Root Cause Analysis for Microservice Systems via Hierarchical Reinforcement Learning from Human Feedback

Lu Wang
Microsoft
Beijing, China
wlu@microsoft.com

Chaoyun Zhang
Microsoft
Beijing, China
chaoyun.zhang@microsoft.com

Ruomeng Ding
Georgia Tech
Atlanta, United States
rmding@gatech.edu

Yong Xu
Microsoft
Beijing, China
yong.xu@microsoft.com

Qihang Chen
Peking University
Beijing, China
v-qihchen@microsoft.com

Wentao Zou
Microsoft
Suzhou, China
wezo@microsoft.com

Qingjun Chen
Microsoft
Suzhou, China
qingche@microsoft.com

Meng Zhang
Microsoft
Suzhou, China
zhangmeng@microsoft.com

Xuedong Gao
Microsoft
Suzhou, China
xuedong.gao@microsoft.com

Hao Fan
Microsoft
Suzhou, China
haofan@microsoft.com

Saravan Rajmohan
Microsoft 365
Seattle, United States
saravan@microsoft.com

Qingwei Lin*
Microsoft
Beijing, China
qlin@microsoft.com

Dongmei Zhang
Microsoft
Beijing, China
dongmeiz@microsoft.com

ABSTRACT

In microservice systems, the identification of root causes of anomalies is imperative for service reliability and business impact. This process is typically divided into two phases: *(i)* constructing a service dependency graph that outlines the sequence and structure of system components that are invoked, and *(ii)* localizing the root cause components using the graph, traces, logs, and Key Performance Indicators (KPIs) such as latency. However, both phases are not straightforward due to the highly dynamic and complex nature of the system, particularly in large-scale commercial architectures like Microsoft Exchange.

In this paper, we propose a new framework that employs **H**ierarchical **R**einforcement **L**earning from **H**uman **F**eedback (HRLHF) to address these challenges. Our framework leverages the static topology of the microservice system and efficiently employs the feedback of engineers to reduce uncertainty in the discovery of the service dependency graph. The framework utilizes reinforcement learning to reduce the number of queries required from $O(N^2)$ to $O(1)$, enabling the construction of the dependency graph with high accuracy and minimal human effort. Additionally, we extend the discovered dependency graphs to window causal graphs that capture the characteristics of time series over a specified time period, resulting in improved root cause analysis accuracy and robustness. Evaluations on both real datasets from Microsoft Exchange and synthetic datasets with injected anomalies demonstrate superior performance on various metrics compared to state-of-the-art methods. It is worth mentioning that, our framework has been integrated as a crucial component in Microsoft M365 Exchange service.

*Corresponding author.

CCS CONCEPTS

• **Computing methodologies → Sequential decision making**.

KEYWORDS

Root cause analysis, Causal discovery, Reinforcement Learning from Human Feedback

ACM Reference Format:
Lu Wang, Chaoyun Zhang, Ruomeng Ding, Yong Xu, Qihang Chen, Wentao Zou, Qingjun Chen, Meng Zhang, Xuedong Gao, Hao Fan, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2023. Root Cause Analysis for Microservice Systems via Hierarchical Reinforcement Learning from Human Feedback. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3580305.3599934

1 INTRODUCTION

The utilization of microservices architecture in cloud systems has become a popular approach in modern production systems [1]. This approach involves breaking down a large system into smaller, independent components that communicate with one another to facilitate agile development and deployment [2]. For example, hundreds of microservices are involved for email delivery in Microsoft Exchange. Unexpected latency at any of the component may lead to a huge number of slow email deliveries and impact customer experience, which can further result in substantial financial loss [3]. To guarantee high quality cloud services, site reliability engineers (SREs) leverage various system data (*i.e.*, key performance indicators, running logs, trace, *etc.*) to develop automatic and intelligent solutions for incident detection and root cause analysis (RCA) [4, 5] to enable prompt mitigation and minimize the impact on business.

However, The identification of the root cause of an incident in large microservice systems is a crucial, yet challenging task due to the complex interactions and interdependencies among the components. This challenge is exacerbated by the increasing size and complexity of these systems. Research has shown that without the use of automated tools, it takes hours on average to identify the root cause of a failure, making the process of RCA time-consuming and challenging [6, 7]. The situation is further complicated in large systems such as Microsoft Exchange, where the spread of latency is influenced by multiple factors, including the scale of the infrastructure, automatic scaling, dynamic load balancing, system updates, and security measures [8]. In such systems, the RCA process becomes even more intricate, making it imperative to have effective and efficient methods for identifying the root cause of incidents.

Prior studies on troubleshooting techniques in microservice systems typically employ a two-stage approach for RCA during incidents [8–11]. The first stage involves constructing a service dependency graph to outline the sequence and structure of the system components that are invoked. The second stage entails localizing the root cause components using traces, logs, and Key Performance Indicators (KPIs) such as latency. However, these practice [8–11] bear shortcomings in the ability to effectively recognize temporal causality in dynamic systems. This is due to the fact that previous methods either assume a constant causal mechanism within a set time frame regardless of the system state, or they perform diagnosis on static causal graphs while ignoring the crucial attributes related to the dynamic evolution of latency. These limitations hinder the RCA from accurately identifying temporal causality, such as the sequence of anomaly propagation and the buildup of latency caused by congestion. In Fig. 1, a partial dependency graph of microservices in the Microsoft Exchange system at two different time spans (t_1 and t_2) illustrates this issue. During time t_1, the services were operating normally, whereas at time t_2 anomalies were observed. In this case, 7 new dependencies were established due to system congestion and load balancing, resulting in a completely different graph from t_1. Static RCA approaches that fail to take into account the dynamic characteristics of such systems are often unsuccessful in localizing the true root cause services, as the dependency graph and causal mechanism may have changed during the anomaly period.

In addition, due to the complexity and hidden relationships among microservices, it can be challenging to accurately identify the complete causal graph solely from observational latency data [11–13]. This can result in an uncertain and inaccurate dependency graph, negatively impacting the performance of the RCA process. However, incorporating human feedback from experienced SREs, who are familiar with the system architecture and diagnosis, can help to reduce this uncertainty and improve the accuracy of the

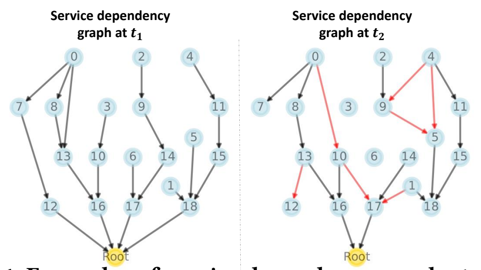
Service dependency graph at t_1
Service dependency graph at t_2
0
2
4
7
8
3
9
11
5
13
10
6
14
15
1
12
16
17
18
Root

Figure 1: Examples of service dependency graph at different time for Microsoft Exchange microservice system. New dependency edges at t_2 are highlighted with red.

dependency graph [14, 15]. The idea of human-in-the-loop training, as applied in large language models such as ChatGPT [16] and Constitutional AI [17], has been demonstrated to be effective in improving model performance. We leverage this idea to incorporate human feedback at the stage of dependency discovery and utilize reinforcement learning to minimize the required human queries from $O(N^2)$ to $O(1)$. This allows for the construction of an accurate causal graph between services with minimal human effort.

