2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid) | 979-8-3503-9566-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/CCGrid59990.2024.00018

Causality Enhanced Graph Representation Learning for Alert-Based Root Cause Analysis

Zhaoyang Yu[1], Qianyu Ouyang[1], Changhua Pei[2*], Xin Wang[3], Wenxiao Chen[4],
Liangfei Su[5], Huai Jiang[5], Xuanrun Wang[6], Jianhui Li[2], Dan Pei[1]
[1]Tsinghua University; BNRist, {yu-zy20, oyqy19}@mails.tsinghua.edu.cn, peidan@tsinghua.edu.cn
[2]Computer Network Information Center, Chinese Academy of Sciences, {chpei, lijh}@cnic.cn
[3]Stony Brook University, x.wang@stonybrook.edu
[4]Huawei, chenwenxiao3@huawei.com
[5]eBay Inc, {liasu, huajiang}@ebay.com
[6]Lingjun Investment, xuanrun_wang@163.com

***Abstract*—Accurate and efficient root cause identification in online service systems is critical for service stability and user experience. When a system failure occurs, numerous alerts are generated, but existing methods fail to effectively integrate all these multi-modal data to pinpoint the root causes. Moreover, most existing approaches are inefficient for large-scale online services due to their high reliance on handcrafted rules and domain expertise. This paper introduces *AlertRCA*, an algorithm for Root Cause Analysis (RCA) based on Alert events. It utilizes a pre-trained Alert2Vec module to encode multi-modal alert information into vectors, and implements an RCA-oriented causality prediction graph attention network (CPGAT) to automatically gauge causal relationships between alerts. Further, we devise a novel dispersing and aggregating graph neural network (DAGNN) to identify root causes. Experiments on a real-world dataset collected from a top-tier e-commerce company reveal *AlertRCA*'s superior performance, achieving 83.9% top-1 and 96.8% top-3 accuracy on average. Our codes are available at https://github.com/NetManAIOps/AlertRCA.**

I. Introduction

The seamless functioning of large-scale online service systems is essential to people's daily lives. Due to their intricate architecture, these systems are susceptible to failures at any component level, potentially inducing severe business impacts [1]–[4]. Hence, swiftly and accurately identifying the root causes of such disruptions and restoring system functionality is of paramount importance.

System recovery after a failure, typically overseen by Site Reliability Engineers (SREs), generally involves a three-step process: anomaly detection [5]–[7], Root Cause Analysis (RCA) [8], and system remediation. Anomaly detection utilizes monitoring data such as business and performance metrics [9]–[13], logs [14]–[17], or a combination thereof [18], [19] to detect if there exist anomalous alerts and aggregates them into a multi-modal alert. RCA then leverages the alert information to perform domain- and system-specific analysis to pinpoint and verify the root cause of a failure, whether software glitches or hardware issues due to configuration changes or malfunctions. With accurate RCA, SREs can promptly restore the system to its normal state. Focusing on the work in the RCA phase, the aim of this work is to generate a ranked list of potential root causes to be addressed in the subsequent failure remediation phase.

Despite the numerous RCA methods proposed, they all adhere to the same workflow. Initially, they create a dependency graph between various RCA entities, including metrics [12], services [20], [21], and alert events. Subsequently, different models such as GCN [22], GNN [23], Random Walk [24], and DFS are applied to the constructed graph. Nonetheless, there are two primary fundamental issues that existing methods do not adequately address:

- *Obtaining a practical and accurate graph that demonstrates the causal relationship among entities related to the failure, referred to as a Causal graph in this paper.* This is a critical yet challenging task. Although ideal graphs can be extracted from traces as shown in previous studies [20], traces are often unavailable in many online services due to their high requirements for computational and storage resources. Some methods, such as Groot [21], attempt to manually construct the causal graph with the help of domain experts. However, this approach is neither scalable nor practical when service upgrades are frequent and the number of graph nodes increases rapidly. Certain causal inference methods, relying on metrics [12] or traces [20], are unsuitable for our multi-modal scenario.
- *A deep learning-based method specifically designed for alert-based RCA scenarios is absent*: The considerable learnable parameters and data-driven mechanism of deep learning-based methods enable them to learn from historical events and exhibit superior performance compared to rule-based methods such as PageRank and DFS. Nevertheless, there is no method explicitly tailored for alert-based RCA, significantly limiting their theoretical performance.

To address the above challenges, we propose ***AlertRCA***, a **R**oot **C**ause **A**nalysis algorithm based on **Alert** events. As illustrated in Figure 1, an alert contains a large number of extensible key-value pairs to describe suspicious events such as the time, type, semantic interpretation, status code, etc. A pre-trained Alert2Vec module is designed in *AlertRCA* to extract the semantic vector for the downstream tasks. In order to circumvent the need for the manual construction of a causal

*Changhua Pei is the corresponding author.

979-8-3503-9566-2/24/$31.00 ©2024 IEEE
DOI 10.1109/CCGrid59990.2024.00018

graph, we first leverage an automated approach to generate a complete graph connecting all events, referred to as the Alert Dependency Graph (ADG) in this paper. Subsequently, we employ an RCA-oriented Causality Prediction Graphical Attention Network (CPGAT) proposed to determine the causality scores for each link within the ADG. Links with a causality score of zero are then removed from the graph.


```
"ServiceClientErrorSpike": {
   "startsAt": 1593958909000,
   "endsAt": 1593959209000,
   "pool": "r1pgwpmtsvc",
   "colo": null,
   "severity": 143,
   "status": "504",
   "type": "Error",
   "typename": "PayPal_GATEWAY_TIMEOUT",
   "indication": "paypal",
   "target": "paypal gateway"
}
```


Fig. 1. Example of the *Alert Event*.

Utilizing the causal graph learned by CPGAT, we introduce a novel Dispersing and Aggregating Graph Neural Network (DAGNN) to identify the root causes. It consists of two main steps: 1) Using a dispersing and aggregating mechanism to measure the root cause contribution along with the failure propagating direction, and 2) Identifying the true root causes from plenty of suspicious ones with a self-residual structure, taking into account both the alert events' own information and self-loop dependencies. By combining the above components together, *AlertRCA* is learned in a supervised manner based on two real-world datasets, containing respectively 782 business domain failures and 170 service-based failures from a top-tier software company over 15 months (Jan 2020 $\sim$ Apr 2021). It is worth mentioning that such datasets are prevalent in the majority of online service companies. This can be attributed to the fact that historical accidents may lead to substantial financial damages. Therefore, each accident is thoroughly examined, pinpointed, eradicated, and documented. The evaluation of these two datasets shows that *AlertRCA* reaches 77.4% top-1 and 96.8% top-3 accuracy on the service-based dataset and 85.3% top-1 and 96.7% top-3 accuracy on the business domain dataset.

