Chain-of-Event: Interpretable Root Cause Analysis for Microservices through Automatically Learning Weighted Event Causal Graph

Zhenhe Yao
Tsinghua University & BNRist
Beijing, China

Changhua Pei*†
HIAS, UCAS & CNIC, CAS
Beijing, China

Wenxiao Chen
Tsinghua University
Beijing, China

Hanzhang Wang
eBay Inc.
Shanghai, China

Liangfei Su
eBay Inc.
Shanghai, China

Huai Jiang
eBay Inc.
Shanghai, China

Zhe Xie
Tsinghua University
Beijing, China

Xiaohui Nie
CNIC, CAS
Beijing, China

Dan Pei
Tsinghua University & BNRist
Beijing, China

ABSTRACT

This paper presents *Chain-of-Event* (CoE), an interpretable model for root cause analysis in microservice systems that analyzes causal relationships of events transformed from multi-modal observation data. CoE distinguishes itself by its interpretable parameter design that aligns with the operation experience of Site Reliability Engineers (SREs), thereby facilitating the integration of their expertise directly into the analysis process. Furthermore, CoE automatically learns event-causal graphs from history incidents and accurately locates root cause events, eliminating the need for manual configuration. Through evaluation on two datasets sourced from an e-commerce system involving over 5,000 services, CoE achieves top-tier performance, with 79.30% top-1 and 98.8% top-3 accuracy on the Service dataset and 85.3% top-1 and 96.6% top-3 accuracy on the Business dataset. An ablation study further explores the significance of each component within the CoE model, offering insights into their individual contributions to the model's overall effectiveness. Additionally, through real-world case analysis, this paper demonstrates how CoE enhances interpretability and improves incident comprehension for SREs. Our codes are available at https://github.com/NetManAIOps/Chain-of-Event.

CCS CONCEPTS

• **Software and its engineering** → **Software reliability**; **Software performance**.

*Corresponding author.
†HIAS, UCAS stands for Hangzhou Institute for Advanced Study, University of Chinese Academy of Sciences.

FSE Companion '24, July 15–19, 2024, Porto de Galinhas, Brazil

ACM ISBN 979-8-4007-0658-5/24/07
https://doi.org/10.1145/3663529.3663827

KEYWORDS

Interpretable Root Causal Localization, Event-based Root Cause Analysis

ACM Reference Format:
Zhenhe Yao, Changhua Pei, Wenxiao Chen, Hanzhang Wang, Liangfei Su, Huai Jiang, Zhe Xie, Xiaohui Nie, and Dan Pei. 2024. *Chain-of-Event*: Interpretable Root Cause Analysis for Microservices through Automatically Learning Weighted Event Causal Graph. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE Companion '24), July 15–19, 2024, Porto de Galinhas, Brazil.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3663529.3663827

1 INTRODUCTION

In the realm of software engineering, microservice architecture has emerged as a revolutionary approach, enabling the development of software applications as collections of independently deployable, small, modular services, offering enhanced scalability and reusability, significantly speeding up the software development lifecycle. Despite its broad adoption due to these advantages, the architecture is inherently susceptible to critical production incidents. Such incidents refer to unexpected system disruptions or component failures at any level, which can have substantial adverse effects on business operations [2, 6, 26, 29, 30]. These challenges underscore the importance of swiftly identifying and addressing the root causes of any issues to ensure quick system recovery. This capability is vital for maintaining the high reliability and robustness of software systems in a microservice architecture, a cornerstone principle in software engineering.

The incident recovery process for software reliability maintenance includes three key stages: anomaly detection [33, 40, 45], root cause analysis (RCA) [31], and remediation. While the detection and remediation phases follow well-defined procedures, the RCA stage frequently poses challenges for Site Reliability Engineers (SREs).

Numerous prior studies have attempted to identify diverse root causes, employing data-type-specific approaches like metric-based methods [19, 22–24, 32, 34, 37, 39], log-based methods [1, 7, 21, 25,

44] and trace-based methods [38, 41]. However, while SREs perform RCA with various forms of observation data, including metrics, logs, and traces, as highlighted in [10, 42], these previous methods only process a specific type of observation data. This constraint restricts these approaches from constructing a comprehensive understanding of the system and deriving the most accurate root cause conclusions [42], which introduces the first challenge in RCA:

> **Challenge 1: Multi-Modal Data Integration.** RCA methods should effectively integrate and analyze multi-modal data, leveraging information from various observation types, including metrics, traces, and logs.

To address these challenges, researchers recently proposed algorithms that can handle multi-modal data simultaneously. Eadro [16] feeds multi-modal data into a deep neural network, learns the correlation and status representations for the multi-modal data through a three-stage modal fusion, and performs root cause localization based on the status representations. Nezha [42] converts multi-modal data into events related to user requests and ranks the root causes based on the changes in statistics of the event patterns.

Although these algorithms can utilize multi-modal observation data, they suffer from limited interpretability and do not facilitate straightforward human feedback. Eadro's method uses neural networks throughout a black-box three-stage modal fusion. For SREs who usually do not have a background in deep learning, it is difficult for them to judge whether the parameters learned from the intermediate layers (such as the self-attention layer) of these models are reasonable, and they cannot optimize the model's performance without experience in parameter tuning. Nezha introduces the concept of event pattern, which is relatively abstract, making it difficult for SREs to intuitively understand what physical meaning is represented by the event pattern's increasing, decreasing, or transforming to another pattern. For RCA models with poor interpretability, it is hard for SREs to utilize their valuable experience accumulated in daily microservice system maintenance to improve the model, limiting the practicality and reliability of these RCA models, which introduces the second challenge:

> **Challenge 2: Interpretability and Straightforward Alignment to Human Knowledge.** An interpretable RCA algorithm should make it easy for SREs to understand. Its parameter structure should ideally have a clear and intuitive physical meaning that aligns with the knowledge of SREs, such that SREs can easily improve the performance of the algorithm model by modifying the parameters based on their operational experience.

To address the previous two challenges, some other methods are compatible with multi-modal data and align with the operational experience of SREs while simultaneously suffering from labor-intensive or tricky manual configuration. PDiagnose [11] first transforms the multi-modal data into different features of a time-series, then utilizes manually defined thresholds to detect issues in the timeseries, and localizes the root cause with the detected issues. Groot [10] converts multi-modal data into events before

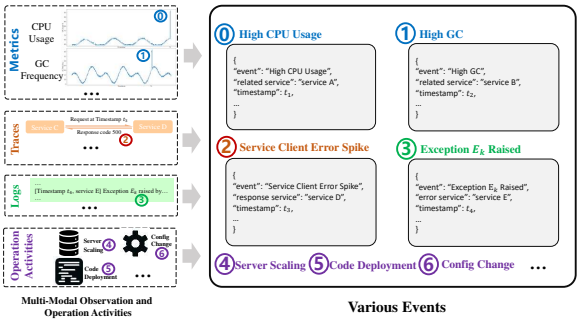


Figure 1: Illustration of Different Events. Each event comprises the event type, associated service, and timestamp information.

constructing an event-causal graph using expert-configured rules. The graph is then used to infer the root cause event through a customized PageRank algorithm. These manually configured rules specify whether one event can be triggered by another.

However, although threshold filtering and event-causal relationships closely align with SREs' operational expertise, the manual configuration required in these methods introduces an extra workload for SREs, thus posing the third challenge in RCA:

> **Challenge 3: Automatic Causality Learning.** An effective RCA algorithm should be able to automatically learn causality in microservice systems, thereby minimizing or eliminating the necessity for manual configuration.

