

CausalRCA: Causal Inference based Precise Fine-grained Root Cause Localization for Microservice Applications

Ruyue Xin

*Multiscale Networked Systems (MNS)
research group, University of Amsterdam
Amsterdam, Netherland*

Peng Chen*

*School of Computer and Software
Engineering, Xihua University
Chengdu, China
chenpeng@mail.xhu.edu.cn*

Zhiming Zhao*

*Multiscale Networked Systems (MNS)
research group, University of Amsterdam
Amsterdam, Netherland
z.zhao@uva.nl*

Abstract—Effectively localizing root causes of performance anomalies is crucial to enabling the rapid recovery and loss mitigation of microservice applications in the cloud. Depending on the granularity of the causes that can be localized, a service operator may take different actions, e.g., restarting or migrating services if only faulty services can be localized (namely, coarse-grained) or scaling resources if specific indicative metrics on the faulty service can be localized (namely, fine-grained). Prior research mainly focuses on coarse-grained faulty service localization, and there is now a growing interest in fine-grained root cause localization to identify faulty services and metrics. Causal inference (CI) based methods have gained popularity recently for root cause localization, but currently used CI methods have limitations, such as the linear causal relations assumption and strict data distribution requirements. To tackle these challenges, we propose a framework named CausalRCA to implement fine-grained, automated, and real-time root cause localization. The CausalRCA uses a gradient-based causal structure learning method to generate weighted causal graphs and a root cause inference method to localize root cause metrics. We conduct coarse- and fine-grained root cause localization to evaluate the localization performance of CausalRCA. Experimental results show that CausalRCA has significantly outperformed baseline methods in localization accuracy, e.g., the average $AC@3$ of the fine-grained root cause metric localization in the faulty service is 0.719, and the average increase is 10% compared with baseline methods. In addition, the average $Avg@5$ has improved by 9.43%.

Index Terms—microservice applications, root cause localization, fine-grained, causal inference, monitoring data

I. INTRODUCTION

Microservices architecture [1] builds cloud applications by decomposing the system functionalities into multiple independently deployable units, making applications more resilient, robust, and adaptable to dynamic cloud environments. The performance of a microservice application is vital to guarantee the quality of user experience (QoE) and service (QoS) [2]. However, performance anomalies, such as degraded response time, are inevitable due to the large scale and complex dependencies of services, causing enormous economic loss and user dissatisfaction [3]. Furthermore, the performance of

applications heavily depends on the underlying resources [4]; for example, high CPU usage results in a congested queue and growing latency [5]. In order to enable application operations to take actions to resolve performance anomalies effectively, root cause localization to identify faulty services or resources is at the core of software maintenance for online service systems [6].

A microservice application can be observed by monitoring tools, which help operators to detect performance anomalies [7], [8]. However, performance anomaly detection only notifies operators when an anomaly occurs. To effectively handle performance anomalies, operators need to be informed about where the anomaly occurs (e.g., the faulty service) and what causes the anomaly (e.g., the memory leak). Root causes of a performance anomaly can be localized at different granularity: coarse-grained and fine-grained [9]. Coarse-grained means that only faulty services can be identified, and the corresponding action will be the migration or restart of the entire service [10], which is simple and straightforward but may not recover anomalies and has a higher risk of affecting other services and longer recovery times [11]. The developer of Instana Autotrace¹ emphasized the importance of identifying anomaly locations and root causes to avoid delays associated with restarting services, as this may not solve anomalies [12].

At a fine granularity, root cause localization will identify not only the faulty service but also the underlying resources through monitoring metrics of the service [13]. Operators can choose accurate actions to mitigate the performance anomaly using fine-grained root cause when pinpointing indicative metrics on the faulty service [9], [14]. For example, service scale-out has a positive effect and shorter recovery time compared with service restart in the case of underlying resources being insufficient [11]. In industry, fine-grained root cause localization attracts much attention. The CCF AIOps Challenge², jointly organized by industry and academia, aims to solve problems in real IT operations scenarios based on

* Peng Chen and Zhiming Zhao are the corresponding authors

¹<https://www.instana.com/>

²<http://iops.ai/>

production systems of industrial companies (e.g., Sougo, eBay, Tencent) [15], providing the performance diagnosis challenge for microservice systems and requires localizing root causes at the metric level in 2020³. Instana Autotrace⁴, Google Cloud Operations⁵ are commercial platforms to identify root causes to help developers and operators fix performance issues [16]. However, these platforms work with trace data that requires integrating tracking codes into applications and require time and expert technologies to analyze data. Monitoring data, which is different from trace data, can be readily collected and utilized for fine-grained root cause localization in microservice applications, aiding service operators in efficiently and cost-effectively identifying faulty services and pinpointing faulty metrics to resolve performance anomalies [9].