The contributions of this paper are summarized as follows:

- We propose ***AlertRCA***, a **R**oot **C**ause **A**nalysis algorithm based on **Alert** events. AlertRCA is practical and precise due to several factors: 1) It utilizes widely recognized alert events as input, rather than challenging-to-obtain trace information, which may be difficult for smaller companies to access; 2) It avoids the dependence on manually created rules that denote causal relationships between events; 3) It has been specifically designed for RCA scenarios.
- We design a novel Dispersing and Aggregate Graph Neural Network (DAGNN) that extends beyond traditional graph models by incorporating not only neighboring nodes but also fault propagation directions and contributions from nodes. Furthermore, DAGNN is equipped with a Causality Prediction Graphical Attention Network (CPGAT) we propose to automatically learn causal links between alert events through an asymmetric attention mechanism.
- Comprehensive evaluations are conducted on two real-world datasets. The results show that *AlertRCA* outperforms other baselines by reaching 83.9% top-1 accuracy and 96.8% top-3 accuracy among a long candidate list on average.
- Besides the real-world datasets, we also evaluate our method on an open-source dataset. Compared with the state-of-the-art method [20], we achieve 24.8% and 15.7% improvements on top-1 accuracy. Our codes are available at https://github.com/NetManAIOps/AlertRCA.

II. Preliminary

A. Alert Event

To continually monitor the health status of a multifaceted online system comprising diverse services, each service employs several anomaly detection mechanisms, including time-series anomaly detection and log anomaly detection. Heterogeneous data are constantly checked, and semi-structured events are reported if there are anomalies. We call such a kind of event **Alert Event** (also written as alert for brevity).

As shown in Fig. 1, an alert contains a set of property-value pairs based on configured templates, which are different depending on the locations where the events are generated. Information in an alert contains various attributes (see Fig. 1), including event name (ServiceClientErrorSpike), service name (paypal gateway), start time, end time, status code, anomaly severity, critical logs, etc. The values of attributes are also of many types, including number, string, boolean, etc. It is important to note that our algorithm can process any semi-structured datasets if they contain the following essential elements: WHEN (the timing of events), WHAT (a brief description of events), and WHERE (the location of events). Utilizing alert event-like data as input for Root Cause Analysis (RCA) algorithms offers three primary advantages:

- *Multi-modal compatibility:* Alert events are not limited to representing anomalies identified by anomaly detection algorithms. They can also record service changes such as code deployment and system rebooting with minimal effort.
- *Comprehensibility:* As demonstrated in Fig. 1, alert events contain sufficient semantic information, making them easily understandable for system maintainers.
- *Resource efficiency:* In comparison to raw monitored data like logs and time-series data, alert events extract only the most valuable information for RCA. This is particularly beneficial for computing, storage, and bandwidth efficiency in online services with hundreds of thousands of sub-modules.

B. Service Dependency Graph

In Fig. 1, it is demonstrated that each event encompasses the service in which it occurs. By integrating this with service topology data, typically found in the system Configuration Management Database (CMDB), a **Service Dependency Graph (SDG)** can be developed. An example of a potential

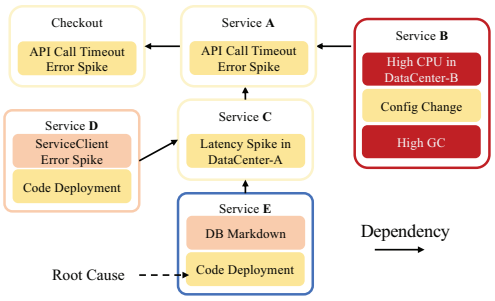


Fig. 2. Illustration of service dependency graph, where an alert in Service E is the ground-truth root cause.

SDG for a checkout system is depicted in Fig. 2. The construction of an SDG is carried out as described below.

When a failure occurs, the construction of the SDG starts with an initial event and the corresponding service. By continuously looking for services that report events and have dependency relationships with the current services in the SDG, the SDG will eventually include all possible failure-related services, their dependencies, and the events reported on them. In practice, RCA algorithms like PageRank [25] and RandomWalk [24] can be utilized in SDG with features extracted from events on services.

Nonetheless, the potential existence of numerous sub-modules and the possibility of multiple alerts occurring within a single service make it insufficient to merely locate the root cause services. This is because it fails to provide specific information necessary for the subsequent failure remediation process. Furthermore, conducting RCA solely at the service-level could potentially lead to inaccurate localization outcomes. Taking a real-world failure (Fig. 2) as an example, the root cause of the failure is the code deployment event on *service-E*. But, *service-B* will be identified as the most likely root cause if only service-level information is considered because there are two alerts on *service-B* and the severity of them is higher.

III. Methodology

In this section, we present the comprehensive design of our AlertRCA, with its workflow illustrated in Fig. 3. Before delving into the specifics, we need to clarify three essential notations:

- **SDG**: Short for **service dependency graph**, an example of which can be seen in Fig. 2. The graph's nodes consist of services, and edges exist only between services, not between alert events.
- **ADG**: Short for **alert dependency graph**, an example is depicted in the second rectangle of Fig. 3. The graph's nodes consist of alert events and edges represent the relationship between alert events. It is important to note that in ADG, the edge between events is constructed in a dummy mode. This means that if there is an edge between Service 5 (S5) and Service 3 (S3) (represented by black arrows), all events belonging to S5 will be connected to all events belonging to S3. Moreover, there are no weights for edges in ADG.
- **ACG**: Short for **alert causal graph**, an example is depicted in the third rectangle of Fig. 3. The distinction between the ACG and ADG is that in the ACG, all weights for edges between events are automatically determined by our algorithm (details in Section III-D), and edges with a weight of 0 will be removed.

A. Overview

The workflow of *AlertRCA* is illustrated in Fig. 3. *AlertRCA* takes an SDG as input to locate the root cause in four steps: ADG construction, Alert2Vec, Causality Prediction Graphical Attention Network (CPGAT) and Dispersing and Aggregating Graph Neural Network (DAGNN).