Among all the methods performing RCA on multi-modal data, some [10, 42] introduce a framework that converts multi-modal data into events before performing root cause analysis based on these events. The transformation from multi-modal data to events is illustrated in Fig. 1. This framework allows SREs to freely choose the way events are generated based on their own needs. For example, SREs can generate events based on their chosen threshold rules or time series anomaly detection algorithms from metric data, or they can generate events with keyword parsing statistics on logs. Compared to directly mining the relationships within vast quantities of data across various modalities, learning causality at the event level can achieve generalized multi-modal root cause localization, maintaining the granularity of crucial original information[10, 42].

To address the three challenges mentioned earlier, we propose CoE. CoE utilizes an event-based RCA framework and implements workflow that automatically learns interpretable model parameters. Based on the event-based RCA framework, CoE demonstrates good adaptability to multi-modal data, and our experiments show promising performance on both service-level and business-level event datasets. The interpretability of CoE is excellent as its parameters have clear physical meanings: (1) the likelihood of one event causing another and (2) the importance of an event within the entire system. The physical meanings behind these parameters align well with SREs' operational experience, facilitating the integration of their valuable knowledge through human feedback.

These parameters of CoE are obtained through automatic learning, eliminating the need for manual configuration.

In summary, the key contributions of CoE are outlined below:

- We present an RCA algorithm, CoE, that follows the event-based RCA framework and automatically learns the causality between events in the microservices, ensuring compatibility with multi-modal data and eliminating the need for manual weight configuration.
- CoE has good interpretability and straightforward alignment to human knowledge, as its parameters with clear physical meanings, such as the likelihood of one event causing another and the importance score of an event within the microservice system, align well with SREs' operational experience, facilitating the integration of SREs' expertise.
- To address the computational overhead of enumerating all event chains when calculating the probability of root causes, we propose an approximation method and provide theoretical analysis for it, ensuring a tight upper bound on approximation error.
- Evaluation on datasets from a global top-5 *e*-commerce system shows that CoE outperforms the baselines, achieving 79.30% top-1 and 98.8% top-3 accuracy on the service dataset and 85.3% top-1 and 96.6% top-3 accuracy on the business dataset collected from online production services. Ablation studies confirm the effectiveness of model components.

2 RELATED WORKS

Recently, various methodologies have emerged for diagnosing the root causes of distributed software systems. These approaches typically leverage information about the system's state through monitoring metrics, microservice traces, system or application logs, and operational records kept by SREs.

However, a significant limitation of most prior research is the underutilization of diverse data types [4, 7, 12, 17–19, 27, 28, 36–38, 41], as illustrated in Table 1. In this table, the *C1* column indicates whether the model can handle multi-modal inputs, including metrics, traces, and logs. Notably, RCA Graph[3] is categorized as "Partial" due to its support for multi-modal inputs at the service level only.

More recent models [10, 11, 16, 42] have been designed to process metrics, traces, and logs together. However, they commonly suffer from two critical drawbacks, as presented by the columns *C2* and *C3* in Table 1:

Limited Interpretability and Restricted Human Knowledge Alignment. Eadro [16] employs deep learning techniques, including dilated causal convolution and self-attention layers, to learn system status representations. While effective, this approach reduces model interpretability, making it challenging for SREs without a deep learning background to comprehend the model and fine-tune its parameters. Nezha [42], although a white-box model, transforms multi-modal inputs into events and infers root causes based on the relative changes in event pattern frequencies between fault-free and fault stages. However, Nezha's interpretability is also limited because the concept of event pattern frequency is somewhat abstract and does not directly align with SREs' practical experience. For example, while it may detect a significant change from $e_1 \rightarrow e_2$ to $e_1 \rightarrow e_3$ as root cause, it may not clearly convey the implications of this "root cause" change to SREs.

Table 1: Recent Works and Their Compliance with the Requirements from the Challenges. Here, *C1* represents whether the model can accept metrics, logs, and traces as multi-modal inputs; *C2* represents whether the model can achieve satisfying interpretability; *C3* represents needing NO manual configuration. *RW* represents whether the model is validated in *Real-World* datasets. Here, the blanks in the table represent that we do not discuss whether the models address some challenges because they have already failed to address the previous challenge(s), aiming for visual clarity.

Work	Year	*C1*	*C2*	*C3*	*RW*
FChain [27]	2013	✗			✗
CauseInfer [4]	2014	✗			✗
MicroScope [18]	2018	✗			✗
APG [36]	2018	✗			✗
Seer [7]	2019	✗			Partial
MicroRCA [37]	2020	✗			✗
Causality RCA [28]	2020	✗			✗
MicroHECL [19]	2021	✗			✓
MicroRank [41]	2021	✗			✓
RCD [12]	2022	✗			✗
Dejavu [17]	2022	✗			✓
GTrace [38]	2023	✗			✓
RCA Graph [3]	2020	Partial			✗
Eadro [16]	2023	✓	✗		✗
Nezha [42]	2023	✓	✗		✗
AlertRCA [43]	2024	✓	✗		✓
PDiagnose [11]	2021	✓	✓	✗	✓
Groot [10]	2021	✓	✓	✗	✓
CoE (ours)	now	✓	✓	✓	✓

Need for Manual Configuration. PDiagnose [11] transforms original multi-modal data into multiple time series features and relies on manually designed threshold-based rules to identify actual issues and perform RCA based on these detected issues. Groot [10] transforms multi-modal data into events and conducts event-based RCA using a manually configured event-causal graph that defines the likelihood of each event causing another. While the threshold rules in PDiagnose and the event-causal relationships in Groot enhance interpretability and enable SREs to easily optimize models based on their experience without extra cognitive burden, the manual configuration in these models proves to be labor-intensive.

Among the multi-modal approaches, some works [10, 42] adopt the strategy of initially transforming multi-modal data into fine-grained events before performing RCA on these events. This event-based RCA approach presents a flexible and effective graph-based framework for handling multi-modal inputs in microservice systems. In our work, we build upon this idea and aim to address the aforementioned limitations of previous research. Our objective is to facilitate automatic causality learning without the need for manual configuration while ensuring the model remains interpretable. Additionally, we strive to empower SREs to seamlessly integrate their valuable experience into the model.

3 PRELIMINARIES

In this section, we will first introduce the definition of events and how to generate events from monitoring metrics, microservices traces, and system logs in the microservices scenario, as well as the relative relationship between events and incidents. Then, we will introduce some concepts in event-based RCA, including causal links between events, event chains, event-causal graphs consisting of events and causal links, and naive event-causal graphs employed to construct event-causal graphs. Finally, we will summarize our problem statement.

3.1 Event and Incident

3.1.1 Event. In modern software architecture, the diversity and tremendous volume of monitoring data on distributed systems shall bring significant challenges for long-term storage and further analysis in the process of operation [10]. In our case, a global top-5 *e*-commerce system with over 5,000 distributed online services produces over 10 TB of daily monitoring data, including KPI metrics, traces, and logs. To the best of our knowledge, there is currently no appropriate algorithm to process all the incoming data of such volume directly to perform RCA at acceptable costs. Meanwhile, much of the observed data is not of concern to SREs (*e.g.,* , non-failure segments in metrics, template information in logs, and normal responses in traces).