We treat the dependency graph discovered in the first stage as the causal relationships between services and adding a temporal dimension to enhance the accuracy of the RCA process. The expanded causal graph incorporates the Markov property and Granger causality in service latency time series, enabling a more representative representation of system dynamics. By incorporating the Causal-RCA method presented in [8], HRLHF leads to a more precise, robust attribution of anomalies to individual nodes in the microservice system over state-of-the-art approaches. We discuss the details of the methodology in Section 5. We summarize the contribution of the paper as follows:

- We propose an **H**ierarchical **R**einforcement **L**earning from **H**uman **F**eedback (HRLHF) architecture to utilize human expertise from experienced Site Reliability Engineers (SREs) during the first stage of RCA, which involves discovering service dependencies, to reduce uncertainty and errors in the process.
- We combine human-in-the-loop training and reinforcement learning to minimize the need for human feedback, reducing the query complexity from $O(N^2)$ to $O(1)$, where the reward function is learned form the human feedbacks.
- We extend the dependency graph constructed in stage one to incorporate a temporal dimension, capturing the evolution of latency and dynamics of the system, leading to more accurate and robust RCA outcomes.
- Our proposed framework was tested on both synthetic data and real data traces collected in the Microsoft Exchange system, showing superior performance compared to state-of-the-art RCA approaches for both causal structure discovery and RCA outcomes.
- The framework has been integrated as a crucial component in Microsoft M365 Exchange, contributing to the improvement of the system's reliability.

To the best of our knowledge, this is the first time human feedback has been incorporated in the causal discovery process of RCA, resulting in remarkable performance improvements.

2 RELATED WORK

2.1 Causal Discovery

The identification of causal relationships is a crucial undertaking that intersects the fields of causality and machine learning. Two of the most commonly employed techniques for this task are the PC algorithm [18] and Fast Causal Inference (FCI) [19]. Zheng *et al.*, propose a generic optimization approach to learn the causal structure from nonparametric data [20, 21]. Work in [22] treats the causal discovery as a Markov decision process and searches the causal graph from data using deep reinforcement learning.

In the context of time series, the Granger causality approach, initially put forth in [23], has gained widespread usage for the examination of multivariate temporal [24–27] causal relationships through evaluating the contribution of one time-series in predicting another. The method of Granger causality has been further developed to accommodate nonlinear [28] and non-stationary [29] circumstances by substituting the prediction model with more advanced machine learning models. Building upon this idea, Löwe *et al.*, [30] expanded this concept to cover causal discovery in multiple samples with similar dynamics but different causal relationships. This improves the causal discovery performance when with added noise and hidden confounding.

Despite the success of these studies in uncovering causal relationships in complex time-series data, they do not incorporate human feedback. This is particularly significant in the context of microservice RCA, as engineers may possess valuable expertise regarding the causal relationships in the microservice architecture.

2.2 Root Cause Analysis

The process of identifying the underlying cause of a problem, commonly known as RCA, typically builds upon the detection of anomalies within a system. When a system failure occurs, RCA is implemented to diagnose the issue in the systems. These methods commonly rely heavily on the utilization of a system dependency graph as a guide during the analysis process (*e.g.*, [10, 31, 32]) and the precision of the can significantly affect the accuracy of the RCA.

Various approaches have been proposed to localize the root cause of failures in microservice systems. Meng *et al.*, proposed MicroCause in their work, which combines causal relationships, anomaly information, and the metric priority of the system to localize the root cause of a failure [9]. Dycause, proposed by Pan *et al.*, in [33], is an efficient and scalable method that performs user-space diagnosis for microservice kernel failures. Additionally, Ikram *et al.*, in [11] proposed a framework that utilizes a causal approach to discover the root cause of a failure by treating it as an intervention.. Our RCA methods differ from the aforementioned approaches, by comparing the temporal causal mechanism between the normal and abnormal periods in a microservice system. This approach results in better RCA accuracy and allows for the evaluation of anomaly attribution for each sub-component.

3 BACKGROUND & PROBLEM FORMULATION

3.1 Microsoft Exchange System

Microsoft Exchange is a cloud-based email delivery system whose goal is to provide a quick and reliable message routing and delivery service. One of the key performance metrics for the system is email delivery latency, which is the core service level agreement promised to customers. The delivery of messages in the cloud system is a complex process that involves hundreds of components from various servers and services, and involves the processing of tens of billions of emails each day. Each email request produces a trace that captures its invocation dependencies [4]. This trace is represented as a directed tree structure, where each node is a microservice or function that is invoked and the edges represent their interdependent relationships. The trace records relevant information, such as structural attributes and temporal information, for later analysis.

Each email request incurs an end-to-end latency that accumulates the delays from all of the microservices in the trace. If this latency exceeds the established service level agreement (SLA), it can result in a high number of slow email deliveries, causing dissatisfaction among customers and leading to escalation. In these context, RCA is crucial in localizing the cause of the issue, so as to allow for prompt mitigation actions and minimize the impact on business operations.

3.2 Root Causal Localization

The localization of root causes for anomalies in microservice systems is divided into two phases, namely *(i)* causal discovery and *(ii)* root cause analysis. In this work, we aim to monitor the latency of each service in a microservice system and to localize the root cause of anomalies by discovering the service dependencies. To this end, we define the latency evolution of the service set as $\mathbf{D^H} = (D_1^H, D_2^H, \cdots, D_m^H)$, where D^H represents the latency metric of each service in the time range from $t - H$ to t. We propose a three-step approach that combines causal discovery and reinforcement learning to construct a dependency graph $\mathcal{G}$. More details are presented in Section 4.

In the second stage of root cause localization, we identify root cause node for the anomaly using the causal graph $\mathcal{G}$ obtained from stage one. Let $\mathbf{X} = (X_1, X_2, \cdots, X_n)$ represent the set of nodes in the graph $\mathcal{G}$, where each node collects its own system metric, such as latency, within a designated time window, *i.e.*, $X_i = (X_i^{t-H}, \cdots, X_i^{t-1}, X_i^t)$. The goal of the RCA process is to determine the most likely subset of nodes $\mathbf{R} \subseteq (1, \cdots, n)$ that are the root cause of the anomaly. These root cause nodes can then be examined by engineers to implement corrective measures. We present more details in Section 5.

4 CAUSAL DISCOVERY VIA HRLHF

As illustrated in Figure 2, the proposed Hierarchical Reinforcement Learning with Human Feedback (HRLHF) approach for causal discovery consists of three key stages:

(1) We use the available information on the static system structure to generate an initial service dependency graph $\mathcal{G}'$ using causal discovery algorithms such as PC [18]. We then train a base causal discovery policy π^l using supervised learning.
(2) We implement an efficient human feedback collection framework through reinforcement learning with a learned reward function r_θ to minimize the human feedback required to $O(1)$.
(3) We optimize the causal discovery framework using the learned reward function to obtain a refined dependency graph $\mathcal{G}$.

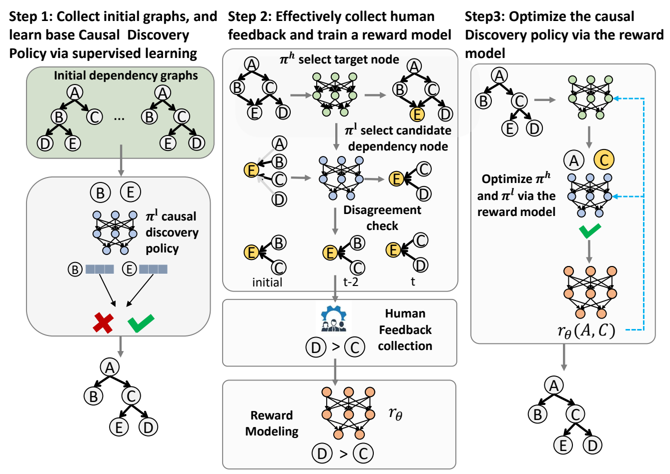
Step 1: Collect initial graphs, and learn base Causal Discovery Policy via supervised learning
Initial dependency graphs
π^l causal discovery policy
Step 2: Effectively collect human feedback and train a reward model
π^h select target node
π^l select candidate dependency node
Disagreement check
initial
t-2
t
Human Feedback collection
D > C
Reward Modeling
r_θ
Step3: Optimize the causal Discovery policy via the reward model
Optimize π^h and π^l via the reward model
$r_\theta(A, C)$

Figure 2: Overview of Causal Discovery Framework.