To track the root cause localization problem of microservice applications, some research has developed in recent years. We can classify them into log-based, trace-based, and metric-based according to the data sources [17]. Log-based [18] and trace-based [19], [20] research have limitations, such as complex real-time processing and information extraction. On the other hand, metric-based research uses real-time monitoring data, including service latency and system resources, and focuses on localizing faulty services and resource metrics. This kind of research can assist anomaly recovery in taking actions like resource scaling easily without intervention of application source code [17]. Nowadays, most metric-based research is coarse-grained [21]–[26], and fine-grained root cause localization is starting to catch the attention of researchers [27], [28]. As for localization methods in metric-based research, causal inference (CI) based methods that can model causal effects between services have been developing recently. For example, CauseInfer [29] applies the PC algorithm (named after its authors, Peter and Clark) [30], and MicroDiag [28] uses the linear non-Gaussian acyclic model (LiNGAM) [31] to obtain causal graphs of metrics, which can be seen as anomaly propagation paths. However, currently used CI methods have limitations, such as uncertainty about some causal relations between metrics and strict assumptions about input data and causal relations [13]. Therefore, advanced CI methods can be considered for fine-grained root cause localization to discover anomaly propagation paths and improve localization performance.

Fine-grained root cause localization in a microservice application is challenging because 1) services are often heterogeneous and have different characteristics, which may result in diverse anomaly symptoms for the same issue; 2) the complex dependency between microservices makes it difficult to model the anomaly propagation resulting from faulty services; 3) a large number of anomalous metrics introduced in a system makes it hard to find out the root one for a performance anomaly. To address these challenges, we formulate our main research question: *how to pinpoint the root cause of performance anomalies at a fine granularity based on monitoring data?*

Three sub-questions are proposed: 1) *how to model anomaly propagation between monitoring metrics using CI methods?* 2) *How to precisely determine the root cause based on the propagation model?* 3) *How to evaluate the performance of root cause localization result?*

To answer the research question, we propose a CI-based fine-grained root cause localization framework named CausalRCA for microservice applications in this paper. The framework works when an anomaly is detected. Based on real-time monitoring data, CausalRCA will perform automatic root cause localization, including modeling anomaly propagation paths as a causal graph and ranking metrics to localize the root cause by traversing along the graph. Finally, CausalRCA outputs predicted root causes, which can be used by operators to determine strategies and recovery actions to solve the anomaly. In this paper, we evaluate the localization performance of CausalRCA on the sock-shop⁶ microservice benchmark. When a performance anomaly in sock-shop, such as the high response time of user requests, is detected, we can input monitoring metrics, including service latency and resource metrics of each service, to CausalRCA. After processing, the faulty service and root cause metric, for example, the memory usage metric in the order service, will be identified. Our experimental results show that CausalRCA has improved localization accuracy. For example, the average improvement of AC@5 for the fine-grained root cause metric localization in the faulty service is 9.43% compared with baseline methods.

Our contributions can be summarized below: noitemsep, nolistsep

- We propose an automated, fine-grained root cause localization framework named CausalRCA, which analyzes monitoring data and localizes faulty services and system resources in real-time.
- We provide a gradient-based causal structure learning method in CausalRCA, which can automatically capture linear and non-linear causal relations between monitoring metrics.
- We conduct coarse- and fine-grained experiments to evaluate the localization performance of CausalRCA and demonstrate that the proposed framework has the best localization accuracy compared with baseline methods. For example, the average AC@3 is 0.719, which is a 10% improvement compared with baseline methods.