Therefore, to tackle these challenges, a natural approach is to preprocess and aggregate data from different modalities, focusing only on the portions containing critical information, namely *events*, and perform root cause analysis at the event level. Fig. 1 illustrates the generation of events. In a microservice system, SREs can extract important abnormal information, such as unusually high CPU metrics, abnormally high Garbage Collection (GC) frequencies, increased error request codes between two microservices, or frequent exceptions reported by a specific microservice code segment, from raw observation data, including metrics, traces, and logs, using different algorithms based on their granularity requirements for analysis. Typically, an event contains three types of fields: *WHAT*, which describes the type of the event (e.g., "CPU HIGH," "GC HIGH"); *WHEN*, which indicates the time or period when the event occurred; and *WHERE*, which identifies the service associated with the event. Apart from the observation data generated automatically by the monitoring system, some operation activities performed by SREs can also be categorized as events, such as "Code Deployment" and "Config Change" in Fig. 1.

Event-based RCA refers to identifying the most likely event within a collection of events that may have caused other events, thereby determining the subsequent remediation actions to be taken by SREs. For example, an event such as "Code Deployment" may have led to events like "Latency Spike" and "API Call Timeout Error Spike." Upon discovering the root cause event, SREs would take actions such as code rollback or hotfix.

Performing RCA based on events not only (1) allows flexible utilization of multi-modal observation data but also (2) supports SREs in determining the granularity of root cause analysis based on their specific requirements. Furthermore, it (3) reduces costs by avoiding direct correlation mining on vast amounts of raw data, and additionally (4) diminishes the cognitive burden for SREs in comprehending the current system status by leveraging events rather than raw metrics, logs and traces.

Events are abstractions of raw monitoring data, which align closely with the understanding of SREs. Typically, SREs accumulate a wealth of experience regarding events during their daily operations and maintenance. For example, (1) a certain type of event is likely to be caused by another type of event; (2) a particular type of event is crucial and requires close attention when it occurs. Integrating such SRE expertise regarding events into the RCA algorithm would effectively enhance its usability.

3.1.2 Incident. Due to the complex dependencies within a microservice system, multiple events often occur simultaneously in a real failure. We consider a set of related events within a certain time window as a collection, which we call an *incident*. An incident is usually a set of events that cause service disruptions or outages[13]. Event-based RCA is to discover the root cause of an incident.

3.2 Event Graph

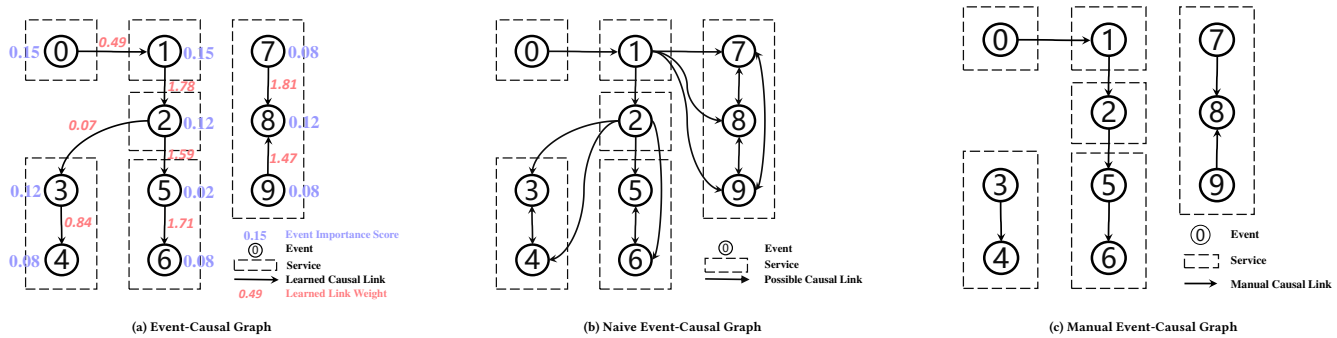
3.2.1 Causal Link. Typically, there is a causal relationship between certain events, meaning that one event can be caused by another. Considering events as nodes and establishing a directed edge between events based on potential causal relationships, the edge is referred to as a *Causal Link*. A causal link points from the resulting event e_1 to the causative event e_2, and its weight represents the likelihood that e_1 is caused by e_2.

3.2.2 Event Chain. A particular event may lead to multiple other events, directly or indirectly, through a series of causal links. We refer to these chained causal links between multiple events as an *Event Chain*. To maintain generality, we define a zero-length event chain to indicate that an event is not caused by any other event, thus being its own root cause.

3.2.3 Event-Causal Graph. The graph composed of a set of events as vertices and the causal links between these events as edges is called an *Event-Causal Graph* (ECG), as shown in Fig. 2a. There are two types of weight information in the Event-causal Graph: the weights of the vertices (event nodes), which represent the importance of an event, and the weights of the edges (causal links), which represent the likelihood that one event is caused by another. It is worth noting that these weights in the event-causal graph are consistent with the SRE experience introduced in the last paragraph of Section 3.1.1, thus providing good interpretability and easy acceptance of straightforward feedback and modifications from SRE.

Depending on the differences in the set of events used to construct the event-causal graph, the event-causal graph can be divided into two categories: (1) the event-causal graph constructed using the set of all possible events in a microservices system, referred to as the *overall* event-causal graph; (2) the event-causal graph constructed using the set of events related to a specific incident, which we call the *incident-specific* event-causal graph. The latter is generally a subgraph of the former.

In a microservices architecture, the overall event-causal graph can help SREs understand the entire system and allow for the integration of SRE knowledge. It is a useful approach to construct an incident-specific event-causal graph for a certain incident before using this graph to perform root cause localization.



(a) Event-Causal Graph (b) Naive Event-Causal Graph (c) Manual Event-Causal Graph

Figure 2: Illustrations of Event Graphs: (a) The Event-causal Graph comprises events represented as vertices and causal links represented as directed edges. The blue and red values indicate the significance of events and the strength of causal connections, respectively. (b) A Naive Event-causal Graph connects all possible causal links between events occurring within the same or adjacent services. (c) Some approaches build upon (b) to create a Manually Configured Event-Causal Graph, where expert-defined rules assign binary weights, removing non-existent causal connections by assigning them a zero weight.

3.2.4 Naive Event-Causal Graph. To help construct an event-causal graph, *Naive Event-causal Graph* (NEG) is introduced.

The NEG is a uniformly weighted and fully connected ECG where a set of events serve as vertices, and edges are formed by connecting all events within the same or adjacent microservices, as depicted in Fig. 2b. In an NEG, the edges between events within the same service are bidirectional, while the direction of the edges between events in adjacent services is determined by the dependencies (*e.g.,* the calls) of the services. The edges in an NEG represent all possible causal relationships among the events, including false causal relationships.

The NEG does not require the weights needed in the ECG, making it possible to be constructed directly from the event collection without additional causal prior knowledge. Consequently, constructing an NEG from an incident can aid in the creation of an incident-specific event-causal graph with the following approaches:

- **Approach 1**: Utilize the NEG directly as the event-causal graph while assigning all nodes and edges with equal weights.
- **Approach 2**: Initialize the event-causal graph with the NEG and supplement *binary* weights through *manually defined* rules, as depicted in Fig. 2c.
- **Approach 3**: Initialize the event-causal graph with the NEG and supplement *continuous* weights using *automatically learned* parameters.

Approach 1 is commonly utilized alongside traditional techniques like PageRank. Some works like Groot [10] employ Approach 2 to construct manually configured binary event-causal graphs. Our CoE adopts Approach 3 to eliminate additional manual effort.

