrics. For RQ1~2, we use **Precision, Recall, and F1-score** to evaluate the effectiveness of the proposed incident detection method. Precision measures the percentage of incidents that are correctly detected over all alerts triggered by a method, while Recall measures the percentage of incidents that are correctly detected over all actual incidents. F1-score is the harmonic mean of Precision and Recall, weighting them equally.

For RQ3, we use the **Precision-Recall (PR) curve** and the area under it (also known as **average precision**, **AP**) to evaluate the usefulness of node, edge and global features. The PR curve can give an informative picture of the performance of the detection results, and it is well suited for an imbalanced dataset [19].

For RQ4, we use **Accuracy** (also denoted as **HR@1** in some literature [42]) to evaluate the root cause localization method. Accuracy denotes the proportion of the correct localization results (generated as in equation (9)) to the total number of cases examined.

For RQ5, we will evaluate the difference between the time that an incident topology is detected by GIED and the alert time and report time described in a corresponding incident report. For simplicity, we name this difference as the lead time in the following.

5.1.3 Hyper-parameter Setting. The length of the latest sliding window of FC curves for analysis is set as 1 hour for real-world datasets and 10 minutes for the simulation dataset. The minimal number of FC curves in an issue impact topology for real-world datasets is set to 3 and that for the simulation dataset is set to 1 (issue impact topologies with a size smaller than this minimal number are filtered out). Then for other hyper-parameters, the setting for the simulation dataset is the same as that for the real-world datasets, which are detailed in the following. For issue extraction, the maximum distance between two FC curves for one to be considered as in the neighborhood of the other in DBSCAN [20] is set to 1.5. The issue extraction method with these hyper-parameters has run on the company for more than 1 years. As for incident detection and diagnosis, the hyper-parameters are set with a cross-validation approach. As for the graph based incident detection model in GIED, the hidden dimension of the edge-conditioned graph layer (equation (2)) is set to 20. A graph attention layer [50] is used as the middle optional graph layer and its hidden dimension is set to a value equal to the dimension of the global features. The model is trained with a learning rate of 0.001 and 60 epochs. For the root cause localization algorithm, different clues are preprocessed using a logarithmic transformation. Then $\alpha_0, \alpha_1, \alpha_2, \alpha_3$ are set to 1, 1, 0.3, 0.3, respectively, and the damping parameter c is set to 0.15.

Table 3: The comparison of GIED with some state-of-the-art methods based on raw alert signal information

Datasets	A			B			C		
Methods	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
AirAlert	0.56	0.24	0.34	0.38	0.24	0.30	0.38	0.16	0.23
Warden*	0.36	0.21	0.26	0.26	0.2	0.23	0.21	0.10	0.13
GIED	**0.85**	**0.86**	**0.85**	**0.87**	**0.87**	**0.87**	**0.81**	**0.88**	**0.85**

5.2 Evaluation Results

5.2.1 ***RQ1:*** *Can the extracted graph based issue representations help incident detection?* To validate the incident detection efficacy of the extracted graph based issue representations, we compare GIED with some state-of-the-art methods which detect incident using alert signal information, i.e., (i) **AirAlert** [14]: A method based on Bayesian network analysis [17] and XGBoost [12]. It detects incidents using raw alert signal information. We implement it based on our alert signals and use its full mode to detect incidents. (ii) **Warden** [35]: A method based on Weighted Mutual Information analysis, feature extraction and Balanced Random Forest [33]. Some features needed in Warden are not persistently stored for analysis, e.g., detailed engineer activities, so we only evaluate Warden using raw alert signal information here. Specifically, we denote the version of Warden without the process of its feature extraction as **Warden***. Then, for each issue impact topology, we construct an input for the above baseline methods by setting the values of non-alerting signals outside an issue impact topology as 0. GIED uses the extracted graph based issue representations for incident detection.