The rest of the paper is organized as follows. In Section II, we review existing root cause localization research and CI methods. In Section III, we propose a framework for root cause localization and a detailed introduction of each method. In Section IV, we design experiments from coarse-grained to fine-grained to evaluate the localization performance of our framework. Finally, discussion and conclusion are provided in Section V and Section VI.

³<https://competition.aiops-challenge.com/home/competition/1484441527290765368>

⁴<https://www.instana.com/>

⁵<https://cloud.google.com/products/operations>

⁶<https://github.com/microservices-demo/microservices-demo>

II. RELATED WORKS

In recent years, research has developed for root cause localization in distributed system [32], clouds [14], [20]. Based on data sources, we can categorize these researches into three groups: log-based, trace-based, and metric-based [17]. Log-based research [18] mainly localizes root causes based on text logs parsing, which is hard to work in real time. Trace-based research [19], [20] gathers information through the complete tracing of the execution paths and then identifies root causes along those paths. However, trace data only focuses on service level, and it is time-consuming for developers to understand source code well enough to extract trace information. In contrast, metrics-based research uses monitoring data collected from applications and underlying infrastructures to construct causal graphs and infer root causes. Metric-based research can achieve automated, real-time root cause localization based on multi-dimensional information. Therefore, this section will mainly review metrics-based research and CI methods.

A. Metric-based root cause localization research

Based on monitoring data, some researchers identify root causes of performance anomalies with statistical analysis, e.g., identifying anomalous monitoring metrics in parallel with detected anomalies. Want et al. [33] conduct correlation analysis based on mutual information to determine the root-cause metric for the anomalies they detect. However, given that correlation does not ensure causation [34], statistical analysis cannot pinpoint root causes. In addition, some researchers have developed topology graph-based analysis, which reconstructs the topology graph of a running application. For example, Wu et al. [14] generate a topology graph based on deployment information and extract a weighted anomalous subgraph by parsing resource-level monitoring data. Brandón et al. [35] make snapshots of abnormal states of the application as graphs and then identify the root cause of a new anomaly by graph matching. This kind of research uses a reconstructed application topology graph to determine root causes, which can only be used for coarse-grained root cause localization.

To identify related papers for our research on CI-based root cause localization, we conduct a thorough search using Google Scholar. We use specific keywords such as "microservice", "causal inference", and "root cause localization" to narrow down the results, and limit the search to recent papers published between 2012 and 2022. We sort the papers based on relevancy and identified the two most related papers [27], [28]. We carefully examine these papers and their benchmark methods and discover CauseInfer [9], which is a well-respected and influential research work. The CauseInfer focused on fine-grained real reasons causing performance problems, introducing a low-cost, black box cause inference system to build a causality graph and infer the causes from the graph. We use the snowballing technique [36] to explore CauseInfer's citation papers to identify valuable papers and establish a comprehensive understanding of related research in CI-based root cause localization. We classify these research

works into coarse- and fine-grained categories and summarize them chronologically, as shown in Table I.

Existing CI-based root cause localization research works by constructing a causal graph based on monitoring data, i.e., including causal structure learning and root cause inference [17]. Coarse-grained root cause localization usually builds a causal graph based on service level objective (SLO) metrics, such as service latency, and focuses on determining faulty services. For example, Microscope [21], [22] collect information on service interactions and monitoring service latency and then processes them based on PC and breadth-first search (BFS) algorithms to determine possible faulty service of detected anomalies. In addition, CloudRanger [23], MS-Rank [24], [25], and AutoMAP [26] all exploit PC and random walk algorithms to build causal graphs and infer root causes. MS-Rank and AutoMAP use metrics not only service latency but also throughput, power, and resource consumption, whereas AutoMAP develops novel operations to refine the causal graph. Various coarse-grained root cause localization studies have been conducted, but they cannot assist operators in resolving application anomalies with accurate actions.