3.3 Problem Statement

In the inference stage, given an incident as input, our CoE aims to infer the root cause scores for every event in the incident, indicating the root cause event. In the training stage, given historical incidents involving different events and the recorded root cause events of these incidents, our CoE seeks to automatically learn the causal link weights and the event importance scores (illustrated by the red and blue values in Fig. 2a) in an overall event-causal graph involving all events in the microservice system.

4 ARCHITECTURE

In this section, we begin with an overview of our workflow, proceed to outline the data preparation stage aimed at generating NEG (serving as the input for subsequent steps), delve into the intricacies of the inference stage of our CoE, and finally introduce the training process of CoE. In addition, we discuss how to integrate human knowledge into CoE to achieve more accurate RCA and deal with events that have not been seen before.

4.1 Workflow

In this section, we briefly present the complete workflow of our CoE, including the training and inference stages, as Fig. 3 shows.

First, in the preparation stage before training, we acquire the events of historical incidents (serving as the inputs for training) and the corresponding root cause events for these incidents (serving as the labels) from the SRE's fault remediation tickets in history. The labels are obtained via a keyword search within the tickets. It is worth noting that although CoE employs a supervised learning approach, since fault remediation is already part of the SRE's routine work, and ticket recording is a natural occurrence, this step does not incur additional resource overhead.

During the training phase, CoE trains an overall event-causal graph involving all events in the microservice system. Using the previously obtained events of historical incidents as inputs, CoE constructs an NEG for each incident with its events. Then CoE constructs an incident-specific event-causal graph for each incident by initializing it with an NEG and assigning the causal link weights and event importance scores queried from the overall event-causal graph. Subsequently, based on the incident-specific event-causal graphs, a graph ranking is performed to provide fault root cause scores for the events within each incident. Using the scores of the ground truth root cause events for each incident, a loss value is calculated and subsequently employed to update the overall event-causal graph. After training, the learned overall event-causal graph is saved for the subsequent inference stage. This overall event-causal graph offers excellent interpretability, making it easy for SREs to understand and integrate their knowledge.

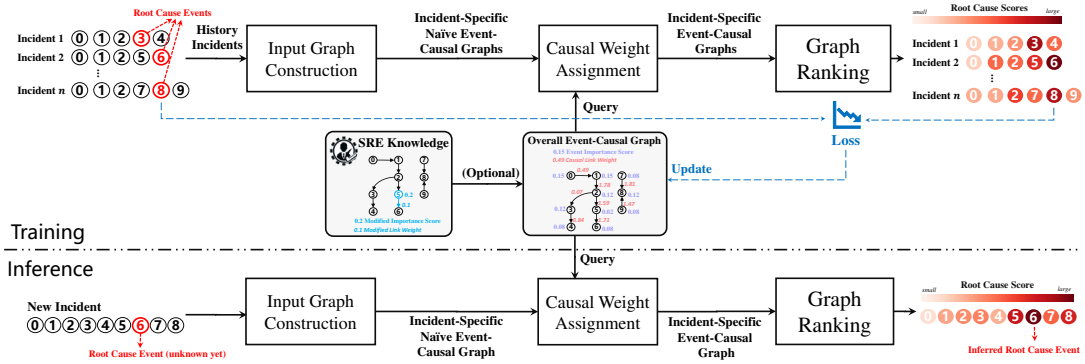


Figure 3: CoE Workflow: Training and Inference

In the inference stage, for a newly occurring incident with an unknown root cause event, CoE constructs an NEG based on the incident's event set. Then, CoE queries the saved overall event-causal graph for causal link weights and event importance scores before assigning them to the causal links and event nodes, constructing an incident-specific event-causal graph initialized by the NEG. Afterward, a graph ranking is performed to provide root cause scores for the events related to this incident. The event with the highest root cause score is expected to be the real root cause event.

4.2 Input Graph Construction

As previously introduced in Section 3.2.4, an incident can be transformed into an incident-specific NEG. In a microservice system, the invocation relationships between microservices can be easily obtained from traces and logs, while events record the associated service information. Therefore, for a given incident, CoE constructs the corresponding NEG using the following approach: the events in the incident are treated as nodes, and for any two events, (1) if they are associated with the same service, bidirectional causal links are set between them, indicating the likelihood of their mutual causation; (2) if they are associated with adjacent services, a unidirectional causal link is set between them, with the direction consistent with the service invocation dependency; (3) if there is no direct dependency relationship between the services they are associated with, no causal link is set between them.

It is worth noting that even if two events are not directly connected by a causal link, there may still be an indirect causal relationship between them, but such a causal relationship can be expressed through an event chain.

The incident-specific NEG serves as the input for inferring the root cause scores of the incident's events in the next step.

4.3 Root Cause Inference in CoE

This section introduces how to calculate the root cause score of each node in CoE based on a given NEG. Denoting a given NEG as G, we define CoE (G) as the root cause score of events in G. The algorithm is shown in Algorithm 1, which mainly includes:

4.3.1 Inputs and Parameters. The input of the CoE function is an NEG transformed from an incident, denoted as G.

The trainable parameters include R_s, R_d, and S. R_d and R_s represent the causal link weights for inter-service and intra-service causal links. S is the event importance scores of the event nodes, measuring how important each event is in the whole system and indicating a crucial event chain starting from this event. R_s,R_d, and S contain information about the event-causal graph composed of all events in the microservice system. How to update them through training will be introduced in Section 4.4.

The parameters α and T jointly determine the algorithm's approximation error bound, $(\frac{1}{1+\alpha})^{T+1}$, as described in Appendix A. T is the upper limit for event chain length. α is the normalization factor. Readers can set α based on their error requirements. We set α to 0.2 in our experiments with $T = 100$, achieving an approximation error less than 1e-8, which is relatively small compared to the average event root cause score of approximately 1e-2.

4.3.2 Causal Weights Assignment. This section corresponds to Lines 2-18 in Algorithm 1.

Given an NEG, G, with its event nodes V, we first assign a weight to each causal link in G with the overall event-causal graph parameters R_s and R_d, where R_d and R_s stores the causal weights of inter-service and intra-service causal links, respectively. The weight assigned to the causal link $a \rightarrow b$ is denoted as $E[(a, b)]$. We calculate the sum of causal link weights originating from event node v, denoted as $sum[v]$, and compute the α-biased mean value of these sums, denoted as $k = mean_v(sum[v]) \times \alpha$. For an event node v, $sum[v]$ and k indicate the probability weight of this event being caused by other nodes versus being its own root cause.

Next, we assign event importance scores to each node with the global event-causal graph's event importance score S and normalize them to obtain Q^0. Q^0 represents the probabilities of each event serving as the starting point of an event chain.