4.1 Learning Base Causal Discovery Policy

In practical applications, the set of topology graphs $\mathcal{G}'$ can be extracted from either the static system structure provided by the product team or from graphs generated through causal discovery methods. Let $G = (X, \mathcal{E})$ be a directed causal graph, where X represents the set of nodes and $\mathcal{E} \subseteq N \times N$ represents the set of observed edges. A node pair (x_i, x_j) is considered to have a causal relationship (positive sample pair) if there is an edge linking x_i to x_j, otherwise, it is considered a negative sample pair. To warm start the causal discovery policy π^l, we utilize a Graph Neural Network (GNN) to learn an encoder $f : X \longrightarrow \mathbb{R}^d$ that maps a node $x \in X$ to a point on a d-dimensional hypersphere. The policy π^l is then initialized as $\pi^l = \sigma(f(x_i)^\top f(x_j))$. We then apply contrastive learning for similarity learning to effectively train π^l synchronously with the initially obtained graphs. The objective for training π^l through contrastive learning is defined as follows:

$$\mathcal{L}_{base} = \mathbb{E}_{x_j \sim p}\Big[\mathbb{E}_{\sim p^+_{x_j}} - \log \sigma\big(f(x_i)^\top f(x_j)\big) - N\mathbb{E}_{x_k \sim p^-_{x_j}} \log \sigma\big(-f(x_k)^\top f(x_j)\big)\Big]. \quad (1)$$

For a specific target node x_j, depending on whether the neighbour node x_i is linked to x_j or not, we use $p^+_v(x_i)$ and $p^-_v(x_k)$ to represent the positive and negative distribution of node pairs associated with target node x_i. In addition, we also add the general causal discovery loss as an additional loss term.

4.2 Efficient Human Feedback Collection

The use of human feedback in reinforcement learning has been shown to improve the performance of large scale model training by aligning the machine's intelligence with that of human experts. However, the main challenge of collecting human feedback is its computational complexity, which is $O(N^2)$ in constructing the full dependency graph, where N denotes the number of nodes, making it an overwhelming task for engineers. To address this, we propose a novel method called HRLHF (Hierarchical Reinforcement Learning with Human Feedback), which decomposes the complexity of the human feedback collection process by formulating it as a Markov Decision Process (MDP).

In HRLHF, we utilize a hierarchical reinforcement learning policy to achieve this decomposition. First, a high-level policy is used to determine the target nodes, while a low-level policy is used to determine the neighborhood of each target node. The low-level policy, π^l, is pretrained in step 1. Our solution is mathematically equivalent to solving the submodular maximizing problem and is bounded by $(1 - \frac{1}{e})R(\mathcal{N}(v)^*)$, where$\mathcal{N}(v)^*$ denotes the optimal neighborhood set. Finally, the low-level policy is used to determine the neighborhood of each target node.

4.2.1 *MDP Formulation of Human Feedback Collection.*

Specifically, we design a sequential human feedback collection process which is also a graph topology optimization process which is formulated as a markov decision process (MDP) $M = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$. Based on this process, we design a hierarchical policy to select node pairs to query human's labels. We first learn a high-level policy π^h to determine the target nodes x_t for human feedback collection $x_t \leftarrow \pi^h(X)$. Next, we determine the neighborhood nodes which have causality with the target node via a low-level policy $\hat{\mathcal{N}}(x_t) \leftarrow \pi^l(\mathcal{N}(x_t))$. The detailed MDP formulation are as follows:

- **State** ($\mathcal{S}$): The state of π^h $\bar{s} \in \mathcal{S}$ is represented by the tuple $\{\bar{G}_t, x_t\}$, where $\bar{G}_t$ is the modified graph and x_t is the target node. The state of the low-level policy is $s_t = [f(x_t), f(x_j)]$, where x_j is the j-th candidate neighborhood of x_t.
- **Action** ($\mathcal{A}$): The action of the high-level policy is $\bar{a}_t \in \{0, 1\}$, which indicates whether select x_t as a target node. Given an order of the neighbors $x_j \in \{x_1, ..., x_{|\mathcal{N}(x_t)|}\}$ of node x_t, the action of the low-level policy is $a_t = \{0, 1\}$, which indicates whether consider x_j as a neighborhood.
- **Reward** (R): The reward function is learned from the feedback of human experts. The reward of the low-level policy $r_\theta(x_t, x_j)$ is the preference of human to annotate that x_t and x_j have causation. The reward function of high-level policy $\bar{r}(\bar{s}_t, \bar{a}_t)$ is the cumulative reward of r for all neighbors of x_t.

4.2.2 *Collect Human Feedback.*

Once the hierarchical policy selects out K candidate neighbor nodes for x_t, we ask human to rank $\binom{2}{K}$ neighbors.

Human Feedback The human experts are given a set of tuples (x_t, x_j, x_k) and the timeseries value of these nodes. The human then indicates which node (x_j or x_k) they prefer as the cause of x_t. Finally, the human judgments are recorded in a database D of triples (x_t, x_j, x_k) which indicates that human prefers x_j than x_k.

$$Var_{\pi^l} = (\pi^l(x_t, x_j) - \pi^l(x_t, x_k)) - E[\pi^l(x_t, x_j) - \pi^l(x_t, x_k)]. \quad (2)$$

If $Var_{\pi^l} \geq \delta$, our model will trigger the human feedback till the total amount of feedback exceeds K.

In this work, we have designed a human feedback collection framework that offers two significant benefits. Firstly, the framework reduces the complexity of human feedback labeling from exponential to linear by using a hierarchical policy. Without this framework, collecting human feedback would have required $O(N^2)$ to select the causal pairs, however, with the hierarchical policy and the introduction of a disagreement threshold, the complexity is reduced to $O(K)$, where K is the number of selected neighbors. As

K is a constant hyperparameter in our experiment setting, the complexity of collecting human feedback can be considered to be $O(1)$. Second, instead of using real causation between services as a reward function, we solicit expert engineers to rank the candidate neighbors. This approach is chosen because assigning an absolute value to causation can vary greatly between experts, whereas providing a ranking is more consistent.

4.2.3 Reward Modeling. After obtaining the feedback from human, we train the reward function model with a cross entropy loss as follows: $L_{ce} = -\frac{1}{\binom{2}{K}} E_{(x_t,x_j,x_k)\sim D}[\log(\sigma(r_\theta(x_t,x_j) - r_\theta(x_t,x_k))]$, where $r_\theta(x_t,x_j)$ is the scalar output of the reward model for target node x_t and candidate neighbor x_j with parameters θ, x_j is the preferred completion out of the pair of x_j and x_k, and D is the dataset of human comparisons.

4.3 Causal Discovery Policy Optimization

After obtaining the reward model, we fine-tune the supervised causal discovery policy π as well as the high-level policy π^h to optimize the reward. We formulate the causal graph topology optimization problem as an MDP and propose a hierarchical reinforcement learning (HRL) algorithm, to decompose the computational complexity. This model includes two phases: *topology exploration and optimization* and *representation learning*. It first uses a HRL framework to optimize the observed graph, and then apply a GNN model to learn node representations with respect to the down-stream tasks. The core ideas to decompose the quadratic edge searching space are that *(i)* we use the high-level policy for target node selection (for structure exploration), *(ii)* candidate neighbor selection and ordering via a greedy algorithm, *(iii)* candidate neighbor action exploration.