To address the drawback of coarse-grained root cause localization, some researchers focus on fine-grained root cause localization. Chen et al. first proposed CauseInfer [9], [29], which infers the faulty service and root cause metric by constructing a causality graph of monitoring metrics in each service with the PC algorithm and traversing the metric causality graph with a depth-first search (DFS) method. After several years, Meng et al. provided MicroCause [27], which mainly focuses on the root cause metric localization in a faulty service. It provides a PC-based causal graph building method for time-series data and infers root causes with the random walk method. Afterward, Wu et al. proposed MicroDiag [28], which focuses on fine-grained root cause localization and applies a DirectLiNGAM to build causal graphs and PageRank to infer root causes. Fine-grained root cause localization focuses mainly on SLO metrics and monitoring resource metrics of services and determining the root cause resource metric to help operators take actions like scaling resources [37].

TABLE I: Classification of metric-based root cause localization research

Reference	Year	Causal structure learning	Root cause inference	Input	Root cause	Granularity
Micorscope [21], [22]	2018	Parallelized PC	BFS	Service latency	Faulty service	Coarse-grained
CloudRanger [23]	2018	PC	Random walk	Service latency	Faulty service	Coarse-grained
MS-Rank [24], [25]	2019	PC	Random walk	Multi metrics	Faulty service	Coarse-grained
AutoMAP [26]	2020	PC	Random walk	Multi metrics	Faulty service	Coarse-grained
CauseInfer [9], [29]	2014, 2016	PC	BFS	Service latency	Faulty service	Fine-grained
MicroCause [27]	2021	PCTS	Random walk	Resource metrics	Root cause metric	Fine-grained
MicroDiag [28]	2021	DirectLiNGAM	PageRank	Service and resource metrics	Root cause metric	Fine-grained

In Table I, we can see that much research is focusing on coarse-grained root cause localization. However, fine-grained root cause localization was proposed early and has begun to attract the attention of more researchers in recent years. As for CI methods, we can see that causal structure learning methods like PC and LiNGAM are applied. At the same time, root cause inference methods, such as BFS and random walk, are popular. Based on these works, we consider precise root cause localization is more helpful for microservice application recovery from performance anomalies. At the same time, the localization accuracy of existing works can be improved; for example, the success rate of accurately identifying the root cause may be under 20% [21], [23]. Therefore, we are motivated to explore fine-grained root cause localization with advanced CI methods.

B. Causal inference methods

Causal inference methods, especially causal structure learning, have been researched for several years, and they play a vital role in many areas, such as genetics [38] and biology [39]. The causal structure learning problem can be formulated to learn a directed acyclic graph (DAG) from observational data. Methods can be classified into constrained-based, score-based, function-based, and gradient-based. Constrained-based methods, such as PC and FCI [30], use conditional independence tests to learn the skeleton of the causal graph and then orient the edges based on pre-defined orientation rules. Score-based methods, like GES [40], assign scores to different causal graphs based on a pre-defined score function and then search over the space of DAGs to find the optimal one. Finally, function-based methods, like LiNGAM [31], [41], construct a linear structural equation model (SEM) based on linear and non-Gaussian assumptions, and solve it to get the DAG. These traditional methods contribute much to causal structure learning but have limitations. PC usually has ambiguous causal relations in causal graphs. GES takes a long time to match graphs, which makes it inappropriate to be applied to large-scale data. LiNGAM has strict linear and non-Gaussian assumptions, which makes it impractical.

With the development of deep neural networks, gradient-based methods are developed. Zheng et al. [42] propose an equality constraint to the linear SEM, which enables a suite of continuous optimization techniques such as gradient descent. After that, Yu et al. [43] provide a deep generative model and apply a variant of the structural constraint to learn the DAG. Gradient-based methods have no limitation of input data, can deal with linear and non-linear causal relations in data, and can automatically generate a weighted DAG. Gradient-based methods have been applied to medical [44] and biology [45]. However, to the best of our knowledge, no research has applied gradient-based methods to root cause localization of microservice applications.

Based on DAGs generated by causal structure learning methods, researchers apply graph methods, like BFS [46], random walk [47], and PageRank [48], for root cause inference. The BFS is to traverse the graph and determine the abnormal

node without descendants or with no abnormal descendants as a root cause. A random walk is walking through paths and choosing neighbors randomly in a graph. It determines the node most visited as the root cause. PageRank improves the random walk by adding the possibility of jumping to a random node, which will be used in this paper.