Table 3 shows the experimental results. We can observe that the extracted graph based issue representations by GIED can benefit incident detection to a great extent. This is because there are too enormous alert signals in a large-scale online system, rendering the learning of a map from these signals to incidents extremely difficult. Previous methods [14, 35] usually cope with this problem with a selection of incident-indicating alert signals in advance. One of their assumptions is that there exist incident-indicating alert signals that work well for most incidents and thus need to be picked out first. However, since incidents can happen anywhere, the incident-indicating alert signals for one specific incident are not bound to be incident-indicating alert signals that work well for most incidents. If such signals are filtered out after the alert signal selection phase, the incident detection performance will suffer a lot. As for GIED, the issue extraction phase can be regarded as a selection of alert signals too. This selection is conducted on demand in the light of the extracted impact topologies, and each extracted impact topology is a group of selected alert signals. After this phase, the number of concerned alert signals can be effectively reduced and only alert signals that are closely related to some ongoing issues in the system are preserved. In a word, the graph based issue representations enhance incident detection to a great extent.

5.2.2 ***RQ2:*** *How effective is the graph neural networks based model for incident detection?* To answer this research question, we compare our approach with some baseline models implemented with

Table 4: The comparison of different models for incident detection based on extracted impact topologies.

Datasets	A			B			C		
Methods	P	R	F1	P	R	F1	P	R	F1
LR	0.79	0.61	0.69	0.83	0.71	0.76	0.74	0.80	0.77
KNN	0.76	0.28	0.41	0.59	0.44	0.50	0.71	0.49	0.58
RF	0.87	0.51	0.64	0.85	0.56	0.68	0.77	0.76	0.77
GBDT	0.90	0.55	0.68	0.89	0.67	0.76	0.76	0.79	0.78
Warden**	0.52	0.91	0.66	0.55	0.93	0.69	0.53	0.96	0.68
GIED	0.85	0.86	**0.85**	0.87	0.87	**0.87**	**0.81**	0.88	**0.85**

conventional classification algorithms including Logistic Regression (**LR** [27]), K Nearest Neighborhood classifier (**KNN** [18]), Random Forest (**RF** [37]) and Gradient Boosting Decision Tree (**GBDT** [22]). These classification algorithms have been employed as part of the model or for comparison in some related researches [14, 39, 60]. We also evaluate the model used in Warden [35] on our extracted features and denote it as **Warden****. For the baseline models, a summation aggregator is utilized to get a representation of the set of node features (i.e., $\sum_{i \in \mathcal{V}} X_i$) and edge features (i.e., $\sum_{e_{i,j} \in \mathcal{E}} E_{i,j}$). Then both of the aggregated representations and the global features are normalized and then act as inputs for the baseline models.

Table 4 presents the evaluation results of GIED and the baseline models. In our scenario, a high Recall with a low Precision is unacceptable because on-call operators will tend to underestimate or even neglect the alerted incident if the Precision of a detection method is low, and a low Recall with a high Precision is unacceptable too because if too many incidents might be missed, the detection method is actually of no use. Therefore, we attach the most importance to F1-score since it comprehensively cover evaluations from Precision and Recall. The results show that GIED outperforms all the baseline models in F1-score. Besides, the performance of GIED is stable under different datasets. This shows that performing an end-to-end learning with the designed graph neural networks based model is useful for improving the performance in incident detection. The baseline models lose insight into the interactions between services and their call-relationships, accounting for their suboptimal performance compared with GIED.

5.2.3 ***RQ3:*** *How useful is each type of features in the extracted graph based issue representations for incident detection?* To answer this question, we employ the LR model as the base model (because it is the most similar model to us, and it is the runner-up model on the evaluations in Table 4) to assess the usefulness of different features. Each type of features will be fed to the LR model individually, and the resulting cases are named **"Only Node"**, **"Only Edge"** and **"Only Global"**, respectively. Besides, we test the performance when different features are simply concatenated together as the model input, and the resulting case is named **"Direct Aggr"**.

Figure 7 presents the results. We can see that if each type of the features is used alone, the performance drops compared with those achieved by all features ("Direct Aggr" and GIED). This indicates that all types of features play an important role in incident detection. Besides, we can observe that only using node features ("Only Node") results in a very low performance, and in two datasets (A and C), using all features through a simple concatenation ("Direct Aggr") only shows a slight improvement over that using only edge features ("Only Edge"). These two observations further demonstrate the necessity of applying graph neural networks to better