4.3.1 Overview of Graph Topology Optimization for GNN. With the definitions of the graph topology optimization environment, we propose a hierarchical reinforcement learning algorithm, referred to as HRLHF, for graph topology optimization. The algorithm consists of three phases: target node determination, neighborhood selection, and representation learning. In the first phase, the algorithm learns a high-level policy π_h to determine the target nodes based on the global state. In the second phase, a sub-policy is learned to select the neighborhood of the target node from a set of ordered candidate neighborhoods. Finally, in the representation learning phase, the node representations h_v are obtained by aggregating information from the selected neighborhood set $\hat{\mathcal{N}}(v)$.

4.3.2 High-level Topology Optimization. High-level policy focuses on determine the candidate target nodes from a global scale. We utilize Q-learning to learn the high-level policy π_h. With the Bellman optimality, the optimize Q function of the high-level policy is defined as $Q^*(\bar{G}_t, x_t) = r_h(\bar{G}_t, x_t) + \gamma \max_{x_t} Q^*(s_{t+1}, x_{t+1})$, where γ is the discount factor to balance the action value of current reward and future rewards. The high-level policy can be determined by a greedy policy $\pi_h(x_t|\bar{G}_t; Q^*) = \arg\max_{x_j} Q(\bar{G}_t, x_t)$. To effectively calculate the reward of π_h, we consider the action of π_h as a batch of selected target nodes, after optimizing the topology of such target nodes, we obtain the reward from the reward model as follows: $\bar{r}(\bar{G}_t, v_t) = \sum_{x_j \in \hat{\mathcal{N}}(x_t)} (r_\theta(x_t, x_j))$, where $\hat{\mathcal{N}}(x_t)$ are neighborhoods selected by the low-level policy.

4.3.3 Determine Candidate Neighborhood nodes. The complexity of determining the target nodes is $O(N)$. To further decompose the complexity, we utilize a greedy algorithm to filter the candidate neighborhoods of each target nodes. We design a score function to calculate the probability of an node to be selected by a candidate node. This function can be any metric that used to measure the relationship between the target node and the candidate neighborhood nodes. Here we directly use the Cosine similarity to measure the two nodes' representation, which is determined as : $g_j = \cos(f(x_t), f(x_j))$, where x_j is the j-th neighborhood in the neighborhood set $\mathcal{N}(v)$ with a random order. Note that if we directly use that metric to determine the final neighborhoods, it can suffer from the compounding errors generated by the model, where the error of the neighborhoods involved in the next step of representation learning. Thus we use this metric to determine the order of the neighborhood for the low-level policy.

4.3.4 Low-level Topology Optimization. Low-level policy focuses on determine the effective neighborhoods for each target nodes with the given order of the candidate neighborhoods. Given the j^{th} neighbor x_j in the generated order, π_l takes an action $a_j = \{0, 1, 2\}$ at time step j to decide whether to select the t-th neighborhood x_j, where $a = 0$ indicates HRLHF considers x_j not x_t's neighborhoods. $a = 1$ indicates HRLHF considers x_j as x_t's neighborhoods, and $a = 2$ indicates HRLHF chooses to stop the neighborhood selection. We will make most $|\mathcal{N}(x_t)|$ decisions to select the effective neighbors for node x_t. Here the complexity are largely reduced. As illustrated in Fig. 2, the low-level policy $\pi^l(a_j|s_j)$ is learned to map the state s_j to the action a_j at time step $t, t = 1, ..., |\hat{\mathcal{N}}(v)|$, meanwhile the corresponding reward $r_j = r_\theta(x_t, x_j)$ will be calculated from the learned model at each time step. Our goal is to maximize the total reward of all the actions taken during these time steps, which can be learned by policy gradient,

$$\nabla J_{\pi^l} = \mathbb{E}_{s\sim\pi_l}[Q_l(s, \pi^l(a_j|s_j)]. \quad (3)$$

Here we utilize DDPG [34] to optimize the low-level policy.

4.4 Equivalence between HRLHF and Submodular Maximization Problem

We will establish the following facts, which imply the equivalence between HRLHF and submodular maximization problem,

- The total reward function R_π defined in the effective neighbor selection phase is a submodular function, which is equivalent to $f_s : 2^N \rightarrow \mathbb{R}$.
- The submodular maximization problem can be formulated as an MDP which is equivalent to HRLHF.
- The objective function in HRLHF is equivalent to the counterpart in submodular maximization.

Firstly, the goal of submodular maximization is to find the w^*, with the objective function:

$$w^*(\Phi) = \arg\max_{A\subseteq\Omega:|A|\leq K} f_s(A), \quad (4)$$

where $f_s(A)$ is the cumulative value of each element in set A. Let A be the selected neighborhood set $\hat{\mathcal{N}}(v)$, then $R_\pi(\hat{\mathcal{N}}(v)) = f_s(\hat{\mathcal{N}}(v))$.

Secondly, the submodular maximization problem can be formulated as an MDP where the set A with the selected items denotes the state. After adding a new item c into A, the state is updated to

$A \cup \{c\}$. Lastly, the objective function of HRLHF also aligns to the optimizer w in submodular maximization, where each π_θ can be considered as an optimizer w in Equation (4):

$$\pi_\theta^* = \arg\max_\theta \mathbb{E}_A R_\pi(A), \tag{5}$$

where A is equivalent to the signal neighbor set $\hat{\mathcal{N}}(v)$.

5 CAUSAL ROOT CAUSE ANALYSIS

In the initial stage, a causal graph $\mathcal{G}$ is established, enabling the determination of the underlying cause of the failure. We then enhance CausalRCA technique in [8] by incorporating the Markov and Granger causality characteristics presented in time-series data.

5.1 Causal Mechanism Change

Inspired by the research in [8], we employ the dependency graph as a graphical causal model and perform RCA by identifying changes in the causal mechanism of each node in a failure. It is worth noting that the causal graph is represented as a Directed Acyclic Graph (DAG). Let $\mathbf{X} = (X_1, X_2, \cdots, X_n)$ denotes the set of nodes in the graph $\mathcal{G}$. A directed edge from node X_i to X_j signifies that node X_i directly impacts node X_j. In the context of microservice systems, this translates to service j being dependent on service i, and thus its latency affecting its parent. A joint distribution $\mathbf{P_X}$ is considered consistent with the causal graph $\mathcal{G}$ if it can be generated following the edges specified in $\mathcal{G}$. By factorizing the graphical causal model, we have:

$$\mathbf{P_X} = \prod_{j=1}^{n} P_{X_j|PA_j}, \tag{6}$$

where $P_{X_j|PA_i}$ refers to the causal mechanism of the variable X_j with respect to its immediate parent variables PA_j in the causal structure. Each parent-child pair characterized by $X_j|PA_j$ symbolizes an independent physical process – modification to any one of these relationships will not have impact on the others [35].

When failures occur, root cause nodes typically exhibit a noticeable increase in latency which in turn affects its parent services. This results in a shift in both the distribution and the causal mechanisms of these root cause nodes. We mathematically define a subset $\mathbf{R} \subseteq (1, \cdots, n)$, as the root cause node set that display a change in their causal mechanisms. Upon the occurrence of a failure, the joint distribution $\mathbf{P_X}$ transforms to $\mathbf{P_X^R}$ by replacing the original causal mechanisms of the subset $\mathbf{R}$ with new ones, *i.e.*,

$$\mathbf{P_X} = \prod_{j \notin \mathbf{R}} P_{X_j|PA_j} \prod_{j \in \mathbf{R}} \tilde{P}_{X_j|PA_j}, \tag{7}$$

where $\tilde{P}_{X_j|PA_j}$ is the new causal mechanism of node j when anomaly happens, which will be subsequently used to evaluate the anomaly attribution. In practice, both $P_{X_j|PA_j}$ and $\tilde{P}_{X_j|PA_j}$ are approximated with machine learning models from observational data.