In conclusion, metric-based research can achieve automated and real-time root cause localization compared with log-and trace-based research. However, most existing research is about coarse-grained faulty service localization, while fine-grained root cause metric localization can be more helpful for rapid recovery and loss mitigation. CI-based methods are popular, but currently used methods have their limitations. Therefore, this paper will mainly focus on fine-grained root cause localization and explore gradient-based methods to build causal graphs.

III. ROOT CAUSE LOCALIZATION FRAMEWORK

In this section, we propose a root cause localization framework named CausalRCA, including causal structure learning and root cause inference, and we will introduce detailed methods in the framework. All codes and data can be found in our Github repository CausalRCA⁷.

A. Framework overview

The CausalRCA can automatically build anomaly propagation paths and localize root causes in real-time based on observable metrics. The CausalRCA framework consists of three components: monitoring metrics, causal structure learning, and root cause inference, as shown in Figure 1.

The CausalRCA works when an anomaly occurs, such as the high latency of user requests, and it then automatically build anomaly propagation paths and localize root causes in real-time based on observable metrics. We first collect monitoring data, including service-level data, that is, service latency, and resource-level data, such as container CPU/memory usage. We use $m_i^{s_j}$ to represent a monitoring metric in the service s_j , and all monitoring data is time-series data as shown in Figure 1(a). Based on monitoring data, we then start the causal structure learning. The causal structure learning will automatically build a causal graph of metrics, which can be seen as anomaly propagation paths. We develop the causal structure learning with a gradient-based CI method, which can output a weighted DAG to represent causal relations between metrics as shown in Figure 1(b). With the DAG, we start root cause inference to localize root causes. We apply PageRank to the weighted DAG and output a ranked list of all metrics, as shown in Figure 1(c). Depending on the input data, the CausalRCA can be used for coarse- or fine-grained root cause localization. Coarse-grained works when input service latency, and CausalRCA will output the faulty service. Fine-grained works when input resource metrics, and CausalRCA will output the root cause metric. We evaluate the localization performance of CausalRCA in experiments in Section IV.

⁷https://github.com/AXinx/CausalRCA_code.git

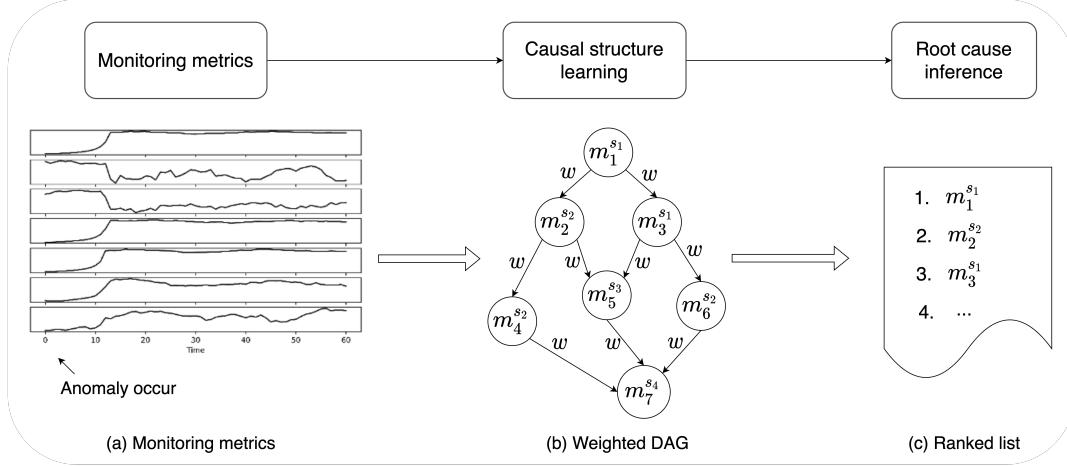


Fig. 1: CausalRCA: details of the root cause localization framework

B. Causal structure learning

The causal structure learning component aims to build a causal graph of monitoring metrics. The causal graphs can be seen as anomaly propagation paths between metrics. We can use a DAG to represent the causal graph, in which each node represents a metric, and each edge represents a cause-effect relationship. Based on related work, we know that traditional causal structure learning methods have strict limitations on input data and relations. Therefore, we implement the causal structure learning in CausalRCA with a gradient-based method, DAG-GNN [43]. DAG-GNN provides a deep generative model, which is a variational autoencoder (VAE) parameterized by a novel graph neural network (GNN) [49], and applies a variant of the structural constraint to learn DAGs. Unlike other causal structure learning methods, the gradient-based method has no limitation of input data, can extract linear or non-linear causal relations between metrics, and automatically outputs a weighted DAG.