Specifically speaking, first, to deal with multi-modal data in alerts, we need to construct an ADG from the original SDG. Then, Alert2Vec encodes alerts to vectors expressing semantic information, called **alert vector**, based on a pre-trained natural language model (Section III-C). Third, to help distinguish between real and fake causal links in the ADG constructed from the SDG, we design CPGAT to assign each causal link in the ADG a **causality score** (Section III-D, Section III-E). Finally, a graph neural network tailored for root cause analysis, DAGNN, is used to locate the root cause based on the ADG with alert vectors and causality scores (Section III-F).

B. Construction of Alert Dependency Graph (ADG)

Constructing a completely accurate causal graph without resorting to the domain knowledge of experts is extremely challenging. We devise **Alert Dependency Graph, (ADG)** (Fig. 4), a type of intermediate graph between SDG and ACG. As mentioned earlier, there exist causal links between all pairs of alerts in ADG constructed from SDG, except those whose services have no dependencies in SDG. For all alerts within the same service, causal links are established between each other in the ADG. It is noteworthy that the edges in ADG do not represent the final causal relationship and may be completely unrelated.

ADG has two important properties: (1) constructing ADG requires only SDG and no additional domain expertise; (2) the final causal graph we want should be a sub-graph of the corresponding ADG. Thus *AlertRCA* locates the root cause only with ADG as input and requires no manual rule.

C. Alert2Vec

The objective of the Alert2Vec phase is to obtain the representation vectors associated with alert events. Intuitively, we propose Alert2Vec to imitate how SREs understand the alerts. When SREs check the failure and relevant alerts in an online service system, they read each attribute in the alerts and then form a general impression.

As shown in Fig. 5, Alert2Vec has two stages, generating a semantic vector for each alert attribute and compressing all semantic vectors into one **alert vector** for each alert. In the first stage, Alert2Vec uses a pre-trained language model (LM)

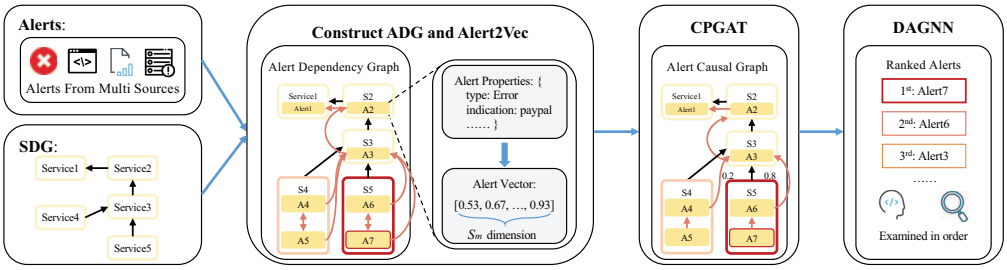


Fig. 3. The workflow of *AlertRCA*. S1 and A1 are abbreviations for Service 1 and Alert 1, respectively.

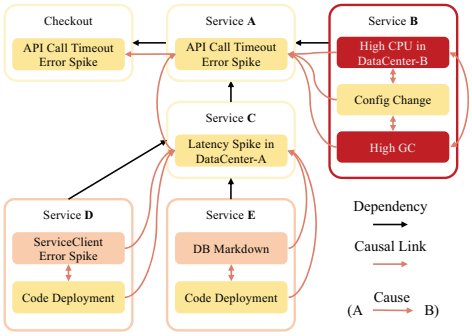


Fig. 4. An example of Alert Dependency Graph

to understand the semantic meaning of each attribute. Here, we choose BERT [26], a widely used open-source large language model that can be much easier deployed in a company than a GPT [27] model. For each attribute k and the corresponding value v, the key-value sentence is constructed as "k is v". For example in the fifth row of the example in Fig. 1, *i.e.,* the attribute "severity", the key-value sentence is constructed as "severity is 143". Then, we feed these key-value sentences to the BERT to obtain the semantic vectors of all attributes.

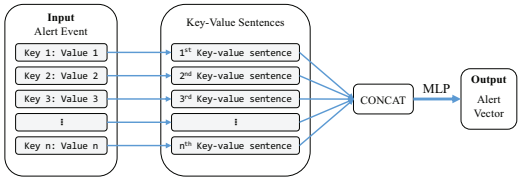


Fig. 5. Workflow of the Alert2Vec module.

The default size of the semantic vector obtained from LM is very large for downstream computing. Therefore, in the second stage, we use a shallow Multi-Layer Perception (MLP) to compress the semantic vectors in an alert. We concatenate all semantic vectors in one alert to compress with MLP and output a domain-specific feature vector called **alert vector** with a much smaller size S_m. In this paper, we empirically set S_m as 100.

The LM helps in two ways: extracting vectors from multi-modal data to provide multi-modal compatibility and improving the generality of RCA methods taking advantage of the knowledge from the natural language domain. This is beneficial in the early stages of a new online service system when sufficient training data have not been collected yet. It is noteworthy that all the parameters of BERT is fixed during the training of our *AlertRCA* model.

D. Causality Prediction Graphical Attention Network

The objective of the Causality Prediction Graphical Attention Network (CPGAT) is to autonomously determine a causality score for each edge within the input ADG, assessing the likelihood of a causal relationship between the two events connected in ADG. We then combine ADG and causality scores to generate a refined graph which is called **Alert Casual Graph (ACG)**.

We design CPGAT with the consideration of three observations that SREs will pay attention to when determining whether one alert is caused by another. Intuitively, when SREs check whether a potential causal link $A \rightarrow B$ (*i.e.,* alert A causes alert B) actually exists, they will read the alert information of A and B and then consider the following three observations:

- **O1**. The probability of a causal connection between alert A and alert B can vary significantly depending on the contextual environment involving other concurrent alerts.
- **O2**. The historical co-occurrence of alert A and alert B plays a significant role in determining the likelihood of a causal relationship between these two alerts.
- **O3**. The summation of the probabilities for all potential causal links associated with alert A, denoted as $(\sum weight(i \rightarrow A)$, should be equal to 1.

For O1, it is necessary to extract the feature vector for each alert in the ADG based on both its own information and the neighboring information. Thus a graph convolutional network [28] (GCN) is very suitable for completing the task. In CPGAT, we use a shallow GCN for feature extraction, because the direct causality between two alerts usually depends on their own (or close neighborhood) contexts instead of those of other alerts. With the features from the GCN, we design a specific causality measure function a (Eqn. (7)) for causality score calculation. The higher the score between two alerts, the more likely they have a causality relationship. The details of the causality measure function are described in Section III-E.