4.3.3 Graph Ranking. This section corresponds to Lines 19-29 in Algorithm 1. With an incident-specific event-causal graph, the root cause score of each event node is as follows:

Calculate Event Root Cause Score. *For a specific event in an incident, its root cause score is contributed by all the event chains pointing to it.*

Algorithm 1: CoE Inference with NEG

def CoE(G):
  Input : G, naive event-causal graph, whose event nodes are V;
  Parameter: R_d, inter-service causal weights; R_s, intra-service causal weights; S, event importance scores; T, the limit for event chain length; α, the normalization factor;
  Output : C, the root cause scores of V;
  /* Causal Weights Assignment: Constructing Incident-specific Event-causal Graph */
  $k = 0, C = EmptyArray, sum = EmptyArray$
  for *a in V* **do**
    for *b neighboring to a* **do**
      if *a, b in the same service* **then**
        $E[(a,b)] = R_s(a.event, b.event)$
      else
        $E[(a,b)] = R_d(a.event, b.event)$
      end
      $sum[a] = sum[a] + E[(a,b)]$
    end
    $k = k + sum[a]$
  end
  $k = k/size(V) * \alpha$
  for *a in V* **do**
    $Q^0[a] = S(a.event)$
  end
  $Q^0 = normalize(Q^0)$
  /* Graph Ranking: Calculate Root Cause Scores of the Events */
  for *i = 1 to T* **do**
    $Q^i = EmptyArray$
    for *a in V* **do**
      for *b neighboring to a* **do**
        $Q^i[b]$ += $Q^{i-1}[a] * E[(a,b)]/(sum[a]+k)$
      end
    end
    for *a in V* **do**
      $C[a]$ += $Q^i[a] * k/(sum[a]+k) * length\_bonus[i]$
    end
  end
  return C

We denote the set of event chains in an event-causal graph as $\{l_1, l_2, \ldots, l_n\}$ where l_i represents the i-th event chain of the event nodes. Denoting the root cause score of event v as $P(v)$ and the root cause score contributed by event chain l as $p(l)$. Then it can be expressed as:

$$P(v) = \sum_{l_i \text{ ends with } v} p(l_i) \tag{1}$$

The event chain contribution $p(l_i)$ in Eqn. (1) is computed from:

Calculate Event Chain Contribution. *For a given event chain, its contribution to the root cause score of the terminal node is determined by (1) the importance score of the starting node; (2) the joint probability of each event node being caused by its next event along the event-chain; (3) the probability of the terminal event not being caused by another event, and (4) the length bonus term of the event chain.*

Assuming an event chain l_i is $v_{d_1^{(i)}} \rightarrow v_{d_2^{(i)}} \rightarrow \ldots \rightarrow v_{d_{|l_i|}^{(i)}}$, where $v_j^{(i)}$ represents the index of the j-th event node in l_i and $|l_i|$ represents the event number of l_i. Then $p(l_i)$ is:

$$p(l_i) = S_{norm}(v_{d_1^{(i)}}) * \left[\prod_{j=1}^{|l_i|-1} p(v_{d_{j+1}^{(i)}} | v_{d_j^{(i)}}) \right] * Term(v_{d_{|l_i|}^{(i)}}) * LB(|l_i|) \tag{2}$$

Here, the first term, S_{norm}, is the normalized importance score of each event, corresponding to the normalized Q^0 in Alg. 1. In the second term, $p(v_{d_{j+1}^{(i)}} | v_{d_j^{(i)}})$ represents the probability of $v_{d_j^{(i)}}$ being *directly* caused by $v_{d_{j+1}^{(i)}}$, corresponding to the $E[(a,b)]/(sum[a]+k)$ in Alg. 1. The third term, $Term(v_{d_{|l_i|}^{(i)}})$, is referred to as *out-edge bonus term*, representing the probability of event $v_{d_{|l_i|}^{(i)}}$ being NOT caused by another event (thus being the root cause of itself), corresponding to the $k/(sum[a]+k)$ in Alg. 1. The last term, $LB(|l_i|)$, represents the *length bonus term* of the event chain of length $|l_i|$, corresponding to the $length\_bonus[i]$ in Alg. 1. LB is designed to reward the contribution of longer event chains. In practice, $LB[i] = \min(1.0, 0.01 * 2^{i-1})$ and it is at most 1 to avoid exponential explosion. In the following context, the out-edge bonus term and the length bonus term are referred to as *bonus terms*.

While Eqn. (1) and Eqn. (2) provide a clear definition for calculating event root cause scores, due to the complex causal relationship in microservice systems, it is not efficient to directly enumerate the event chains and accumulate their contributions to the root cause scores of their terminal nodes. Here, we introduce:

Efficient Calculation. *To compute the root cause scores of events, we can iteratively calculate the root cause scores contributed by event chains clustered by chain length.*

In the Appendix A, a comprehensive proof of this step is presented, accompanied by a theoretical analysis establishing an upper bound, $(\frac{1}{1+\alpha})^{T+1}$, on the approximation error when T is not infinite. T and α are parameters previously introduced in Section 4.3.1. The Alg. 1 follows this design, where $Q^i[v]$ is the bonus-terms-excluded contribution of all i-length event chains to event v

4.4 Training and Testing

The training process of the CoE is geared towards learning the learnable parameters, denoted as R_s, R_d, and S, in the overall event-causal graph, eliminating manual configuration.

Here, we introduce the training algorithm for CoE, as outlined in Alg. 2. The CoE model is trained to maximize the normalized root cause scores of the ground-truth root cause using the loss function $\mathcal{L}_C$ as depicted in Eqn. (3). ω represents the parameters including R_d, R_s, and S , trained using the Adam optimizer [14] along with an L2 penalty [20].

Algorithm 2: Train CoE

```
def TrainCoE(G,Y):
    Input  : training data 𝔾, NEGs; training labels Y, the
             ground-truth root cause events;
    Output : ω, parameters including R_d, R_s, and S;
    ω ← initialize(𝔾)
    for i = 1 to max_epoch do
        for (G, y) is a batch in (𝔾, Y) do
            ℒ_C = Σ_i CoE(G_i)[y_i]
            ω ← Adam(−∇_ω ℒ_C, ω)
        end
    end
    return ω
```

$$\mathcal{L}_C = -\mathbb{E}_{G,y}\,\mathrm{CoE}(G)[y] \tag{3}$$

In online mode, training can be regularly conducted, with incidents accumulating over time.

4.5 Integration of Human Knowledge

A significant advantage of the CoE model is that it is a white-box algorithm with an illustrative event-causal graph. This design ensures the model is transparent and intuitive for SREs, allowing them to easily comprehend and even modify the automatically learned parameters. Leveraging their expertise, SREs can fine-tune these parameters to enhance the precision of root cause identification.

For instance, as shown in Fig.3, based on historical data, CoE learns the causal link weights (R_s and R_d) and event importance scores (S). These parameters have clear physical meanings and align with the SREs' cognitive processes and observations during daily operations, ensuring ease of comprehension. A large causal link weight signifies that one event is highly probable to be triggered by another, whereas a high event importance score denotes that an event holds significance within the overall system, thereby necessitating its rigorous monitoring. Therefore, unlike models that require deep learning knowledge from SREs, CoE supports SREs in adjusting its parameters based on their experience, even without relevant parameter-tuning expertise, to improve the RCA output.

However, it is still necessary to note that while CoE allows SREs to easily modify model parameters based on their experience, the overall event-causal graph automatically learned by CoE is already outstanding without additional manual intervention. In the experiments in Section 5.2, we compared CoE with the overall event-causal graph learned entirely on its own against a completely manually configured event-causal graph. The results showed that the former still performs better. This section solely emphasizes that CoE supports further human knowledge integration based on the automatically learned causality.

5 EVALUATION

To evaluate the performance of CoE, we carry out a comprehensive set of experimental studies addressing the following research questions:

- **RQ1:Effectiveness**, how does the CoE improve the RCA accuracy by learning the overall event-causal graph compared with Groot and other baselines?
- **RQ2:Ablation Study**, what are the individual contributions of each component within CoE?
- **RQ3:Interpretability**, how does CoE achieve interpretability in real-world cases?