We use $X \in \mathbb{R}^{m \times n}$ (m is metrics, n is samples of each metric) to represent input data. To get a DAG from X , Zheng et al. [42] adopt a linear SEM as a data generation model, which is $X = A^T X + Z$ ($A \in \mathbb{R}^{m \times m}$ is the weighted adjacency matrix. $Z \in \mathbb{R}^{m \times n}$ is the noise matrix). To ensure the acyclicity of the DAG, a constraint of A is proposed as:

$$h(A) = \text{tr}[(I + \alpha A \circ A)^m] - m = 0 \quad (1)$$

Based on the linear SEM, we can get $X = (I - A^T)^{-1}Z$, which can be written as $X = f_A(Z)$. This equation is a general form recognized as an abstraction of parameterized GNNs [50]. We can also see that X is generated from a latent representation Z by defining a probabilistic graphical model. The generative model can be developed based on a VAE, and Z follows a standard Gaussian distribution [51] as shown in Figure 2.

With latent representation Z , we can define the decoder to reconstruct X as:

$$X = f_2((I - A^T)^{-1}f_1(Z)) \quad (2)$$

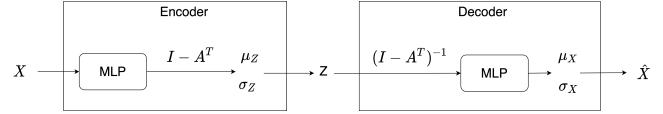


Fig. 2: Architecture of the causal structure learning method

Then, the corresponding encoder can be defined as:

$$Z = f_4((I - A^T)f_3(X)) \quad (3)$$

Combining with deep neural networks, we use multilayer perceptron (MLP) to simulate f_1 , f_2 , f_3 , and f_4 , which all are parameterized functions. Based on VAE, the output of encoder and decoder are data distributions, so we get Z by sampling from μ_Z and σ_Z , and \hat{X} by sampling from μ_X and σ_X .

For a VAE model, with a variational posterior $q(Z|X)$ to approximate the actual posterior $p(Z|X)$, evidence lower bound (ELBO) can be represented as:

$$\begin{aligned} L_{\text{ELBO}} &= E_{Z \sim q} [\log p(X|Z)] - KL(q(Z|X), p(Z)) \\ &= E_{Z \sim q} \left(-\frac{1}{2c} \|X - \hat{X}\|^2 \right) - KL(q(Z|X), p(Z)) \end{aligned} \quad (4)$$

Thus, the learning problem can be defined as:

$$\begin{aligned} \min_{A, \theta} f(A, \theta) &= -L_{\text{ELBO}} \\ \text{s.t. } h(A) &= 0 \end{aligned} \quad (5)$$

where θ is all the parameters of the VAE. For a non-linear equality-constrained problem, we can use augmented Lagrangian method [52] to solve it.

$$L_c(A, \theta, \lambda) = f(A, \theta) + \lambda h(A) + \frac{c}{2} |h(A)|^2 \quad (6)$$

where λ is the Lagrange multiplier and $c > 0$ is the penalty parameter. The following update rules are defined:

$$\begin{aligned} A^k, \lambda^k &= \arg \min_{A, \theta} L_{c^k}(A, \theta, \lambda^k) \\ \lambda^{k+1} &= \lambda^k + c^k h(A^k) \\ c^{k+1} &= \begin{cases} \eta c^k, & \text{if } |h(A^k)| > \gamma |h(A^{k-1})| \\ c^k, & \text{otherwise} \end{cases} \end{aligned} \quad (7)$$

In the augmented Lagrangian, the penalty parameter c is typically updated using an exponentially increasing function of the iteration number, and the Lagrange multiplier λ is correspondingly updated to converge to the optimal condition. The update rule for the penalty parameter c is important to balance the trade-off between feasibility and optimality in the optimization problem. The rule states that if the constraint violation at the next iteration is larger than the current violation, the value of c should be increased. Conversely, if the constraint violation at the next iteration is smaller than the current violation, the value of c should be kept the same. To achieve faster convergence and find optimal solutions, the update rule depends on two tuning parameters, η and γ . Usually, we set $\eta > 1$ to induce fast convergence and $\gamma < 1$ to limit the convergence speed [43]. If γ is set too high, the convergence will be slow, while if η is set too high, the convergence will be fast, but the results may oscillate. Parameter analysis is provided in our experiments in Section IV-B.