For O2, we train *AlertRCA* using failures that have historically occurred in the system. Since the measurement function

Algorithm 1: CPGAT

Input: V, the alert vectors; G, the ADG
Output: A causality score, G^*, the ACG
Parameters: $h = 2$, the depth of GCN. $\beta = 0$.
$H = V$
for i = *1 to* h **do**
  H = GCN(H, G)
end
initialize A as empty dict
add self-loop edge to each node in G
for u, v *as the edge in* G **do**
  $x = H_u$
  $y = H_v$
  $A_{u,v} = \hat{a}(x, y, \beta)$
end
combine G and A to generate G^*
return G^*

contains learnable parameters, our model can take into account the historical frequency of co-occurring alert pairs.

For O3, it is necessary to normalize the attention scores of all in-edges of a certain alert. We propose a novel causality measure function $\hat{a}$ (Eqn. (8)) to better model the causality relationship between alerts. Section III-E will introduce the method in details.

In a nutshell, Algorithm 1 describes the workflow of CPGAT, where H_u denotes the hidden vector of node u obtained from GCN. CPGAT takes ADG and alert vectors of all alerts as input and outputs the causality scores between alerts. Then we combine ADG and causality scores to generate ACG.

E. Causality Measure Function

Here we describe how we design our causality measure function used in CPGAT. In the first place, we exploit the following widely used measure functions in graph representation learning [26], [29], [30]:

$$a_0(x, y) = sum(x \cdot y) \tag{1}$$
$$a_1(x, y) = sum(v \cdot tanh(x + y)) \tag{2}$$
$$a_2(x, y) = sum(x \cdot y)/\sqrt{D} \tag{3}$$
$$a_3(x, y) = sum(x^T W y) \tag{4}$$
$$a_4(x, y) = sum(v \cdot (x \| y)) \tag{5}$$

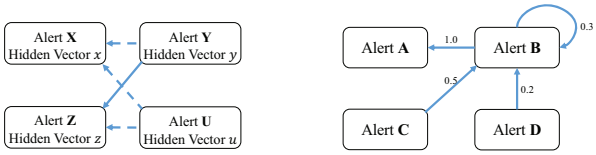
where D is the dimension of the input vector.

However, these simple measure functions are not suitable for root cause analysis (RCA). For example, for alert A and alert B, the attention score a_0, a_1, a_2 will give the same score at the link $A \to B$ and the link $B \to A$. However, there is no relationship between "A causes B" or "B causes A." a_3 is an asymmetric attention score but not practical because the size of the parameter matrix W in a_3, $D \times D$, is too large for training. a_4 is a popular asymmetric measure function used in graph attention networks [31], which can be simplified to the following formula (the vector v is the learnable parameters and can be divided into two parts v_0, v_1 where $v = v_0 \| v_1$)

$$a_4(x, y) = sum(v_0 \cdot x) + sum(v_1 \cdot y) = n_0(x) + n_1(y) \tag{6}$$

where $n_0(x) = sum(v_0 \cdot x)$ and $n_1(y) = sum(v_1 \cdot y)$.

This simplification indicates that the causality score given by a_4 at each link is the sum of the scores of two nodes on both sides. However, it brings some troubles into RCA tasks.



(a) (b)

Fig. 6. The dotted/solid lines in Fig. 6a indicate the fake/real causality links. Fig. 6b shows self-loop attention and normalization.

a_4 is not suitable for RCA since it induces an unexpected potential relationship between causality scores. For example, as shown in Fig. 6a, there is one real causal link, *i.e.,* $Y \to Z$, in the ADG. Therefore, $a_4(x, y) = a_4(z, u) = a_4(x, u) = 0$ and $a_4(z, y) = 1$ are expected. Then we will have:

$$n_0(x) + n_1(y) = n_0(z) + n_1(u) = n_0(x) + n_1(u) = 0$$

Then $a_4(z, y) = n_0(z) + n_1(y) = (n_0(x) + n_1(y)) - (n_0(x) + n_1(u)) + (n_0(z) + n_1(u)) = 0 + 0 + 0 = 0$. It contradicts our expectation that $a_4(z, y) = 1$.

In conclusion, these existing measure functions above are unsuitable for RCA. Inspired by self-attention in Transformer [26], we propose a novel casual measure function for attention mechanism:

$$a(x, y) = \sum_{h=1}^{H} (\sum_i (\sum_j Q_{hij} x_j) * (\sum_j K_{hij} y_j))/\sqrt{D} \tag{7}$$

where D is the dimension of the input vector and Q, K are multi-head learnable matrices whose shape is $H \times D_a \times D$ (H is the multi-head number and D_a is the attention dimension, which is relatively small). In Section IV-D, we evaluated all the measure functions we mentioned and proved the effectiveness of our proposed function a shown in Eqn. (7).

According to the definition of the causal link, *i.e.,* $A \to B$ represents that A causes B. The sum of the attention scores at the in-edges of alert B should be 1 to satisfy the mathematical rules of the causality relationship. Thus we propose a normalization method based on Eqn. (7) to get Eqn. (8). Besides, we define the **self-loop causality score**. The self-loop causality score $a(x, x)$ of an alert whose hidden vector is x aims to measure the probability that no other alert causes it. Meanwhile, the self-loop causality score is also an important factor in determining whether an alert is the root cause (see Section III-F). Fig. 6b shows an example where there is a 30% chance that other alerts do not cause Alert B.

On the other hand, we will remove the links whose predicted causality score is too low to simplify the topology of the ADG. Therefore we set the causality score as 0 if the attention score

predicted by CPGAT is too low (less than β, a hyperparameter whose default value is 0). The causality score is:

$$\hat{a}(x, y, \beta) = \frac{max(0, \exp a(x, y) - \beta)}{\sum_z max(0, \exp a(x, z) - \beta)} \tag{8}$$

where z iterates the hidden vectors of nodes at all in-edges of the alert that x belongs to and x itself for the self-loop causality score.

F. Dispersing and Aggregating Graph Neural Network

The primary function of the Dispersing and Aggregating Graph Neural Network (DAGNN) is to allocate a score, r_u, to every alert present in the ACG generated by CPGAT. This score signifies the likelihood of a specific alert event being the root cause behind a particular system failure. The Algorithm 2 shows the process of DAGNN.