5.1 Experimental Design

5.1.1 Dataset. We evaluate CoE on datasets collected from a global top-5 *e*-commerce system with over 5,000 services in three data centers, serving 185 million active users. Our dataset contains events from 46 monitoring signals per service, aggregated from 800,000 monitoring signals. Our datasets consist of two subsets: Service dataset and Business dataset. The Service dataset includes service-level incidents (e.g., connection stacking issues), while the Business dataset covers customer/business impact incidents (e.g., failed interactions) related to business-to-business relationships. Both datasets contain 170 service incidents and 782 business incidents collected from Jan. 2020 to Apr. 2021, evenly split for training and testing through multiple rounds of random splitting.

5.1.2 Baselines. As previously mentioned, extracting the overall event-causal graph is crucial for interpretable SRE-friendly RCA algorithms in microservices systems. We primarily select some baseline algorithms to learn the event-causal graph from historical incidents and further perform root cause localization. Our baselines include:

- **Groot [10]**. Groot is an RCA algorithm with a manually configured event-causal graph.
- **Groot with NEG**. Groot runs on a naive event-causal graph without manual configuration.
- **PageRank**. Employing Approach 1 in Section 3.2.4, the PageRank is applied to the NEG to locate the root cause by updating the value of nodes by their adjacent nodes and out-edge numbers.
- **GCN**. GCN [15] has achieved remarkable results in graph embedding, graph classification, and node classification. RCA in the NEG is a binary node classification task.
- **GAT** [35]. [35] propose GAT to assign corresponding weights to different adjacent nodes. Unlike GCN, GAT predicts the importance of other neighboring nodes. Recent works like Eadro [16] and AlertRCA[43] use GAT to learn the dependency-aware status of the microservice system. We employ GAT with historical incidents to learn the event-causal graph.
- **GraphSAGE** [9]. GraphSAGE can be generalized to the unseen graph (different from the training graph).

Groot and PageRank represent existing RCA approaches. GCN, GraphSAGE, and GAT are the popular graph neural networks in node classification, graph classification, link prediction, and graph embedding [9, 15, 35].

Some other approaches do not support multi-modal root causes, as shown in Table 1. The traditional graph-based approaches (*e.g.,* CauseInfer [4] and Microscope [18]) do not perform well as introduced in [10, 42]. Other works include log-based methods [1, 7, 19, 21, 25, 44]. Some other methods [8] do not concentrate on event-level RCA and can not extract the causality between events.

Table 2: Performance of CoE and baselines, with *MEG* representing that the algorithm uses a Manual Event-causal Graph constructed based on rules defined by human experts. In this table, bold indicates the best performance.

	Model	MEG	Service		Business	
			Top-1	Top-3	Top-1	Top-3
Baselines	PageRank		16.1%	25.3%	1.2%	1.8%
	GraphSAGE		62.2%	78.1%	81.1%	93.7%
	GAT		12.2%	47.6%	60.5%	79.2%
	GCN		29.3%	57.3%	69.2%	85.3%
	Groot w/o *MEG*		17.1%	48.8%	23.2%	45.5%
	Groot	✓	74%	92%	81%	96%
Ours	CoE with *MEG*	✓	78.1%	93.9%	78.7%	95%
	CoE		**79.3%**	**98.8%**	**85.3%**	**96.6%**

We use GCN [15], GraphSAGE [9] and GAT [35] to solve RCA as a binary node classification task. BERT [5] is used in GCN, GraphSAGE, and GAT to transform the event name and service name into an embedding vector as the node's feature.

5.1.3 Metrics. We use top-1 and top-3 accuracy rates as metrics. The top-k accuracy is the ratio of incidents where the RCA score in the top-k sorts its ground-truth root cause.

5.1.4 Environment and efficiency. Our experiments run with an Intel(R) Core(TM) i9-9980HK CPU, an 11GB GTX1080Ti GPU, and 32GB memory. It takes about 45 minutes to train the CoE with all the incidents in each dataset. The average execution time and event number are 4.06s and 18.73 for the Business incidents and 2.16s and 14.70 for Service incidents. Its speed is close to Groot (2.98s for the Business dataset and 3.16s for the Service dataset), requiring minimal storage cost at just 52.06KB.

5.2 RQ1:Effectiveness

We evaluate CoE on the Service and Business datasets, compared with other approaches in Table 2. In the experiments, we use a minimum learning rate (4e-5) and few parameters in CoE to ensure reproducibility. As a result, the training is stable, and the top-1 and top-3 accuracy of CoE keep the same in five repeated experiments.

Among all the baselines, Groot [10] significantly outperforms most other baselines. However, it's worth noting that when an event-causal graph defined by manually configured rules is unavailable, Groot's performance is noticeably diminished (refer to "Groot without *MEG*"). We also explored the performance of widely-used graphical models, including GCN [15], GraphSAGE [9], and GAT [35] , which do not perform well when learning from the NEGs, as demonstrated in Table 2.

Compared to Groot, CoE not only eliminates the need for labor-intensive configuration but also boosts performance, with a 5.3% top-1 and 6.8% top-3 improvement in the Service dataset and a 4.3% top-1 improvement in the Business dataset.

Intriguingly, CoE not only surpasses Groot but also exceeds CoE with an event-causal graph completely configured by human experts (refer to "CoE with *MEG*"). This finding implies that CoE's ability to encapsulate causality surpasses that of human expertise.

Table 3: Length bonus and out-edge bonus ablation study.

	Service		Business	
	Top 1	Top 3	Top 1	Top 3
CoE	**79.3%**	**98.8%**	**85.3%**	**96.6%**
CoE w/o length bonus	79.3%	96.3%	83.2%	96.1%
CoE w/o out-edge bonus	75.6%	96.3%	83.4%	95.3%
CoE w/o both bonus terms	75.6%	93.9%	83.4%	95.3%

Table 4: Performance enhancements for each component integrated into a naive CoE sequentially.

	Service		Business	
	Top 1	Top 3	Top 1	Top 3
Naive CoE	31.7%	64.6%	71.7%	90.3%
+bonus terms	33.0%	67.1%	72.1%	90.5%
+learn S	51.2%	81.7%	82.1%	95%
+learn R_d	51.2%	86.6%	84.5%	95.5%
+learn R_s	**79.3%**	**98.8%**	**85.3%**	**96.6%**

We believe this enhancement primarily stems from the superiority of continuous types for casual weights, as opposed to the binary edges built according to the expert-configured manual rules. This methodology facilitates a more precise representation of event relationships.

5.3 RQ2: Ablation Study

In this part, an ablation study is performed on the following components in CoE:

- Out-edge bonus term, $Term$, and length-bonus term, LB
- Event importance scores, S
- Inter-service causal link weights, R_d
- Intra-service causal link weights, R_s

We initiated our analysis with an ablation study to investigate the impact of two essential terms, the out-edge bonus and length bonus terms, on CoE performance, as illustrated in Table 3. When we remove a term, we set this term as a constant 1. The experiments validate our intuition that bonus terms can lead to a more reasonable contribution to an event chain.

Subsequently, we conducted two additional sets of ablation experiments on CoE. In the first set, we incrementally introduced and evaluated the impact of each component on performance improvement, as elucidated in Table 4. In the second set, we systematically assessed the model's overall performance by removing each component individually, as outlined in Table 5. The results shed light on the significance of each component: The incorporation of component S led to a substantial enhancement in performance. Component R_d contributed to a modest improvement. Notably, component R_s played a pivotal role in elevating the performance of the Service dataset, suggesting that incidents within this dataset are typically associated with a single service.

Table 5: Impact of removing individual steps in CoE.