During training, parameters A and θ will be updated every epoch. After training, we can get the A , which is the adjacency matrix of a DAG. For root cause localization in microservice applications, we define $X = [m_1^{s_1}, m_2^{s_1}, \dots, m_i^{s_j}, \dots]$. With this causal structure learning method, we can get a weighted DAG (G) which represent causal relations between metrics as shown in Figure 1(b). Each node in G represent a metric, for example, $m_1^{s_1}$ means a metric in service s_1 . The edge from $m_1^{s_1}$ to $m_2^{s_1}$ indicates that a change in $m_1^{s_1}$ will result in a change in $m_2^{s_1}$ with the weight w . Weight w represents the degree of the impact. If w is large, it indicates that a small change in $m_1^{s_1}$ will result in a large change in $m_2^{s_1}$. Furthermore, w can be either negative or positive, implying that an increase in $m_1^{s_1}$ may result in an increase or decrease in $m_2^{s_1}$. Based on the weighted DAG, we then use a root cause inference method to pinpoint the root cause metric.

C. Root cause inference

For the weighted DAG (G), we can rank metrics with the PageRank algorithm. PageRank works according to the number of incoming edges and the probability of anomalies spreading through the graph. We define P_{ij} as the transition probability of node i to j :

$$P_{ij} = \begin{cases} \frac{w_{ij}}{\sum_j w_{ij}}, & \text{if } w_{ij} \neq 0 \\ 0, & \text{otherwise} \end{cases} \quad (8)$$

Here, w_{ij} is the weight between node i and j . We define P as the transition probability matrix. Then, we can get the PageRank vector v as proposed by [53] as:

$$v = \alpha Pv + \frac{1-\alpha}{n} \quad (9)$$

Here, n is the number of nodes, $\alpha \in (0, 1)$ is the teleportation probability, and it means that the random walk will continue with probability α and jump to a random node with probability $1-\alpha$. We use the default setting $\alpha = 0.85$ [54]. To get results of the root cause inference method better, we first reverse edges in G and use the absolute value of all weights. After running the root cause inference method, we rank the PageRank scores of all nodes and get the ranked list as shown in Figure 1(c). The higher the ranking on the list, the more likely the root cause is.

IV. EXPERIMENTS AND RESULTS

To evaluate the root cause localization framework Causal-RCA, we conduct experiments on both coarse-grained and fine-grained. As for coarse-grained root cause localization, we design experiments to identify faulty services. As for fine-grained root cause localization, we first localize root cause metrics in the faulty service. In addition, taking into account the lack of understanding of services and underlying infrastructures of an application, we provide another fine-grained experiment to localize the root cause metric with all monitoring metrics in all services. In this section, we will introduce experimental settings and experimental results.

A. Experimental settings

1) *Testbed*: To evaluate our framework, sock-shop⁶, which simulates an e-commerce website that sells socks, is deployed. It is widely used as a microservice benchmark designed to aid demonstration and test microservices and cloudnative technologies [13], [14], [21]. Sock-shop consists of 13 services, which are implemented in heterogeneous technologies and communicate via REST over HTTP. Except for communication services, it contains 7 functional services, which are, *frontend* serves as the entry of user requests; *catalogue* provides product catalogue and information; *carts* holds shopping carts; *user* stores user accounts, including payment cards and addresses; *orders* place orders of login users from carts, and it consumes memory a lot; finally, *payment* and *shipping* services are provided for orders, which require network for processing transactions.