5.2 RCA for Marginal Distribution Change

We define a front-end node X_f, which encloses the end-to-end latency of the microservice system. The objective of the RCA is to identify the root cause nodes that lead to the anomaly of X_f. Assume that its distribution change from P_{X_f} to $\tilde{P}_{X_f}$ when failures

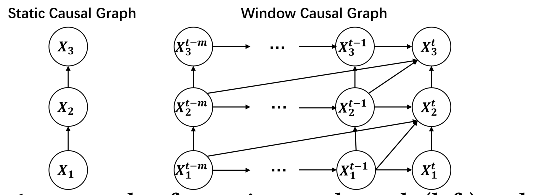

Figure 3: An example of a static causal graph (left) and a window causal graph (right).

happen. By marginalizing all exclusive variables, we have

$$P_{X_f} = \sum_{X_j, j \neq f} \prod_{j=1}^{n} P_{X_j|PA_j}. \tag{8}$$

Given the root cause node set $\mathbf{R}$,

$$\tilde{P}_{X_f} = \sum_{X_j, j \neq f} \prod_{j \notin \mathbf{R}} P_{X_j|PA_j} \prod_{j \in \mathbf{R}} \tilde{P}_{X_j|PA_j}. \tag{9}$$

We evaluate the attribution of each node on X_f by determining the extent of variation in P_{X_f} when each "original" mechanism $P_{X_j|PA_j}$ in $\mathbf{R}$ is substituted with its new counterpart $\tilde{P}_{X_j|PA_j}$. However, the alteration in P_{X_f} is susceptible to the arrangement of replaced causal mechanisms in $\mathbf{R}$. To mitigate the variance caused by permutation, we follow the practice in [8] and utilize the Shapley value [36] as a debiasing mechanism. The Shapley value provides the average marginal contribution across all possible permutations.

Given an evaluation function Φ defined for marginal distributions and a set of root cause nodes $\mathbf{R}$, the marginal contribution of node X_j towards the variation in the functional of the marginal distribution of the target front-end node X_f, *i.e.*, $\Delta\Phi = \Phi(P_{X_f}) - \Phi(\tilde{P}_{X_f})$ can be calculated using the following equation:

$$\mathcal{V}_\Phi(j|\mathbf{R}) := \Phi(P_{X_f}^{\mathbf{R}\cup\{j\}}) - \Phi(P_{X_f}^{\mathbf{R}}). \tag{10}$$

The marginal contribution $\mathcal{V}_\Phi(j|\mathbf{R})$ quantifies the impact of substituting the causal mechanism of X_j on the alteration in the aggregate of P_{X_f}, taking into account that the causal mechanisms of variables in set $\mathbf{R}$ have already been substituted. Then, the contribution of the Shapley value to $\Delta\Phi$ can be expressed as $\vartheta_j(\Phi) := \overline{\mathcal{V}_\Phi(j|\mathbf{R})}$, where $\overline{\mathcal{V}_\Phi(j|\mathbf{R})}$ is obtained by computing the average of $\mathcal{V}_\Phi(j|\mathbf{R})$ across all possible permutations in $\mathbf{R}$. In this paper, the median function is utilized to assess the alteration in latency of the front-end node during anomalies through Φ. However, any function that quantifies the disparity between the latency distributions can be applied as Φ.

5.3 Extension to Time Series Data

The initial CausalRCA approach presented in [8] has limitations in that it only works with static data and does not consider the Markov property or Granger causality present in microservice systems' time series. To address these shortcomings, we enhance CausalRCA by transforming the static causal graphs into window causal graphs [37], which enable the capture of these important properties and refine the graphical causal model.

To this end, we make modifications to each service node in the original CausalRCA method:

(1) We add historical latency data with a maximum lag of m to the graph, resulting in each node X_i being extended to $\mathbf{X_i} = (X_i^{t-m}, \cdots, X_i^{t-1}, X_i^t)$ to incorporate temporal information.

(2) We treat the time series at each node as a Markov chain by connecting consecutive variables $X_i^{t-l} \rightarrow X_i^{t-l+1}$ for each $0 < l \leq m$ and for all node $X_i \in \mathbf{X}$.
(3) For any existing causal mechanism between nodes $X_i \rightarrow X_j$, we add links between historical $\mathbf{X_i}$ and X_j^t to capture the Granger causation, *i.e.*, $(X_i^{t-m}, \cdots, X_i^{t-1}, X_i^t) \rightarrow X_j^t$.

We show the original static causal graph and extended window causal graph in Fig. 3. We propose this method as T-CausalRCA, an extension of the original CausalRCA method in [8], to capture the temporal properties of microservice systems. The T-CausalRCA method only updates the graph structure, while preserving the original RCA process introduced in this section.

The utilization of the window causal graph allows for the depiction of the system's dynamic within a specified time frame, as well as the quantification of service latency accumulation attributed to congestion among its descendants. This addresses the shortcomings of previous research methods [8–11] and enhances the precision and robustness of the RCA process. Once the attribution scores $\overline{\mathcal{V}_\Phi(j|\mathbf{R})}$ are obtained, we select nodes with top-K highest scores as the suspicious root cause services.

6 EXPERIMENTS

6.1 Dataset

In this paper, we conduct extensive experiments to demonstrate the robustness and effectiveness of our proposed method. Benchmark datasets employed in this study include multiple incidents collected from two simulation environments, namely *(i)* **Pymicro**, and *(ii)* **SynExchange**, and a real-world data trace *(iii)* **RealExchange** [1].

We utilize the Pymicro simulation system [38], which comprises of 16 microservices configured in a specific architecture. This system is widely used to serve the root cause analysis purpose. To establish the benchmark dataset, we conducted simulation where the front-end service was requested for a duration of 1,500 seconds, and the response times of all the microservices were recorded. Additionally, to simulate incidents, we manipulated the internal latency of one or multiple selected backend services, which served as the primary cause of the incident. By repeating this process, we employ Pymicro to generate 5 synthetic dataset for the experiment purpose, where each dataset includes 1-3 root cause services that lead to abnormal latency on the root node.

In addition to the Pymicro simulation system, this study also employs a real-world industrial trace dataset collected from the Microsoft M365 Exchange service [4]. The dataset, which is generated when users send emails, comprises log records that are created as the emails pass through different microservice components in the system. This real-world system can produce hundreds of millions of records per day, spread across hundreds of microservices. To acquire the datasets for this study, we employ a two-fold approach: *(i)* synthetic datasets, generated using the random walk algorithm and *(ii)* a real dataset collected in industrial production environments.

The synthetic datasets, referred to as **SynExchange**, are generated by sampling from the topology graph of the Exchange system using the random walk algorithm, resulting in subgraphs of the full architecture with 12-36 microservices. These subgraphs were then used to simulate system anomalies by injecting internal latency into selected backend services. In total, 5 such synthetic datasets were generated using this process.

The real dataset, referred to as **RealExchange**, was collected over a period of six days and aggregated in 5-minute intervals. The dataset comprises 20,748 data points across 12 microservices. During the data collection period, three different services exhibited exceptionally high latency, resulting in an incident on the front-end node. These services were subsequently identified as the root causes of the incident in the RealExchange dataset. It is worth noting that, all data collection procedures adhered to relevant regulations. It is fully anonymous and desensitized, thereby eliminating any potential privacy issues associated with its usage in this study.