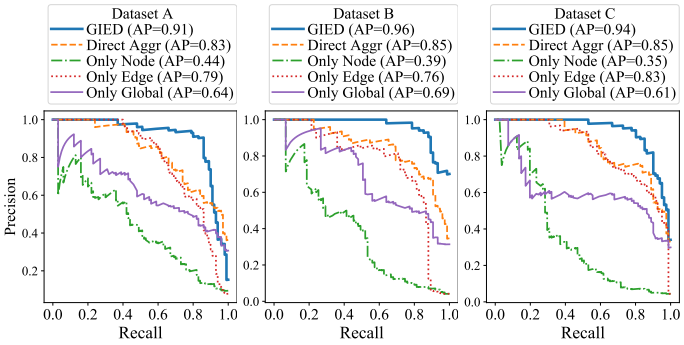


Figure 7: The PR curves for the efficacy of different features

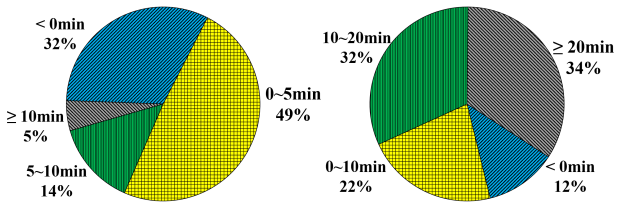
Table 5: The Accuracy comparison of root cause localization

Datasets	SL	SD	MicroHECL	MonitorRank*	GIED
Production	0.88	0.86	0.55	0.90	**0.99**
Sim-Avail	0.99	0.93	0.73	0.78	0.99
Sim-Perf	0.62	0.89	0.53	0.74	0.88
Sim-Total	0.81	0.91	0.63	0.76	**0.94**

incorporate both node and edge features. This is because the affected service attributes are of little use if the corresponding issue symptom information is not well attached and considered together.

5.2.4 ***RQ4:*** *How effective is the root cause service localization method for incident diagnosis?* To answer this research question, we conduct experiments to validate the accuracy of the root cause service localization method in GIED. Four methods are compared, which are: (i) **Simple Largest (SL)**: The callee in a call-relationship with the largest FC in the issue impact topology is selected as the root cause service localization result. (ii) **Simple Deepest (SD)**: The deepest service in the issue impact topology is selected as the root cause service localization result (if there is more than one deepest service, we select the one that whose FC is the largest). (iii) **MicroHECL**: A root cause localization method proposed in [42]. (iv) **MonitorRank**: A root cause localization method proposed in [30]. Specifically, because some KPIs of the whole service topology (including thousands of services) have been lost in many historical cases in practice, we use MonitorRank on the extracted impact topology and name it as the modified version **MonitorRank***.

Table 5 shows the experimental results on both the production dataset and the simulation dataset. The simulation dataset is further divided into two parts according to the injected fault types, namely **"Sim-Avail"** for availability issues and **"Sim-Perf"** for performance issues. From the results on these datasets, we find that if a corresponding impact topology can be extracted, the root cause localization at a service granularity is not a very tricky problem because even simple strawman strategies can achieve a good result (both of SL and SD achieve a good accuracy on both datasets). MicroHECL achieves the worst result in the comparison (note that its accuracy here, i.e., 0.55 and 0.63, complies with the HR@1 result presented in its original paper [42]). Overall, GIED outperforms all these methods. Specifically, GIED achieves a competitive result on each type of injected faults in the simulation dataset and achieves a super result in the production dataset. This demonstrates that the designed clues for root cause service localization are effective, and



(a) Compared with the alert time (b) Compared with the report time

Figure 8: The distribution of the lead time of GIED

our formulation of the transition probability matrix helps improve the root cause localization performance.

5.2.5 ***RQ5:*** *Can the proposed approach help early incident management in industrial practice?* To answer this research question, we first introduce two concepts, namely the alert time and the report time of an incident. The **alert time** is defined as the time that the earliest manually configured alert is triggered when an incident occurs, and the **report time** is defined as the time that an operator decides to begin an incident report. We carefully record the alert time and the report time from each historical incident report in one month if a corresponding issue impact topology can be extracted. The records will be compared with the time that GIED detects the corresponding issue as an incident, and then the lead time (defined in Section 5.1.2) of GIED can be calculated.