Generally speaking, in a failure, if A is the root cause, then it will satisfy the following three conditions:

- **C1.** Alert A causes numerous neighbor nodes to be anomalous.
- **C2.** Few other alerts lead to A. Mostly A is self-looping.
- **C3.** The higher frequency of Alert A appearing in the history of system failures, the greater the likelihood of Alert A being the root cause of the current system failure.

As shown in Algorithm 2, DAGNN first generates a vector representation, c_u (Line 18), for each alert in the ACG that is able to reflect how the alert satisfies the three conditions above. And then an MLP with $Softmax(\cdot)$ as the activation function computes a suspicious score, r_u (Line 19), for each alert. DAGNN uses two special designs to obtain more informative representations: (1) dispersing aggregating structure; (2) self-residual structure.

Dispersing aggregating structure: Traditional GAT [31] aggregates the features of a node's neighbors in each iteration with the corresponding scores. This process can be simplified as the following formula:

$$x'_j = \sigma(\sum_{i \in N_j} \alpha_{i,j} x_i) \tag{9}$$

where x denotes the features of nodes, σ denotes a neural network such as MLP, N_j denotes the neighbors of node j, and α denotes a measure score. And we have $\sum_{i \in N_j} \alpha_{i,j} = 1$. Recall the definition of the causal link, *i.e.,* $A \rightarrow B$ represents that A causes B. To make the node features represent the information about the **C1**, DAGNN aggregates for each node the features of all nodes linking to it at each iteration. And the weights are the causality scores (Eqn. (8)) from other nodes to the node receiving the messages:

$$x'_j = \sigma(\sum_{j|i \in N_j} \hat{a}(x_i, x_j, \beta) x_i) \tag{10}$$

This process, Eqn. (10), is **dispersing** and **aggregating**. Because it can be regarded as two steps. First, the feature of each node is dispersed to every out-edge except the self-loop edge according to the corresponding causality score. Second, the information is aggregated from the in-edges of each node.

Algorithm 2: DAGNN

```
1  Input: G, the ACG; A, the causality scores; V, the
     alert vectors for all nodes in G.
2  Output: r, Root Cause Probability of each node.
3  Parameters: h = 20, the number of layers of DAGNN.
4  x = V
5  for k = 1 to h do
6      xlast = x
7      initialize x as empty dict
8      for u, v as the edge in G do
9          x_v = x_v + xlast_u * A_{u,v}
10     end
11     x = MLP_k(x)
12     if k < h then
13         x = x + V
14     end
15 end
16 initialize r as empty dict
17 for u as the node in G do
18     c_u = (x_u || V_u) * A_{u,u}
19     r_u = MLP(c_u)
20 end
21 return r
```

Self-residual structure: DAGNN also proposes a self-residual structure for **C2** and **C3**. The traditional graph neural networks use the simple hierarchical structure like Eqn. (9), which ignores self-loop attention and thus, also gradually loses the feature of the node itself in the iterations. According to the physical meaning of self-loop attention, an alert with higher self-loop attention is more likely not to be caused by other alerts and thus more likely to be the root cause. Inspired by this, DAGNN proposes a self-residual structure. Specifically, DAGNN directly concatenates a node's feature vector and its alert vector in the final representation, c_u, while multiplying by its self-loop causality score. This makes c_u have information of the alert (**C2**), while also reflecting the likelihood of other alerts causing it (**C3**). Meanwhile, at the end of each iteration of DAGNN, we add the alert vector of each node directly to the corresponding feature vector. In this way, the information from the shallow layer will contribute to the final representation to mitigate the over-smoothing.

Finally, a two-layer MLP with $Softmax(\cdot)$ as the output activation is adopted on c to infer the probability, r_u, that a node is the root cause, called root cause probability. For a failure, All alerts in the ACG are ranked by their root cause probabilities in the descending order. SREs can then examine them one by one.

IV. Experiments

We conduct several experimental studies to answer the following research questions for evaluating *AlertRCA*:

- **RQ1.** How effective and efficient is *AlertRCA*?

- **RQ2.** How is the performance of *AlertRCA* in the early stage of the system, where the training data are insufficient?
- **RQ3.** Does each main module contribute to *AlertRCA*?
- **RQ4.** Can CPGAT and DAGNN work well in other systems besides the alert-based system?

A. Setup

1) Dataset: We evaluate our model, *AlertRCA*, for RQ1-RQ3 on real-world e-commerce system data, serving 185 million active users across 3 data centers with 5000 services. We use two types of labeled datasets, service-based and business-based, with 782 and 170 failures respectively from Jan 2020 to Apr 2021. Each failure is resolved, labeled, and verified by the SRE team. Each dataset is split 50/50 for training and testing.

For RQ4, we use DejaVu's A1 and A2 datasets [20] to assess CPGAT and DAGNN performance when given metric data instead of alert metadata. These datasets are from a production system at a major ISP, with ten failure types. We replace Alert2Vec with DejaVu's feature extractor to handle this data, and use the same training and testing sets for credibility. Only A1 and A2 datasets are used, due to their singular root causes per failure, aligning with our model's training objectives.

2) Baseline: Our baseline approaches include:

- **Groot** [21]. Groot uses a pre-defined manual rulebook to construct a causal graph for a failure. Then a customized PageRank is applied to locate the root cause.
- **GrootN**. We replace the input of Groot, a causal graph, with the corresponding ADG. In this way, we evaluate the performance of Groot in the absence of expert rules.
- **PageRank-SDG**. It applies the PageRank to the SDG for service-level root cause localization.
- **GCN** [32]. Graph Convolutional Network, GCN is a kind of basic graph neural network. It has been widely used in node classification. RCA in ADG can be seen as a special case of binary node classification, which calculates a score $\in [0, 1]$ for each alert.
- **GraphSAGE** [28]. GraphSAGE is an optimized GCN for the embedding of unseen nodes.
- **GAT** [31]. Graph Attention Network, GAT introduces an attention mechanism to improve GCN that assigns weights to different adjacent nodes.
- **DejaVu** [20]. DejaVu represents a failure as a graph similar to our ADG. The difference is that nodes in DejaVu are a series of metric data, while each node of ADG is an alert. DejaVu uses GRU [33] and GAT to find the root cause nodes.