6.2 Baselines & Evaluation Metrics

For constructing dependency graphs, we select 7 state-of-the-art method as baselines. Peter-Clark Algorithm (PC) [19] is a traditional method for causal discovery. Linear Notears [20] formulates the graph structure learning problem as a purely continuous optimization problem based on the linear structure equation model (SEM). Nonlinear Notears [21] extends the previous Linear Notears to a nonlinear version by using the non-parametric SEM. Dycause [33] first analyzes the temporal dynamics of correlations then adapts Graph Fusing with adaptive thresholding. GC [23] uses Granger Causality to find linear correlations. Furthermore, Neural Granger Causality [39] is an extension of the original GC model that uses neural networks to capture non-linear correlations. MTGNN [40] is proposed to conduct a multivariate time series forecasting by using a graph neural network, which involves a graph construction module. We used the outputs of the graph construction module as dependency graphs. The obtained dependency graphs is refined by randomly removal of edges for all existing cycles to ensure its directed acyclic properties required by RCA. Next, we compared our proposed method T-CausalRCA with its naive CausalRCA [8] and Backtrace RCA [33], which performs the backward BFS and estimates abnormality scores by considering path correlation strength and Pearson correlation coefficient.

As a means of evaluating HRLHF, we applied a variety of metrics on both graph similarity and precision of RCA results. To measure graph similarity, we used Average precision (AP), Area Under the Curve (AUC), Jaccard similarity [41], Network Portrait Divergence (NPD) [42] and Path-Acc. Jaccard similarity is a statistic used in understanding the similarities between sample sets. The NPD, $D_{JS}(G, G')$ between two graphs G and G', is the Jensen-Shannon divergence, which provides a single value to quantify the dissimilarity of the two networks by means of their distance distributions. Path-Acc measures the number of paths from root cause to frontend existing on a dependency graph among all the ground-truth root causes: Path-Acc $= \frac{\sum_{i=1}^{k} \text{Path}(\text{RC}_i^{\text{true}}, \text{FE})}{|\text{RC}^{\text{true}}|}$, where Path$(a, b)$ is 1 if there exists a path from node a to node b on a DAG, otherwise 0. FE represents the frontend service.

As for the evaluation on RCA results, two evaluation metrics are presented: **PR@k** [7] and **RankScore** [33]. **PR@k** is the number of correct root causes among the top-k predictions: PR @k =

[1]Due to the non-disclosure agreement, the **RealExchange** data cannot be released. Synthetic dataset used in this study, including **Pymicro** and **SynExchange** will be released upon publication.

Table 1: Performance of dependency graph discovery.

	Pymicro					SynExchange					RealExchange				
Method	AP	AUC	Jaccard	NPD	Path-Acc	AP	AUC	Jaccard	NPD	Path-Acc	AP	AUC	Jaccard	NPD	Path-Acc
PC	0.1700	0.6233	0.1911	0.3370	0.5000	0.2679	0.7942	0.3158	0.3177	**0.9333**	0.1413	0.5528	0.1154	0.3634	0.0000
Linear Notears	0.1432	0.6148	0.1699	0.3669	0.4583	0.0551	0.4878	0.0150	0.4220	0.0000	0.1155	0.4859	0.0698	0.3856	**1.0000**
Nonlinear Notears	0.1625	0.6557	0.1804	0.3893	0.7500	0.0555	0.4866	0.0237	0.4746	0.0000	0.1308	0.5447	0.1163	0.3457	0.6667
Dycause	0.0889	0.5216	0.0704	0.3830	0.6667	0.0576	0.4957	0.0321	0.4344	0.0500	0.1251	0.5252	0.0909	0.3600	**1.0000**
GC	0.1488	0.5825	0.1456	0.2985	0.5000	0.0544	0.3611	0.0000	0.6488	0.0000	0.1312	0.5428	0.1111	0.3193	0.0000
NGC	0.1214	0.6354	0.1358	0.6429	0.3750	0.0726	0.5100	0.0586	0.5552	0.3000	0.1473	0.5878	0.1489	0.3500	0.3333
MTGNN	0.1594	0.7004	0.1843	0.5462	**1.0000**	0.0515	0.4259	0.0183	0.6460	0.1167	0.1251	0.5271	0.1000	0.2003	0.3333
HRLHF	**0.7017**	**0.9657**	**0.7054**	**0.3222**	**1.0000**	**0.6485**	**0.9383**	**0.6613**	**0.2625**	0.8667	**0.6441**	**0.9176**	**0.6522**	**0.1412**	**1.0000**

$\frac{\sum_{i=1}^{k} RC_i^{\text{pred}} \in RC^{\text{true}}}{\min(|RC^{\text{true}}|, k)}$, where RC^{true} is the ground-truth root cause list, RC^{pred} is the set of predicted root cause. Moreover, we average **PR@k**, k=1,2,...,5 as **PR@Avg**. Another metric we use to evaluate root cause analysis rankings is **RankScore** $(0-1)$: $\text{RankScore} = \frac{1}{|RC^{\text{true}}|} \sum_{v \in RC^{\text{true}}} s(v)$,

where $s(v) = \begin{cases} \frac{N - \max(0, \text{rank}(v) - |RC^{\text{true}}|)}{N}, & \text{if } v \in RC^{\text{pred}} \\ 0 & \text{otherwise} \end{cases}$ where N is the length of predicted root cause list. rank (v) is the rank of service v ranging from 1 to N. In RankScore, each ground-truth root cause service's rank performance is averaged.

6.3 Causal Discovery Visualization

In this section, we present visualizations of the learned causal graphs generated by our framework, HRLHF, on the RealExchange dataset to gain insights into its performance in RCA. The visualizations are shown in Figure 4. Our observations show that: *(i)* Compared to the ground truth causal graph, HRLHF exhibits the largest overlap, while the other baseline methods only capture a limited number of causality relationships, highlighting the importance of incorporating human feedback in uncovering hidden dependencies. *(ii)* HRLHF may learn a few redundant causality relationships for nodes with sparse links, which does not have a significant impact on RCA results. This is because our framework only requires limited human feedback, and only the most critical causality relationships are detected.

6.4 Results of Causal Discovery

We evaluate the performance of Causal Discovery via the similarity between the learned causal graph and the ground-truth causal structure. Table 1 depicts the comparative results of HRLHF and its 7 state-of-the-art competitors on graph similarity. We observe that: (i) HRLHF performs much better than the other methods on Pymicro and RealExchange. Specifically, HRLHF achieves average Jaccard about 250% higher than the second best method (PC) and an average PathAcc about 33% higher than the second best method (MTGNN). It demonstrates the effectiveness of the human feedback mechanism in guiding to learn the causal graph in cloud system. (ii) HRLHF acieves the most significant improvement compared to the baselines, which verifies that for more complex environment, the human feedback can raise more benefits to guide to learn a correct causal graph. (iii) As for SyncExchange, HRLHF also shows significant improvement compared to the baselines on most metrics. Due to the anomalies are injected only in the final phase, even we learn a much similar calsual graph, we would also be possible that we achieve a little lower PathAcc than PC. Thus, we leverage the

Table 2: RCA performance of all approaches in this study.