Figure 8 shows the distribution of the recorded lead time of GIED. In **68%** of the recorded cases, GIED detected the incidents earlier than the manually configured alerts. In these cases, GIED has an average lead time of **4 minutes** compared to the alert time. Besides, in **88%** of the recorded cases, GIED detected the incidents earlier than the time that an operator reported the corresponding incident. In these cases, GIED has an average lead time of **61 minutes** compared to the report time. In the comparison with the alert time, though the superiority of GIED is not significant, it should be noted that this superiority is meaningful. Because many alerts might not be noticed by operators in time since they are not assigned a high priority due to their high false alert rate. For GIED, benefited from its Precision declared in Section 5.2.2, we can set its corresponding alerts with a high priority, ensuring a relatively quick action for the incident. In general, these results show that GIED can help early incident detection and report in large-scale online systems.

5.3 Case Studies

In the following, we present some real cases. Some confidential details are anonymized.

Real Case I: This is an incident caused by an overload to a database service. At the beginning of this incident, an offline proxy service suddenly load a great volume of new data into a database, rendering its memory allocation time much longer. As a result, the overloaded database service fails to process online requests, and many important services, e.g., services for video playing, are affected. A simplified version of the extracted impact topology of this incident is presented in Figure 9. With its help, the root cause is diagnosed as the database service. Moreover, the system is finally recovered by stopping the data loading operations.

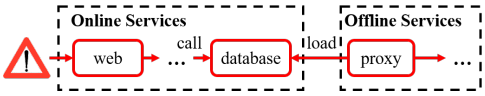


Figure 9: A simplified issue impact topology for Real Case I

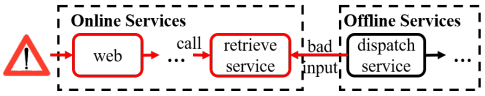


Figure 10: A simplified issue impact topology for Real Case II

Table 6: The execution time of the dominated part in issue extraction when different numbers of abnormal call-relationships are detected in a minute

#Anomalies	2.5K	5K	10K	20K	40K	80K
Avg Time	0.17s	0.56s	2.22s	8.51s	34.30s	3.29min

Table 7: The execution time of offline model training and online incident detection and diagnosis

Procedure	Offline Training	Per-Sample Detection	Per-Sample Localization
Avg Time	1.17min	0.29ms	6.44ms

Real Case II: This is an incident caused by a program bug. The error handling logic of a service (i.e., retrieve service in Figure 10) is not correctly implemented. As a result, when it receives a bad input with incomplete content, a memory out of bound error happens and the service finally crashes. This incident seriously affects the recommendation function of the system. Figure 10 presents a simplified version of the extracted impact topology of this incident. After operators examined the issue impact topology and identified the root cause of this incident, they rollbacked the input to the root cause service and restarted it to recover the system.

5.4 Discussions

5.4.1 Mitigation of Anomaly Flood. GIED can help operators filter out irrelevant anomalies and concentrate on the real important incidents. Specifically, at the incident detection phase, about 95% of issues are filtered out. Moreover, at the incident diagnosis stage, the search space of root causes is narrowed down by **96%**, since the average size of an incident impact topology in our system is 26 while only one of them is recognized as the root cause. With this mitigation, operators can focus on the key information of incidents and diagnose an incident more quickly.

5.4.2 Time Efficiency of GIED. GIED is very efficient and particularly suitable for large-scale online systems. Table 6 and 7 present the execution time of GIED on a server with 32 Intel(R) Xeon(R) E5-2667 v4 CPUs and a 251GB RAM. For issue extraction, since this procedure is executed in real time and its execution time depends on the number of detected abnormal FC curves, we simulate inputs with different numbers of detected abnormal FC curves to evaluate its efficiency under different stresses. It can be seen that even when a half of the system behaves anomalously (the real-world system contains a total of around 80K call-relationships and 40K of them are detected as anomalies), the execution time of issue extraction is just around half a minute. Most of the time, the execution time of issue extraction is much shorter when there is a smaller number of abnormal call-relationships detected. For incident detection and

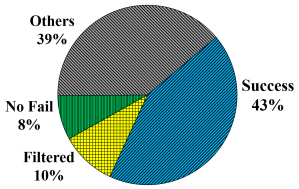


Figure 11: The frequencies of different situations

diagnosis, since after issue extraction, only relevant information are left for analysis, the execution is very efficient. For example, the offline training on each real-world dataset can be completed in less than 2 minutes. Besides, the per-sample detection and localization can be completed in a millisecond level.