	Service		Business	
	Top 1	Top 3	Top 1	Top 3
CoE	**79.3%**	**98.8%**	**85.3%**	**96.6%**
CoE w/o learning S	75.6%	96.3%	84.5%	96.1%
CoE w/o learning R_d	78.1%	97.6%	84.5%	95.8%
CoE w/o learning R_s	51.2%	86.6%	84.5%	95.5%
CoE w/o bonus terms	75.6%	93.9%	83.4s%	95.3%

Table 6: Each step in CoE, where blanks represent 0 and $Q^i[j]$ represents bonus-terms-excluded root cause score contributed by all i-length event chains to node j.

Nodes	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
$Q^i[0]$	0.15				
$Q^i[1]$	0.15	0.107			
$Q^i[2]$	0.12	0.135	0.097		
$Q^i[3]$	0.12	0.005	0.005	0.004	
$Q^i[4]$	0.08	0.097	0.004	0.004	0.003
$Q^i[5]$	0.02	0.103	0.116	0.083	
$Q^i[6]$	0.08	0.018	0.092	0.104	0.074
$Q^i[7]$	0.08				
$Q^i[8]$	0.12	0.143			
$Q^i[9]$	0.08				

5.4 RQ3: Interpretability

In this section, we demonstrate how the CoE outputs can provide interpretability for SREs from three aspects in the real-life case in Fig. 2a. The causal link weights and event importance scores, as learned by CoE (depicted by the red and blue numbers in Fig. 2a), along with the bonus-terms-excluded root cause contribution of event chains of various lengths (as depicted in Table 6), collectively contribute to the interpretability of the model. This is exemplified in the case study presented in Fig. 2a, where these elements are used to answer three common questions SREs ask during fault diagnosis:

Firstly, how do the events influence each other? CoE provides insights into the propagation probabilities between events. By intelligently trimming redundant paths to eliminate spurious causality, CoE enhances clarity. For instance, the edge from Node 2 to Node 3 isn't outright severed but is assigned a weight of 0.07, significantly lower than the edge from Node 2 to Node 5. This weight assignment makes Node 2 more likely to be directly influenced by Node 5. The causal links between irrelevant events are automatically assigned zero weights after CoE training. Using continuous and differentiable causal link weights better represents the direct impacts between events than binary-style edges in the event-causal graphs defined by manually configured rules, facilitating a more comprehensive grasp of the system for SREs.

Secondly, how exactly does the model arrive at its conclusion? CoE not only computes the root cause score of nodes but also unveils the fault propagation process in each round, as shown in Table 6. For example, although events such as 7, 8, and 9 are assigned initial scores, their root cause scores diminish rapidly within one or two steps, indicating a limited influence range. Conversely, Event 6 grows increasingly suspicious in longer propagation paths, as evidenced by its score proportions of 42.4% for $i=2$, 42.5% for $i=3$, and a significant 96.1% for $i=4$. This illustrative propagation process derives the final root cause scores $C[j]$. For example, $C[2]=\sum_{i=1}^{2} Q^i[2]*Term(2)*LB(i)=0.00035$, $C[4]=0.00145$, and $C[6]=0.01214$. These detailed insights into propagation during each round empower SREs to better understand the entire process.

Thirdly, given an event chain, can SREs determine how suspicious this event chain is? CoE helps the SRE to evaluate the contributions of event chains. For the Fig. 2a case, $k=0.195$, SREs can examine the contributions of event chains as the following examples:

- $l_1=5\rightarrow 6$, $p(l_1)=0.00018$.
- $l_2=0\rightarrow 1\rightarrow 2\rightarrow 5$, $p(l_2)=0.000679$.
- $l_3=0\rightarrow 1\rightarrow 2\rightarrow 5\rightarrow 6$, $p(l_3)=0.00595$.
- $l_4=0\rightarrow 1\rightarrow 2\rightarrow 3\rightarrow 4$, $p(l_4)=0.00237$.
- $l_5=0\rightarrow 1\rightarrow 8$, $p(l_5)=0$.
- $l_6=7\rightarrow 8$, $p(l_6)=0.000722$.

By comparing $p(l_1)$ with $p(l_2)$, SREs can discern that the presence of Event 0 contributes more significantly to the root cause score of Event 6 than the presence of Event 5 does. Despite l_2 and l_3 originating from the same node, the fact that $p(l_2)<p(l_3)$ suggests that Event 0 is more likely to designate Event 6 as the root cause rather than Event 5. Furthermore, a comparison between $p(l_3)$ and $p(l_4)$ allows SREs to infer that Event 0 is more likely to be propagated from Node 6 along l_3, as opposed to from Event 4 along l_4. $p(l_5)$ is zero, indicating that the fault in Event 0 is irrelevant to Event 8. In contrast, a nonzero $p(l_6)$ implies that Event 8 is more likely to cause the issue in Event 7.

These three aspects of CoE enable SREs to understand the incident better, encompassing the root cause event and the fault propagation details.

6 CONCLUSION

In this paper, we present a white-box algorithm, CoE, to automatically learn the event-causal graph and accurately detect root causes while ensuring interpretability and enabling SREs to integrate human knowledge. CoE eliminates the need for time-consuming manual configuration by automatically learning the parameters in an overall event-causal graph. Furthermore, it helps SREs comprehend fault propagation and event chains within the event-causal graphs, guaranteeing interpretability. Our evaluations, conducted on two datasets encompassing 952 real-life incidents sourced from SRE remediation records, illustrate CoE's superiority over baseline methods. Our ablation study provides insight into the effectiveness of each component of the CoE algorithm. In our future work, we plan to delve further into causality relationships, exploring avenues such as unsupervised or active learning techniques.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China under grants 62072264 and 62202445, the Beijing National Research Center for Information Science and Technology (BNRist) key projects, and the National Key Research and Development Program of China under grant 2022YFB2901800.

REFERENCES

[1] Marcos K. Aguilera, Jeffrey C. Mogul, Janet L. Wiener, Patrick Reynolds, and Athicha Muthitacharoen. 2003. Performance debugging for distributed systems of black boxes. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (Bolton Landing, NY, USA) *(SOSP '03)*. Association for Computing Machinery, New York, NY, USA, 74–89. https://doi.org/10.1145/945445.945454

[2] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. 2016. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. *IEEE Software* 33, 3 (2016), 42–52. https://doi.org/10.1109/MS.2016.64

[3] Álvaro Brandón, Marc Solé, Alberto Huélamo, David Solans, María S Pérez, and Victor Muntés-Mulero. 2020. Graph-based root cause analysis for service-oriented and microservice architectures. *Journal of Systems and Software* 159 (2020), 110432. https://doi.org/10.1016/j.jss.2019.110432

[4] Pengfei Chen, Yong Qi, Pengfei Zheng, and Di Hou. 2014. CauseInfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. 1887–1895. https://doi.org/10.1109/INFOCOM.2014.6848128

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[6] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. 2017. Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering* (2017), 195–216. https://doi.org/10.1007/978-3-319-67425-4_12

[7] Yu Gan, Yanqi Zhang, Kelvin Hu, Dailun Cheng, Yuan He, Meghna Pancholi, and Christina Delimitrou. 2019. 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* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 19–33. https://doi.org/10.1145/3297858.3304004

[8] Xiaofeng Guo, Xin Peng, Hanzhang Wang, Wanxue Li, Huai Jiang, Dan Ding, Tao Xie, and Liangfei Su. 2020. Graph-based trace analysis for microservice architecture understanding and problem diagnosis. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1387–1397. https://doi.org/10.1145/3368089.3417066

[9] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf

[10] Wang Hanzhang, Wu Zhengkai, Jiang Huai, Huang Yichao, Wang Jiamu, Kopru Selcuk, and Xie Tao. 2021. 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)*. 419–429. https://doi.org/10.1109/ASE51524.2021.9678708

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

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

[13] 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 *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1410–1420. https://doi.org/10.1145/3368089.3417054

[14] Diederik Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*. San Diega, CA, USA.