Dataset	RCA Method	PR@5	PR@Avg	RankScore
Pymicro	Backtrace	0.7083	0.6583	0.7014
	Causal	0.7500	0.6000	0.7750
	T-Causal	**0.7917**	**0.6833**	**0.8620**
SynExchange	Backtrace	0.3000	0.1400	0.4107
	Causal	0.5667	0.6133	0.6858
	T-Causal	**0.8667**	**0.8300**	**0.8591**
RealExchange	Backtrace	0.6667	0.8000	0.6667
	Causal	**1.0000**	0.8667	0.9333
	T-Causal	**1.0000**	**0.9333**	**0.9667**

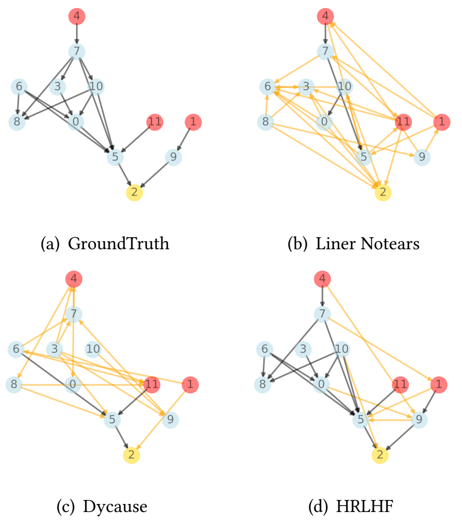

Figure 4: Causal graph discovery results visualization on the RealExchange dataset. True root causes are in red, frontend services are in yellow, mispredicted edges are in orange.

temporal information to help enhance the RCA, where the results are discussed in next subsection. (iv) PC outperforms the other baselines in most metrics and datasets, likely due to the facts that dependency relationship are highly complex in the cloud system, where it is hard to train a machine learning model to learn the causal structure only from the observable datasets.

6.5 Results of Root Cause Analysis

In this study, we compare the performance of three RCA methods: Backtrace RCA [33], CausalRCA [8] and the proposed T-CausalRCA. The RCA was conducted based on the dependency graphs constructed in stage 1 using RLHF, and the results were compared

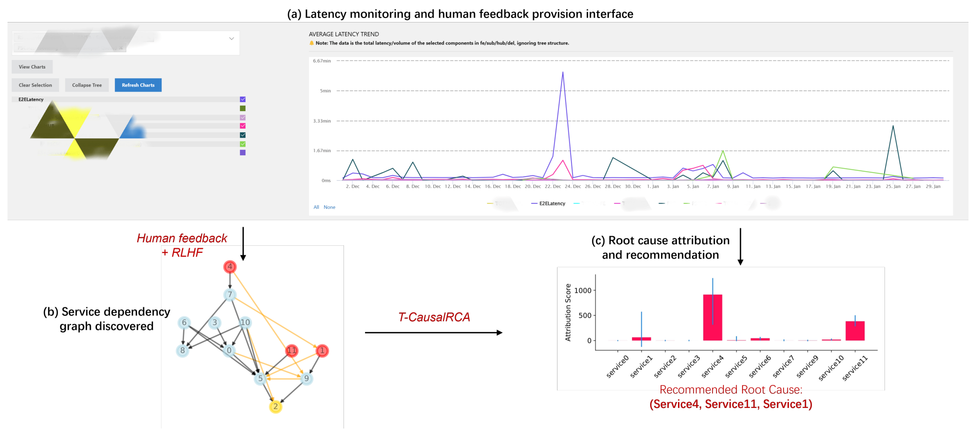

Figure 5: The implementation of a proposed RCA framework within Microsoft Exchange. An interface was created to aid engineers in monitoring latency and provide feedback to the causal discovery process. The T-CausalRCA algorithm was utilized to generate anomaly attribution scores and provide root cause service recommendations.

across three datasets. The results are presented in Table. 2 and averaged over 5 sub-datasets for Pymicro and SynExchange respectively. Observe that the proposed T-CausalRCA method achieves the best results across all evaluation metrics compared to the other methods. Additionally, T-CausalRCA demonstrates robust performance, regardless of the number of root cause nodes present in each sub-dataset of Pymicro and SynExchange. This is attributed to the use of extended window causal graphs in T-CausalRCA, which enables capturing the dynamic behavior of the system. Furthermore, the results on the real-world dataset RealExchange show that both CausalRCA and T-CausalRCA achieved perfect PR@5 scores, however, T-CausalRCA outperforms CausalRCA in ranking the problematic nodes. These results highlight the superiority of the ranking ability of the T-CausalRCA method, which can provide valuable guidance to engineers in prioritizing the investigation and mitigation efforts.

In addition, it is observed that causal methods such as CausalRCA and T-CausalRCA demonstrate superior performance compared to the traditional correlation-based approach of Backtrace RCA. This implies that the relationship between service latency and its root cause cannot be solely explained by the correlation between the latencies of different services. The existence of confounding factors that contribute to the high latency of multiple services can result in misleading conclusions from purely correlation-based RCA methods. Hence, it can be concluded that causal approaches are more reliable for RCA than correlation-based methods.

7 INDUSTRIAL PRACTICE

The proposed framework has been integrated as a crucial component within the Microsoft Exchange team for the purpose of localizing the root cause of service anomalies. We show the online RCA platform in Fig. 5. Service names are masked to comply with the company policy. It has shown a remarkable 52% improvement in RCA PR@5 when compared to the existing online model. This system can be utilized on a daily basis to detect problematic nodes, or it can be activated upon detection of anomalies to promptly identify the root cause.

During the causal discovery phase, a user interface has been developed to facilitate interaction between the framework and engineers, as depicted in part **(a)** of Figure 5. The model will query the relationships between nodes X_i and X_j, and the engineers will provide feedback based on their domain expertise, service metadata, and latency behavior. This feedback will assist in revealing the dynamic causal relationships within the microservice system, as illustrated in part **(b)** of Figure 5.

Subsequently, latency data is collected from all nodes in the causal structure to perform T-CausalRCA. The results will attribute the anomaly to each service, as demonstrated in part **(c)** of Figure 5. Based on the attribution score, the framework will provide recommendations for the root cause services of the failure, which will then be subject to further investigation by the engineers. It is imperative to differentiate between normal and abnormal periods in the data, which can be accomplished through the use of modern time series anomaly detection techniques [43–45], or through the actions of site reliability engineers.

8 CONCLUSION

In this paper, we propose a novel machine learning framework for performing effective RCA in dynamic microservice-based systems. Our framework incorporates human expertise in the form of human-in-the-loop training, where experienced Site Reliability Engineers (SREs) assist in the discovery of the service dependency graph. This results in a more accurate and robust service dependency graph, reducing uncertainty and error in the RCA process. To minimize the human effort required in the training process, we leverage Hierachical Reinforcement Learning with Human Feedback (HRLHF) to optimize the model query complexity. Our framework reduces the query complexity from $O(N^2)$ to $O(1)$ while maximizing the utilization of expert knowledge. In addition, we extend the causal graph to a temporal dimension, capturing the temporal characteristics of latency evolution and system dynamics. We validate the effectiveness of the proposed framework through experiments on both synthetic datasets and real-world scenarios in the Microsoft Exchange service. It is noteworthy that our framework has been successfully integrated as a crucial component in the Microsoft M365 Exchange service, improving the reliability of the system.

REFERENCES

[1] Jacopo Soldani and Antonio Brogi. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. *ACM Computing Surveys (CSUR)*, 55(3):1–39, 2022.

[2] Haifeng Liu, Jinjun Zhang, Huasong Shan, Min Li, Yuan Chen, Xiaofeng He, and Xiaowei Li. Jcallgraph: tracing microservices in very large scale container cloud platforms. In *Cloud Computing–CLOUD 2019: 12th International Conference, Held as Part of the Services Conference Federation, SCF 2019, San Diego, CA, USA, June 25–30, 2019, Proceedings 12*, pages 287–302. Springer, 2019.

[3] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Transactions on Software Engineering*, 47(2):243–260, 2018.