5.4.3 Limitation of GIED. Limitations of GIED mainly lie in two aspects. The first limitation is that due to some configurations of the approach or the target online system, GIED may fail in extracting impact topologies for some issues based on call failures. Cases influenced by this limitation can be divided into two categories, namely **Filtered** and **No Fail**, which will be detailed in the following. The other limitation is, as introduced in Section 1, GIED targets at availability issues / incidents of online systems, so it cannot handle issues that do not manifest themselves in the backend availability monitoring, such as some functionality issues. Cases influenced by this limitation are referred to as **Others** in the following. Overall, there are four categories of situations, i.e., (i) **Success:** The impact topology of the incident is successfully extracted. (ii) **Filtered:** The incident only affects FC of one caller-callee pair, so since we filter out impact topologies with only one edge as described in Section 4.2, it results in no extracted impact topology. (iii) **No Fail:** The incident does not give rise to call failures though it does affect some services according to other metrics. For example, if the timeout mechanism is not set properly, a performance issue might not manifest itself in call failures. (iv) **Others:** The incident does not affect any of the currently collected data for the availability monitoring. For example, a bug may not always manifest itself at backend, which is also named as a Silent Backend Issue in [61].

Figure 11 shows the frequency of each situation calculated according to all recorded incident reports in one month. It can be seen that nearly a half of the incident reports are associated with an extracted impact topology. For the other situations, we may take the measures listed below. For **Filtered**, usually the involved services are near the frontend or are extremely important since though only two services are affected, the case is still considered as an incident. We can pay special attention to these services and set specific alert rules for them. Actually, since these services are usually important, most of them have already been armed with specific alert rules designed for them at the very beginning. For **No Fail**, we will develop a more comprehensive impact topology extraction algorithm which can consider and incorporate more types of metrics in the future. For **Others**, incident detection based on user feedback can be employed, which are discussed in Section 2.2.

5.4.4 Generalizability of GIED. Incident management is a crucial problem for all online systems. Though different companies may apply different specific software technologies, their incident management processes [1, 2, 8, 15, 35, 44] are usually similar to the process that we describe in Section 2.1. Therefore, the techniques proposed in GIED can be merged into their incident management practice too. Specifically, the incident detection method in GIED applies a data-driven framework, which is widely employed and has been validated in some related research [14, 35, 60]. We believe such a framework is applicable and effective in industrial practice. As for the incident diagnosis method proposed in GIED, it is designed to be a highly extensible framework. One can easily implement clues configured in this study to validate our method, or further optimize it on their systems by exploring more clues if needed.

6 THREATS TO VALIDITY

The *internal threat to validity* mainly lies in the implementation of our approach and the baseline approaches. We implement the baseline approaches [14, 30, 35, 42] by ourselves since they have no publicly available implementations. To alleviate this threat, we have followed their papers, implemented them based upon matured machine learning toolbox [12, 25, 34] and carefully checked the implementations. Besides, some baselines are modified versions of the original work because not all their requirements can be fulfilled in our system. This is normal in the research of incident management for modern online systems. To alleviate this threat, we have tried to implement approximated versions for them.

The *external threat to validity* mainly lies in the target system. Though we have applied GIED in the real production system, the system may not represent all online systems in other companies. We discussed the generalizability of GIED in Section 5.4.4. Besides, to facilitate the promotion of GIED to other systems, we have implemented an artifact based on an open-sourced benchmark system and thus in the future we can test GIED in other systems.

7 CONCLUSION AND FUTURE WORK

We propose GIED, a novel approach to perform incident detection and diagnosis based upon extracted issue impact topologies, so as to ensure the availability of online systems. We make use of the DBSCAN algorithm as well as the service call-relationships to extract issue impact topologies from online systems. Then a graph neural networks based model is leveraged to perform incident detection and the PageRank algorithm is employed to perform root cause service localization. We have evaluated GIED using both real-world datasets and the simulation dataset, and have successfully merged it to the incident management of a real production system. The experimental results and real deployment of GIED confirm its effectiveness. In the future, we plan to consider more information to provide a more effective incident extraction and diagnosis method.