To our best knowledge, there is no end-to-end RCA algorithm designed based on the analysis of alert data. Therefore, we evaluate PageRank at the service level. Some graph neural networks, such as GCNs [32], GraphSAGE [28], and GAT [31], are popular and widely used in graph-related tasks like node classification and link prediction. The BERT [26] is used in GCN, GraphSAGE, and GAT to transform alerts into node features. To prove the effectiveness of the RCA algorithm (CPGAT and DAGNN) in *AlertRCA*, we evaluate *AlertRCA* on a system not generating alert data, and compare its performance with DejaVu, a state-of-the-art RCA method for time series data.

TABLE I
PERFORMANCE OF *AlertRCA* AND BASELINES, WHERE MC INDICATES WHETHER THE MANUAL CONFIGURATION IS USED, AND DL INDICATES WHETHER DEEP LEARNING IS USED.

Model	MC	DL	Service-based		Business domain	
			Top-1	Top-3	Top-1	Top-3
Groot	✓	×	0.743	0.924	0.812	0.955
GrootN	×	×	0.171	0.488	0.232	0.455
PageRank-SDG	×	×	0.161	0.253	0.012	0.018
GCN	×	✓	0.293	0.573	0.692	0.853
GraphSAGE	×	✓	0.622	0.781	0.811	0.937
GAT	×	✓	0.122	0.476	0.605	0.792
AlertRCA	×	✓	**0.774**	**0.968**	**0.853**	**0.967**

3) Metric: We use top-1 and top-3 accuracy rates as the metrics. The top-k accuracy represents the ratio of failures whose root causes can be found by checking the first top-k recommendations.

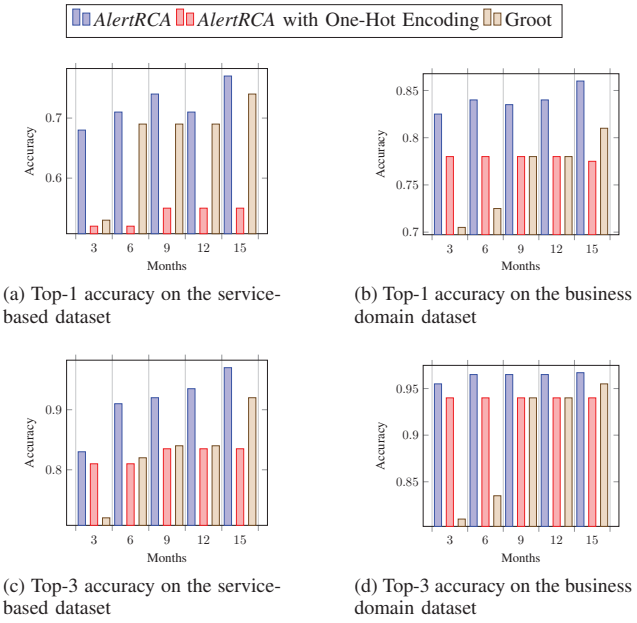
In experiments for RQ4, we replace the Alert2Vec with the GRU feature extractor in DejaVu to deal with metric data. Meanwhile, the hyperparameters of *AlertRCA* are almost identical to those of Dejavu.

B. RQ1: Effectiveness and Efficiency

Table I shows the results of RCA accuracy of *AlertRCA* and baselines. We repeat each experiment five times and present the average results for eliminating the randomness in gradient descent. The experiments in other RQs are conducted several times as well. Except for Groot with the manual rulebook and service-level algorithm, the top-1 accuracy of *AlertRCA* outperforms the baselines by 5.2%∼534.4%, and on the top-3 accuracy, *AlertRCA* outperforms the baselines by 3.2%∼112.5%. With domain knowledge, Groot outperforms other baselines and is slightly less effective than *AlertRCA*. The popular graph neural network methods (GCN, GraphSAGE, and GAT) do not perform well because they aggregate messages from multi-hop neighbors to form the alert features, which only contain the information about **C1** or **C3** in Section III-F. Thus, they lack the comprehensive information needed in RCA to determine whether an alert is the root cause. Based on the traditional PageRank methods that run RCA on ADG, GrootN performs poorly because there are many fake causal links that should have been filtered out by manual rules. PageRank is not able to deal with incorrect causal relationships.

To evaluate the efficiency of *AlertRCA*, we measure the training and inference time of *AlertRCA* without using GPU. Since the hidden dimension is small in *AlertRCA*, the average training time for one failure of *AlertRCA* is only 12s, and the average inference time per failure is only 2s, satisfying the requirement for an online production system.

Conclusion 1. *AlertRCA* is more effective in locating root cause alerts even when compared to the RCA algorithm with manual rules. Meanwhile, *AlertRCA* is efficient enough to support RCA in online systems.



(a) Top-1 accuracy on the service-based dataset

(b) Top-1 accuracy on the business domain dataset

(c) Top-3 accuracy on the service-based dataset

(d) Top-3 accuracy on the business domain dataset

Fig. 7. Performance of *AlertRCA*, *AlertRCA* with one-hot encoding, and Groot.

C. RQ2: Quick-Start

We study how *AlertRCA* performs when the running time of the system is not very long. When deploying a new microservice system in a short time, the collected training data are insufficient. In practice, Groot has a similar problem: SREs need much time discussing and designing the rulebook.

To evaluate *AlertRCA* and Groot in this situation, we sort the failures of both datasets by time and then use only the first $m(m = 3, 6, 9, 12, 15)$ months of data for training and testing. The results are shown in Fig. 7, where One-hot Encoding *AlertRCA* replaces BERT in Alert2Vec.

Both the top-1 accuracy and the top-3 accuracy of Groot drop significantly, 0.105∼0.21, when only three months of data are used which are insufficient for SREs to organize a comprehensive set of rules. However, the performance of *AlertRCA* degrades only 0.012∼0.14, because the pre-trained model in *AlertRCA* has learned extensive semantic knowledge from a large corpus. One-hot *AlertRCA* has no obvious performance difference with different data used, as the one-hot encoder does not require much data to train.

Conlcusion 2. *AlertRCA* is able to reach a promising performance after a new microservice system is started in 3 months. Therefore, *AlertRCA* is a quick-start RCA approach.

D. RQ3: Abalation Study

There are three major modules in *AlertRCA*, namely Alert2Vec, CPGAT, and DAGNN. To understand the function of each module, we evaluate *AlertRCA* with these modules removed or replaced:

- BERT
- Causality Measure Functions in CPGAT.
- Dispersing Aggregating Structure in DAGNN.
- Self-residual Structure in DAGNN.

TABLE II
ABLATION STUDY ABOUT MODELS FOR THE NATURAL LANGUAGE PROCESS IN ALERT2VEC MODULE.