[4] Zhengran Zeng, Yuqun Zhang, Yong Xu, Minghua Ma, Bo Qiao, Wentao Zou, Qingjun Chen, Meng Zhang, Xu Zhang, Hongyu Zhang, Xuedong Gao, Hao Fan, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. Traceark: Towards actionable performance anomaly alerting for online service systems. In *To appear in Proc. of ICSE*, 2023.

[5] Xu Zhang, Chao Du, Yifan Li, Yong Xu, Hongyu Zhang, Si Qin, Ze Li, Qingwei Lin, Yingnong Dang, Andrew Zhou, et al. Halo: Hierarchy-aware fault localization for cloud systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 3948–3958, 2021.

[6] Dewei Liu, Chuan He, Xin Peng, Fan Lin, Chenxi Zhang, Shengfang Gong, Ziang Li, Jiayu Ou, and Zheshun 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)*, pages 338–347. IEEE, 2021.

[7] Ping Wang, Jingmin Xu, Meng Ma, Weilan Lin, Disheng Pan, Yuan Wang, and Pengfei Chen. Cloudranger: Root cause identification for cloud native systems. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 492–502. IEEE, 2018.

[8] Kailash Budhathoki, Dominik Janzing, Patrick Bloebaum, and Hoiyi Ng. Why did the distribution change? In *International Conference on Artificial Intelligence and Statistics*, pages 1666–1674. PMLR, 2021.

[9] Yuan Meng, Shenglin Zhang, Yongqian Sun, Ruru Zhang, Zhilong Hu, Yiyin Zhang, Chenyang Jia, Zhaogang Wang, and Dan Pei. Localizing failure root causes in a microservice through causality inference. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2020.

[10] JinJin Lin, Pengfei Chen, and Zibin Zheng. Microscope: Pinpoint performance issues with causal graphs in micro-service environments. In *Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16*, pages 3–20. Springer, 2018.

[11] Muhammad Azam Ikram, Sarthak Chakraborty, Subrata Mitra, Shiv Saini, Saurabh Bagchi, and Murat Kocaoglu. Root cause analysis of failures in microservices through causal discovery. In *Advances in Neural Information Processing Systems*.

[12] Murat Kocaoglu, Amin Jaber, Karthikeyan Shanmugam, and Elias Bareinboim. Characterization and learning of causal graphs with latent variables from soft interventions. *Advances in Neural Information Processing Systems*, 32, 2019.

[13] Judea Pearl. *Causality*. Cambridge university press, 2009.

[14] Shiqi Hao, Yang Liu, Yu Wang, Yuan Wang, and Wenming Zhe. Three-stage root cause analysis for logistics time efficiency via explainable machine learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2987–2996, 2022.

[15] Mattia Carletti, Chiara Masiero, Alessandro Beghi, and Gian Antonio Susto. Explainable machine learning in industry 4.0: Evaluating feature importance in anomaly detection to enable root cause analysis. In *2019 IEEE international conference on systems, man and cybernetics (SMC)*, pages 21–26. IEEE, 2019.

[16] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

[17] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

[18] Peter Spirtes and Clark Glymour. An algorithm for fast recovery of sparse causal graphs. *Social science computer review*, 9(1):62–72, 1991.

[19] Peter Spirtes, Clark N Glymour, Richard Scheines, and David Heckerman. *Causation, prediction, and search*. MIT press, 2000.

[20] Xun Zheng, Bryon Aragam, Pradeep K Ravikumar, and Eric P Xing. Dags with no tears: Continuous optimization for structure learning. *Advances in neural information processing systems*, 31, 2018.

[21] Xun Zheng, Chen Dan, Bryon Aragam, Pradeep Ravikumar, and Eric Xing. Learning sparse nonparametric dags. In *International Conference on Artificial Intelligence and Statistics*, pages 3414–3425. PMLR, 2020.

[22] Shengyu Zhu, Ignavier Ng, and Zhitang Chen. Causal discovery with reinforcement learning. In *International Conference on Learning Representations*.

[23] Clive WJ Granger. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica: journal of the Econometric Society*, pages 424–438, 1969.

[24] Chaoyun Zhang, Marco Fiore, Cezary Ziemlicki, and Paul Patras. Microscope: mobile service traffic decomposition for network slicing as a service. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–14, 2020.

[25] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. Deep learning in mobile and wireless networking: A survey. *IEEE Communications surveys & tutorials*, 21(3):2224–2287, 2019.

[26] Chaoyun Zhang, Marco Fiore, Iain Murray, and Paul Patras. Cloudlstm: A recurrent neural model for spatiotemporal point-cloud stream forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10851–10858, 2021.

[27] Chaoyun Zhang, Xi Ouyang, and Paul Patras. Zipnet-gan: Inferring fine-grained mobile traffic patterns via a generative adversarial neural network. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 363–375, 2017.

[28] Daniele Marinazzo, Mario Pellicoro, and Sebastiano Stramaglia. Kernel-granger causality and the analysis of dynamical networks. *Physical review E*, 77(5):056215, 2008.

[29] Anonymous. CUTS: Neural causal discovery from unstructured time-series data. In *Submitted to The Eleventh International Conference on Learning Representations*, 2023. under review.

[30] Sindy Löwe, David Madras, Richard Zemel, and Max Welling. Amortized causal discovery: Learning to infer causal graphs from time-series data. In *Conference on Causal Learning and Reasoning*, pages 509–525. PMLR, 2022.

[31] Pengfei Chen, Yong Qi, Pengfei Zheng, and Di Hou. Causeinfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 1887–1895. IEEE, 2014.

[32] Myunghwan Kim, Roshan Sumbaly, and Sam Shah. Root cause detection in a service-oriented architecture. *ACM SIGMETRICS Performance Evaluation Review*, 41(1):93–104, 2013.

[33] Yicheng Pan, Meng Ma, Xinrui Jiang, and Ping Wang. Faster, deeper, easier: crowdsourcing diagnosis of microservice kernel failure from user space. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 646–657, 2021.

[34] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[35] Judea Pearl et al. Models, reasoning and inference. *Cambridge, UK: CambridgeUniversityPress*, 19(2), 2000.

[36] Lloyd S Shapley et al. A value for n-person games. 1953.

[37] Charles K. Assaad, Emilie Devijver, and Eric Gaussier. Causal discovery of extended summary graphs in time series. In *The 38th Conference on Uncertainty in Artificial Intelligence*, 2022.

[38] Microservice-based sample application written in Python. https://github.com/rshriram/pymicro, 2023. [Online; accessed 27-Jan-2023].

[39] Chan Li Long, Yash Guleria, and Sameer Alam. Air passenger forecasting using neural granger causal google trend queries. *Journal of Air Transport Management*, 95:102083, 2021.

[40] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 753–763, 2020.

[41] Allan H Murphy. The finley affair: A signal event in the history of forecast verification. *Weather and forecasting*, 11(1):3–20, 1996.

[42] James P Bagrow and Erik M Bollt. An information-theoretic, all-scales approach to comparing networks. *Applied Network Science*, 4(1):1–15, 2019.

[43] Zhihan Li, Youjian Zhao, Jiaqi Han, Ya Su, Rui Jiao, Xidao Wen, and Dan Pei. Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 3220–3230, 2021.

[44] John Paparrizos, Yuhao Kang, Paul Boniol, Ruey S Tsay, Themis Palpanas, and Michael J Franklin. TSB-UAD: an end-to-end benchmark suite for univariate time-series anomaly detection. *Proceedings of the VLDB Endowment*, 15(8):1697–1711, 2022.

[45] Minghua Ma and Shenglin Zhang. Jump-starting multivariate time series anomaly detection for online service systems. In *Proceedings of the 2021 USENIX Annual Technical Conference*, 2021.