ACKNOWLEDGMENTS

The research is supported by the National Natural Science Foundation of China (No. U1811462), the Natural Science Foundation of Guangdong Province (No. 2019A1515012229), the Key-Area Research and Development Program of Guangdong Province (No. 2020B010165002), and the Basic and Applied Basic Research of Guangzhou (No. 202002030328), the Wechat Rhino-Bird Joint Research Program (No. JR-WXG-2021621). The corresponding author is Pengfei Chen.

REFERENCES

[1] 2021. *Computer Security Incident Handling Guide.* https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf

[2] 2021. *Google Cloud: Data incident response process.* https://services.google.com/fh/files/misc/data_incident_response_2018.pdf

[3] 2022. *3-sigma rule.* https://en.wikipedia.org/wiki/68-95-99.7_rule

[4] 2022. ChaosBlade. https://github.com/chaosblade-io/chaosblade

[5] 2022. Istio. https://istio.io/

[6] 2022. Online Boutique: A Cloud-Native Microservices Demo Application. https://github.com/GoogleCloudPlatform/microservices-demo

[7] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. 2016. *Site reliability engineering: How Google runs production systems.* " O'Reilly Media, Inc.".

[8] Junjie Chen, Xiaoting He, Qingwei Lin, Yong Xu, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. An empirical investigation of incident triage for online service systems. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP).* IEEE, 111–120.

[9] Junjie Chen, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. Continuous Incident Triage for Large-Scale Online Service Systems. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019.* IEEE, 364–375.

[10] Junjie Chen, Shu Zhang, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Yu Kang, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2020. How Incidental are the Incidents? Characterizing and Prioritizing Incidents for Large-Scale Online Service Systems. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020.* IEEE, 373–384.

[11] Pengfei Chen, Yong Qi, and Di Hou. 2019. CauseInfer: Automated End-to-End Performance Diagnosis with Hierarchical Causality Graph in Cloud Environment. *IEEE Trans. Serv. Comput.* 12, 2 (2019), 214–230.

[12] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016.* ACM, 785–794.

[13] Yujun Chen, Xian Yang, Hang Dong, Xiaoting He, Hongyu Zhang, Qingwei Lin, Junjie Chen, Pu Zhao, Yu Kang, Feng Gao, Zhangwei Xu, and Dongmei Zhang. 2020. Identifying linked incidents in large-scale online service systems. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020.* ACM, 304–314.

[14] Yujun Chen, Xian Yang, Qingwei Lin, Hongyu Zhang, Feng Gao, Zhangwei Xu, Yingnong Dang, Dongmei Zhang, Hang Dong, Yong Xu, et al. 2019. Outage prediction and diagnosis for cloud service systems. In *The World Wide Web Conference.* 2659–2665.

[15] Zhuangbin Chen, Yu Kang, Liqun Li, Xu Zhang, Hongyu Zhang, Hui Xu, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, Yingnong Dang, Feng Gao, Pu Zhao, Bo Qiao, Qingwei Lin, Dongmei Zhang, and Michael R. Lyu. 2020. Towards intelligent incident management: why we need it and how we make it. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020.* ACM, 1487–1497.

[16] Zhuangbin Chen, Jinyang Liu, Yuxin Su, Hongyu Zhang, Xuemin Wen, Xiao Ling, Yongqiang Yang, and Michael R. Lyu. 2021. Graph-based Incident Aggregation for Large-Scale Online Service Systems. In *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021.* IEEE, 430–442.

[17] Diego Colombo, Marloes H Maathuis, Markus Kalisch, and Thomas S Richardson. 2012. Learning high-dimensional directed acyclic graphs with latent and selection variables. *The Annals of Statistics* (2012), 294–321.

[18] Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory* 13, 1 (1967), 21–27.

[19] Jesse Davis and Mark Goadrich. 2006. The relationship between Precision-Recall and ROC curves. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006 (ACM International Conference Proceeding Series, Vol. 148),* William W. Cohen and Andrew W. Moore (Eds.). ACM, 233–240.

[20] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *Kdd*, Vol. 96. 226–231.