Model	Service-based		Business domain	
	Top 1	Top 3	Top 1	Top 3
BERT	**0.774**	**0.968**	**0.853**	**0.967**
GloVe	0.613	0.871	0.800	0.946
Word2Vec	0.581	0.871	0.813	0.953
Fasttext	0.548	0.839	0.800	0.953
One-hot Encoding	0.548	0.839	0.773	0.940

TABLE III
ABLATION STUDY ABOUT MEASURE FUNCTIONS.

Model		Service-based		Business domain	
		Top 1	Top 3	Top 1	Top 3
AlertRCA		**0.774**	**0.968**	**0.853**	**0.967**
Replace Attention by	a_0	0.710	0.871	0.827	0.973
	a_1	0.774	0.903	0.813	0.987
	a_2	0.613	0.806	0.793	0.967
	a_3	0.742	0.935	0.773	0.953
	a_4	0.613	0.871	0.806	0.980

Table II shows how the language model improves the performance of *AlertRCA*. BERT outperforms other popular NLP methods (GloVe [34], Word2Vec [35] and Fasttext [36] and the simple one-hot encoding), especially in the service-based dataset.

Conclusion 3. BERT is essential for the Alert2Vec module to understand the semantics in alerts.

In Table III, we explore how the attention score function improves the performance of *AlertRCA*. We compare the traditional measure functions $a_0 \sim a_4$ mentioned in Section III-E and the causality score Eqn. (8) adopted in *AlertRCA*. It is worth noting that in the business domain dataset, our causality score improves the top-1 accuracy and lowers the top-3 accuracy. After analyzing testing cases, we find that the root cause alerts of some failures rank second or third when other measure functions are used, and rank first when the causality score was used. This is because we introduce the self-loop attention representing the probability that an alert is not caused by others. When several alerts all cause a failure, the self-loop attention helps distinguish the alert not caused by other alerts, *i.e.,* the root cause alert.

Conclusion 4. In RCA, the normalized causality attention can help to locate the root cause alert more precisely.

We conduct three comparative experiments to study how the DAGNN contributes to *AlertRCA* and summarize the results in Table IV:

1) ***AlertRCA* w/o SS**: Remove the self-residual structure. The alert vector V is not added in each iteration, and only x is retained in the final representation c.
2) ***AlertRCA* w/o DAS**: Remove the dispersing aggregating structure, where we apply Eqn. (9) in each iteration like GAT.

3) ***AlertRCA* w/o SA**: Remove the self-loop attention and set $A_{u,u} = 1$.

These three parts correspond to the three properties listed in Section III-F (**C1**∼**C3**) of the root cause. As can be seen in Table IV, *AlertRCA* decreases to some extent on both top-1 accuracy and top-accuracy (except for the effect of self-loop attention on top-3 accuracy, which is consistent with the conclusion above).

Conclusion 5. Compared with the simple hierarchical architecture, the self-residual structure, dispersing aggregating structure, and self-loop attention are crucial to *AlertRCA*.

TABLE IV
ABLATION STUDY ABOUT DAGNN.

Model		Service-based		Business domain	
		Top 1	Top 3	Top 1	Top 3
AlertRCA		**0.774**	**0.968**	**0.853**	**0.967**
w/o	SA	0.613	0.935	0.806	0.986
	DAS	0.451	0.817	0.813	0.942
	SS	0.452	0.903	0.720	0.866

E. RQ4: Metric as Input

We investigate whether CPGAT and DAGNN in *AlertRCA* are effective on other systems by conducting experiments on datasets, A1 and A2 in DejaVu [20], from a system generating time series data instead of alert data.

Table V shows the performance of the two methods on the A1 and A2 datasets respectively. *AlertRCA* and DejaVu do not differ significantly in terms of top-3 accuracy. However, the top-1 accuracy of *AlertRCA* outperforms DejaVu by 24.8% in A1 and 15.7% in A2, which is significant. It is mainly caused by the following reasons. First, DejaVu uses a GAT-based structure to aggregate the features for each of its neighbors. As mentioned in Section III-E, the measure function in GAT (a_4) does not meet the requirements of RCA both on how they are calculated and how they are normalized. Second, Dejavu feeds the aggregated feature of each node directly into the MLP for classification. However, without the self-residual structure, a node is unable to retain its own information sufficiently when GAT propagates information, making it more difficult to find the correct root cause.

Conclusion 6. CPGAT and DAGNN in *AlertRCA* can work well in other systems that do not generate alerts.

TABLE V
PERFORMANCE ON METRIC DATASETS

Model	Dataset A1		Dataset A2	
	Top 1	Top 3	Top 1	Top 3
AlertRCA	**0.875**	**0.968**	**0.723**	**0.883**
DejaVu	0.701	0.937	0.625	0.851

V. RELATED WORKS

In recent years, many approaches have been proposed to tackle the RCA problem in online service systems. These approaches capture the system's state by instrumentation or monitoring metrics. Then they construct an SDG to represent the causality relationship. However, existing methods [8], [12], [18] have limited effectiveness in industrial environments.

In terms of operational complexity, [37], [38] has good adaptability to the diverse technology stack. In terms of scale and complexity, [37]–[40] are only evaluated in a small-scale environment, with few services (< 100) and no real events; [41] support limited types of events with manual causality configurations. In terms of monitoring complexity, existing log analysis-based methods [14]–[17], [41] can not adapt to the frequent context changes caused by the rapidly evolving technology stack. These and most existing approaches cannot prove practical effectiveness due to the limited coverage of diversified root cause types and validations in small-scale and non-production environments with few actual failures.

DejaVu [20] performs RCA on the failure dependency graph where a node represents a metric set. It cannot be used to deal with alert data. Meanwhile, DejaVu directly employs GAT for feature aggregation of nodes without considering the physical meaning of RCA and is unable to get high top-1 accuracy.

Groot [21] presents a flexible and effective graph-based framework for RCA to deal with the complexities of large-scale microservice systems. However, constructing a causal graph heavily relies on time-consuming manual configurations making Groot infeasible in the real-world environment.

VI. CONCLUSION

In this paper, we introduce *AlertRCA*, a novel root cause analysis model leveraging alert events for large-scale online service systems. *AlertRCA* addresses two major limitations in existing methods, obtaining a practical and accurate causal graph and designing a specific deep learning-based approach for alert-based RCA. To construct the causal graph, we propose an automated process involving Alert Dependency Graph (ADG), Alert2Vec, and Causality Prediction Graphical Attention Network (CPGAT). We design a unique Dispersing and Aggregating Graph Neural Network (DAGNN) to identify the root causes of system failures.