[15] N. Thomas Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. *international conference on learning representations* (2017). https://openreview.net/forum?id=SJU4ayYgl

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

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

[18] JinJin Lin, Pengfei Chen, and Zibin Zheng. 2018. Microscope: Pinpoint performance issues with causal graphs in micro-service environments. In *International Conference on Service-Oriented Computing*. Springer, 3–20.

[19] 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 *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 338–347. https://doi.org/10.1109/ICSE-SEIP52600.2021.00043

[20] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Bkg6RiCqY7

[21] Siyang Lu, BingBing Rao, Xiang Wei, Byungchul Tak, Long Wang, and Liqiang Wang. 2017. Log-based Abnormal Task Detection and Root Cause Analysis for Spark. In *2017 IEEE International Conference on Web Services (ICWS)*. 389–396. https://doi.org/10.1109/ICWS.2017.135

[22] Meng Ma, Weilan Lin, Disheng Pan, and Ping Wang. 2019. MS-Rank: Multi-Metric and Self-Adaptive Root Cause Diagnosis for Microservice Applications. In *2019 IEEE International Conference on Web Services (ICWS)*. 60–67. https://doi.org/10.1109/ICWS.2019.00022

[23] Jonathan Mace, Ryan Roelke, and Rodrigo Fonseca. 2018. Pivot Tracing: Dynamic Causal Monitoring for Distributed Systems. *ACM Trans. Comput. Syst.* 35, 4, Article 11 (dec 2018), 28 pages. https://doi.org/10.1145/3208104

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

[25] Vinod Nair, Ameya Raul, Shwetabh Khanduja, Vikas Bahirwani, Qihong Shao, Sundararajan Sellamanickam, Sathiya Keerthi, Steve Herbert, and Sudheer Dhulipalla. 2015. Learning a Hierarchical Monitoring System for Detecting and Diagnosing Service Issues. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Sydney, NSW, Australia) *(KDD '15)*. Association for Computing Machinery, New York, NY, USA, 2029–2038. https://doi.org/10.1145/2783258.2788624

[26] Sam Newman. 2021. *Building microservices*. " O'Reilly Media, Inc.".

[27] Hiep Nguyen, Zhiming Shen, Yongmin Tan, and Xiaohui Gu. 2013. FChain: Toward Black-Box Online Fault Localization for Cloud Systems. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*. 21–30. https://doi.org/10.1109/ICDCS.2013.26

[28] Juan Qiu, Qingfeng Du, Kanglin Yin, Shuang-Li Zhang, and Chongshu Qian. 2020. A causality mining and knowledge graph based method of root cause diagnosis for performance anomaly in cloud applications. *Applied Sciences* 10, 6 (2020), 2166. https://doi.org/10.3390/app10062166

[29] Chris Richardson. 2018. *Microservices patterns: with examples in Java*. Simon and Schuster.

[30] Jacopo Soldani and Antonio Brogi. 2022. Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey. *ACM Comput. Surv.* 55, 3, Article 59 (feb 2022), 39 pages. https://doi.org/10.1145/3501297

[31] Marc Solé, Victor Muntés-Mulero, Annie Ibrahim Rana, and Giovani Estrada. 2017. Survey on models and techniques for root-cause analysis. *arXiv preprint arXiv:1701.08546* (2017). https://doi.org/10.48550/arXiv.1701.08546

[32] Yongqian Sun, Youjian Zhao, Ya Su, Dapeng Liu, Xiaohui Nie, Yuan Meng, Shiwen Cheng, Dan Pei, Shenglin Zhang, Xianping Qu, and Xuanyou Guo. 2018. HotSpot: Anomaly Localization for Additive KPIs With Multi-Dimensional Attributes. *IEEE Access* 6 (2018), 10909–10923. https://doi.org/10.1109/ACCESS.2018.2804764

[33] Liang Tang, Tao Li, Florian Pinel, Larisa Shwartz, and Genady Grabarnik. 2012. Optimizing system monitoring configurations for non-actionable alerts. In *2012 IEEE Network Operations and Management Symposium*. 34–42. https://doi.org/10.1109/NOMS.2012.6211880

[34] Jörg Thalheim, Antonio Rodrigues, Istemi Ekin Akkus, Pramod Bhatotia, Ruichuan Chen, Bimal Viswanath, Lei Jiao, and Christof Fetzer. 2017. Sieve:

actionable insights from monitored metrics in distributed systems. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference* (Las Vegas, Nevada) *(Middleware '17)*. Association for Computing Machinery, New York, NY, USA, 14–27. https://doi.org/10.1145/3135974.3135977

[35] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *ICLR* (2018). https://openreview.net/forum?id=rJXMpikCZ

[36] Jianping Weng, Jessie Hui Wang, Jiahai Yang, and Yang Yang. 2018. Root Cause Analysis of Anomalies of Multitier Services in Public Clouds. *IEEE/ACM Transactions on Networking* 26, 4 (2018), 1646–1659. https://doi.org/10.1109/TNET.2018.2843805

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

[38] Zhe Xie, Changhua Pei, Wanxue Li, Huai Jiang, Liangfei Su, Jianhui Li, Gaogang Xie, and Dan Pei. 2023. From Point-wise to Group-wise: A Fast and Accurate Microservice Trace Anomaly Detection Approach *(ESEC/FSE 2023)*. Association for Computing Machinery, New York, NY, USA, 1739–1749. https://doi.org/10.1145/3611643.3613861

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

[40] Jingmin Xu, Yuan Wang, Pengfei Chen, and Ping Wang. 2017. Lightweight and Adaptive Service API Performance Monitoring in Highly Dynamic Cloud Environment. In *2017 IEEE International Conference on Services Computing (SCC)*. 35–43. https://doi.org/10.1109/SCC.2017.80

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

[42] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. 2023. Nezha: Interpretable Fine-Grained Root Causes Analysis for Microservices on Multi-modal Observability Data. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2023)*. Association for Computing Machinery, New York, NY, USA, 553–565. https://doi.org/10.1145/3611643.3616249 event-place: , San Francisco, CA, USA,.

[43] Zhaoyang Yu, Qianyu Ouyang, Changhua Pei, Xin Wang, Wenxiao Chen, Liangfei Su, Huai Jiang, Xuanrun Wang, Jianhui Li, and Dan Pei. 2024. Causality Enhanced Graph Representation Learning for Alert-Based Root Cause Analysis. In *Proceedings of the 24th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*.

[44] Hamzeh Zawawy, Kostas Kontogiannis, and John Mylopoulos. 2010. Log filtering and interpretation for root cause analysis. In *2010 IEEE International Conference on Software Maintenance*. 1–5. https://doi.org/10.1109/ICSM.2010.5609556

[45] Nengwen Zhao, Panshi Jin, Lixin Wang, Xiaoqin Yang, Rong Liu, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Automatically and Adaptively Identifying Severe Alerts for Online Service Systems. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. 2420–2429. https://doi.org/10.1109/INFOCOM41043.2020.9155219

Received 2024-02-08; accepted 2024-04-18