[21] Rodrigo Fonseca, George Porter, Randy H Katz, and Scott Shenker. 2007. X-trace: A pervasive network tracing framework. In *4th {USENIX} Symposium on Networked Systems Design & Implementation ({NSDI} 07).*

[22] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[23] Jiazhen Gu, Chuan Luo, Si Qin, Bo Qiao, Qingwei Lin, Hongyu Zhang, Ze Li, Yingnong Dang, Shaowei Cai, Wei Wu, Yangfan Zhou, Murali Chintalapati, and Dongmei Zhang. 2020. Efficient incident identification from multi-dimensional issue reports via meta-heuristic search. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020.* ACM, 292–303.

[24] Xiaofeng Guo, Xin Peng, Hanzhang Wang, Wanxue Li, Huai Jiang, Dan Ding, Tao Xie, and Liangfei Su. [n.d.].

[25] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX.* Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[26] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R. Lyu, and Dongmei Zhang. 2018. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018.* ACM, 60–70.

[27] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. 2013. *Applied logistic regression.* Vol. 398. John Wiley & Sons.

[28] Jinho Hwang, Guyue Liu, Sai Zeng, Frederick Y Wu, and Timothy Wood. 2014. Topology discovery and service classification for distributed-aware clouds. In *2014 IEEE International Conference on Cloud Engineering.* IEEE, 385–390.

[29] Jiajun Jiang, Weihai Lu, Junjie Chen, Qingwei Lin, Pu Zhao, Yu Kang, Hongyu Zhang, Yingfei Xiong, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2020. How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020.* ACM, 1410–1420.

[30] Myunghwan Kim, Roshan Sumbaly, and Sam Shah. 2013. Root cause detection in a service-oriented architecture. In *ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '13, Pittsburgh, PA, USA, June 17-21, 2013.* ACM, 93–104.

[31] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.*

[32] Nikolay Laptev, Saeed Amizadeh, and Ian Flint. 2015. Generic and Scalable Framework for Automated Time-series Anomaly Detection. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 1939–1947.

[33] Guillaume Lemaitre, Fernando Nogueira, and Christos K. Aridas. 2017. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. *J. Mach. Learn. Res.* 18 (2017), 17:1–17:5.

[34] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. 2017. Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. *Journal of Machine Learning Research* 18, 17 (2017), 1–5. http://jmlr.org/papers/v18/16-365.html

[35] Liqun Li, Xu Zhang, Xin Zhao, Hongyu Zhang, Yu Kang, Pu Zhao, Bo Qiao, Shilin He, Pochian Lee, Jeffrey Sun, Feng Gao, Li Yang, Qingwei Lin, Saravanakumar Rajmohan, Zhangwei Xu, and Dongmei Zhang. 2021. Fighting the Fog of War: Automated Incident Detection for Cloud Systems. In *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021.* USENIX Association, 131–146.

[36] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. 2016. Gated Graph Sequence Neural Networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings.*

[37] Andy Liaw, Matthew Wiener, et al. 2002. Classification and regression by randomForest. *R news* 2, 3 (2002), 18–22.

[38] JinJin Lin, Pengfei Chen, and Zibin Zheng. 2018. Microscope: Pinpoint Performance Issues with Causal Graphs in Micro-service Environments. In *Service-Oriented Computing - 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11236).* Springer, 3–20.

[39] Qingwei Lin, Ken Hsieh, Yingnong Dang, Hongyu Zhang, Kaixin Sui, Yong Xu, Jian-Guang Lou, Chenggang Li, Youjiang Wu, Randolph Yao, Murali Chintalapati, and Dongmei Zhang. 2018. Predicting Node failure in cloud service systems. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018.* ACM, 480–490.

[40] Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, and Dongmei Zhang. 2016. iDice: problem identification for emerging issues. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016,* Laura K. Dillon, Willem Visser, and Laurie A. Williams (Eds.). ACM, 214–224.

[41] 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, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume.* ACM, 102–111.

[42] 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 *43rd IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2021, Madrid, Spain, May 25-28, 2021*. IEEE, 338–347.

[43] Ping Liu, Yu Chen, Xiaohui Nie, Jing Zhu, Shenglin Zhang, Kaixin Sui, Ming Zhang, and Dan Pei. 2019. FluxRank: A Widely-Deployable Framework to Automatically Localizing Root Cause Machines for Software Service Failure Mitigation. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 35–46.