AlertRCA is a novel RCA algorithm due to its practicality and accuracy, and it uses widely recognized alert events as input, making it accessible for companies of all sizes. It circumvents the dependence on handcrafed rules, and is designed to be explicitly tailored for alert-based RCA scenarios. Our extensive evaluations on real-world and open-source datasets underscore *AlertRCA*'s superior performance, demonstrating its effectiveness and feasibility for real-world applications. We believe *AlertRCA* opens a promising direction for RCA in large-scale online service systems, providing a practical tool to swiftly and accurately address service failures, thus ensuring the seamless functioning of these vital systems.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under Grant No.62072264 and No.62202445, and the Beijing National Research Center for Information Science and Technology (BNRist) key projects.

REFERENCES

[1] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables devops: Migration to a cloud-native architecture," *Ieee Software*, vol. 33, no. 3, pp. 42–52, 2016.

[2] Z. Yu, C. Pei, S. Zhang, X. Wen, J. Li, G. Xie, and D. Pei, "Autokad: Empowering kpi anomaly detection with label-free deployment," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, 2023, pp. 13–23.

[3] W. Meng, Y. Liu, S. Zhang, F. Zaiter, Y. Zhang, Y. Huang, Z. Yu, Y. Zhang, L. Song, M. Zhang *et al.*, "Logclass: Anomalous log identification and classification with partial labels," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1870–1884, 2021.

[4] N. Zhao, J. Chen, Z. Yu, H. Wang, J. Li, B. Qiu, H. Xu, W. Zhang, K. Sui, and D. Pei, "Identifying bad software changes via multimodal anomaly detection for online service systems," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 527–539.

[5] N. Zhao, P. Jin, L. Wang, X. Yang, R. Liu, W. Zhang, K. Sui, and D. Pei, "Automatically and adaptively identifying severe alerts for online service systems," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2420–2429.

[6] J. Xu, Y. Wang, P. Chen, and P. Wang, "Lightweight and adaptive service api performance monitoring in highly dynamic cloud environment," in *2017 IEEE International Conference on Services Computing (SCC)*. IEEE, 2017, pp. 35–43.

[7] L. Tang, T. Li, F. Pinel *et al.*, "Optimizing system monitoring configurations for non-actionable alerts," in *2012 IEEE Network Operations and Management Symposium*. IEEE, 2012, pp. 34–42.

[8] M. Solé, V. Muntés-Mulero, A. I. Rana, and G. Estrada, "Survey on models and techniques for root-cause analysis," *arXiv preprint arXiv:1701.08546*, 2017.

[9] J. Mace, R. Roelke, and R. Fonseca, "Pivot tracing: Dynamic causal monitoring for distributed systems," in *Proceedings of the 25th Symposium on Operating Systems Principles*, 2015, pp. 378–393.

[10] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng *et al.*, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 187–196.

[11] M. Ma, W. Lin, D. Pan, and P. Wang, "Ms-rank: Multi-metric and self-adaptive root cause diagnosis for microservice applications," in *2019 IEEE International Conference on Web Services (ICWS)*. IEEE, 2019, pp. 60–67.

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

[13] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, "Microrca: Root cause localization of performance issues in microservices," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–9.

[14] M. K. Aguilera, J. C. Mogul, J. L. Wiener *et al.*, "Performance debugging for distributed systems of black boxes," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 74–89, 2003.

[15] H. Zawawy, K. Kontogiannis, and J. Mylopoulos, "Log filtering and interpretation for root cause analysis," in *2010 IEEE International Conference on Software Maintenance*. IEEE, 2010, pp. 1–5.

[16] S. Lu, B. Rao, X. Wei *et al.*, "Log-based abnormal task detection and root cause analysis for spark," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 389–396.

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

[18] M. Kim, R. Sumbaly, and S. Shah, "Root cause detection in a service-oriented architecture," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 1, pp. 93–104, 2013.

[19] H. Wang, P. Nguyen, J. Li, S. Kopru, G. Zhang, S. Katariya, and S. Ben-Romdhane, "Grano: Interactive graph-based root cause analysis for cloud-native distributed data platform," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 1942–1945, 2019.

[20] Z. Li, N. Zhao, M. Li, X. Lu, L. Wang, D. Chang, X. Nie, L. Cao, W. Zhang, K. Sui, Y. Wang, X. Du, G. Duan, and D. Pei, "Actionable and interpretable fault localization for recurring failures in online service systems," in *Proceedings of the 2022 30th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2022, Nov. 2022.

[21] H. Wang, Z. Wu, H. Jiang, Y. Huang, J. Wang, S. Kopru, and T. Xie, "Groot: An event-graph-based approach for root cause analysis in industrial settings," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 419–429.

[22] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR (Poster)*. OpenReview.net, 2017.

[23] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[24] L. Lovász, "Random walks on graphs," *Combinatorics, Paul erdos is eighty*, vol. 2, no. 1-46, p. 4, 1993.

[25] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[26] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[27] OpenAI, "Gpt-4 technical report," *ArXiv*, vol. abs/2303.08774, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:257532815

[28] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 1025–1035.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[30] P. Fang, J. Zhou, S. K. Roy, L. Petersson, and M. Harandi, "Bilinear attention networks for person retrieval," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 8030–8039.

[31] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *ICLR*, 2018.

[32] N. T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *international conference on learning representations*, 2017.

[33] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[34] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[35] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, 2013.

[36] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[37] H. Nguyen, Z. Shen, Y. Tan, and X. Gu, "Fchain: Toward black-box online fault localization for cloud systems," in *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, 2013, pp. 21–30.

[38] Á. Brandón, M. Solé, A. Huélamo *et al.*, "Graph-based root cause analysis for service-oriented and microservice architectures," *Journal of Systems and Software*, vol. 159, p. 110432, 2020.

[39] J. Weng, J. H. Wang, J. Yang, and Y. Yang, "Root cause analysis of anomalies of multitier services in public clouds," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1646–1659, 2018.

[40] J. Qiu, Q. Du, K. Yin *et al.*, "A causality mining and knowledge graph based method of root cause diagnosis for performance anomaly in cloud applications," *Applied Sciences*, vol. 10, no. 6, p. 2166, 2020.

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