[44] Jian-Guang Lou, Qingwei Lin, Rui Ding, Qiang Fu, Dongmei Zhang, and Tao Xie. 2013. Software analytics for incident management of online services: An experience report. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*. IEEE, 475–485.

[45] Meng Ma, Jingmin Xu, Yuan Wang, Pengfei Chen, Zonghua Zhang, and Ping Wang. 2020. AutoMAP: Diagnose Your Microservice-based Web Applications Automatically. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*. ACM / IW3C2, 246–258. https://doi.org/10.1145/3366423.3380111

[46] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[47] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. 2019. Time-Series Anomaly Detection Service at Microsoft. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 3009–3017.

[48] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouet. 2017. Anomaly detection in streams with extreme value theory. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1067–1075.

[49] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 29–38.

[50] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *Proceedings of the International Conference on Learning Representations*.

[51] Hanzhang Wang, Zhengkai Wu, Huai Jiang, Yichao Huang, Jiamu Wang, Selçuk Köprü, and Tao Xie. 2021. Groot: An Event-graph-based Approach for Root Cause Analysis in Industrial Settings. In *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 419–429.

[52] 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 *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 885–896.

[53] Li Wu, Johan Tordsson, Erik Elmroth, and Odej Kao. 2020. MicroRCA: Root Cause Localization of Performance Issues in Microservices. In *NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020*. IEEE, 1–9. https://doi.org/10.1109/NOMS47738.2020.9110353

[54] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. 2018. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, 187–196.

[55] 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 *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*. ACM / IW3C2, 3087–3098.

[56] Xu Zhang, Chao Du, Yifan Li, Yong Xu, Hongyu Zhang, Si Qin, Ze Li, Qingwei Lin, Yingnong Dang, Andrew Zhou, Saravanakumar Rajmohan, and Dongmei Zhang. 2021. HALO: Hierarchy-aware Fault Localization for Cloud Systems. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*. ACM, 3948–3958.

[57] Xu Zhang, Qingwei Lin, Yong Xu, Si Qin, Hongyu Zhang, Bo Qiao, Yingnong Dang, Xinsheng Yang, Qian Cheng, Murali Chintalapati, Youjiang Wu, Ken Hsieh, Kaixin Sui, Xin Meng, Yaohai Xu, Wenchi Zhang, Furao Shen, and Dongmei Zhang. [n.d.].

[58] 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 *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*. ACM, 1253–1263.

[59] Nengwen Zhao, Junjie Chen, Xiao Peng, Honglin Wang, Xinya Wu, Yuanzong Zhang, Zikai Chen, Xiangzhong Zheng, Xiaohui Nie, Gang Wang, Yong Wu, Fang Zhou, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Understanding and handling alert storm for online service systems. In *ICSE-SEIP 2020: 42nd International Conference on Software Engineering, Software Engineering in Practice, Seoul, South Korea, 27 June - 19 July, 2020*. ACM, 162–171.

[60] Nengwen Zhao, Junjie Chen, Zhou Wang, Xiao Peng, Gang Wang, Yong Wu, Fang Zhou, Zhen Feng, Xiaohui Nie, Wenchi Zhang, et al. 2020. Real-Time Incident Prediction for Online Service Systems. In *The 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Sacramento, California, United States,November 8-13, 2020*. ACM.

[61] Wujie Zheng, Haochuan Lu, Yangfan Zhou, Jianming Liang, Haibing Zheng, and Yuetang Deng. 2019. iFeedback: Exploiting User Feedback for Real-Time Issue Detection in Large-Scale Online Service Systems. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 352–363. https://doi.org/10.1109/ASE.2019.00041

[62] Zibin Zheng, Tom Chao Zhou, Michael R Lyu, and Irwin King. 2011. Component ranking for fault-tolerant cloud applications. *IEEE Transactions on Services Computing* 5, 4 (2011), 540–550.

[63] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. 2021. Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study. *IEEE Trans. Software Eng.* 47, 2 (2021), 243–260.

[64] 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 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*. ACM, 683–694.

[65] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Wenhai Li, Chao Ji, and Dan Ding. 2018. Delta debugging microservice systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*. ACM, 802–807.