We deploy the sock-shop with Kubernetes on several VMs in the cloud, as shown in Figure 3. In the Kubernetes cluster, we have one master node and three worker nodes. Their configurations are Ubuntu 18.04, 4vCPU, 16G RAM Memory, and 80G Disk. On the master node, we deploy open-source monitoring and visualization tools, Prometheus⁸ and Grafana⁹, respectively. Prometheus and Grafana are widely used for monitoring in microservice applications [55], [56]. Prometheus can keep monitoring the whole system and collecting both service-level and resource-level data [14]. In addition, a load generation tool, Locust¹⁰, is deployed on the master node

⁸<https://github.com/prometheus/prometheus>

⁹<https://github.com/grafana/grafana>

¹⁰<https://locust.io/>

to simulate workloads for the microservice application. On worker nodes, we deploy 13 services of the sock-shop application, and they are allocated to different VMs automatically by Kubernetes.

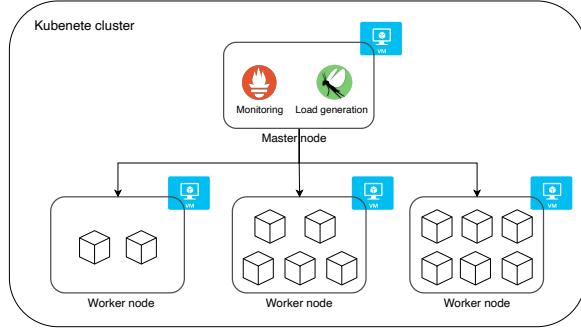


Fig. 3: The microservice application sock-shop deployed on VMs with Kubernetes

2) Anomaly injection: Microservice applications are deployed and distributed in clouds, and their performance is highly dependent on the resources of the underlying infrastructures. There are several common and widespread real performance anomalies in distributed systems [57]. Anomalous CPU consumption in VMs due to infinite loops, busy waits, or deadlocks of competing actions in applications can cause a slowdown of user requests [58]. Memory leak, one of the most prominent software bugs that severely threaten the availability and security of systems [59], happen when allocated chunks of memory are not freed after their use. Accumulations of unfreed memory may exhaust the system resource and lead to memory shortage and system failures. In addition, network resources are vulnerable to being attacked because of the frequent communication between servers and clients. Network latency anomalies usually originate from queuing or processing delays of packets on gateways [58]. The three anomalies are common and frequent in microservice application [14], [21], which will be used to evaluate our framework.

Our method can be applied to any anomaly that manifests as increased microservice response time. In this evaluation, we inject the three common anomalies: CPU hog, memory leak, and network delay. We inject CPU hog by consuming CPU resources of each service. For memory leak, we allocate memory continuously for each service. For network delay, we enable traffic control to delay the network packets. We implement anomaly injection with the tool Pumba¹¹, which can emulate network failures and stress-testing resources for Docker containers. Based on anomaly detection research [60], [61], anomalies usually last several minutes, so each anomaly of each service we injected lasts 5 minutes, and the application will have 10 minutes to cold down before another injection.

3) Data collection: We deploy Prometheus to monitor the microservice application and collect monitoring data in real-time. Prometheus is configured to collect data every 5 seconds.

We collect both service-level and resource-level data. At the service level, we collect the latency of each service. At the resource level, we collect container resource-related metrics, including CPU usage, memory usage, disk read and write, and network receive and transmit bytes, as shown in Table II.

Metrics type	Metrics
Service-level	Service latency
Resource-level	CPU usage
	Memory usage
	Disk read
	Disk write
	Network receive
	Network transmit

TABLE II: Collected monitoring metrics

4) Baseline methods: Related work in Section II shows that CI-based root cause localization uses different causal structure learning methods. Our CausalRCA is developed based on a gradient-based causal structure learning method. Therefore, to evaluate the localization performance of our CausalRCA, we design baseline methods by combining different causal structure learning methods with PageRank. We chose the constraint-based method PC, the score-based method GES, and the function-based method LiNGAM.

For these baseline structure learning methods, we use their default parameter settings in causal-learn¹². In CausalRCA, we use 2-layers MLP in the encoder and decoder, respectively. We set the learning rate as $1e - 3$, and training epochs as 1000. In addition, we train the model with the Adam optimizer. We use $\eta = 10$ and $\gamma = 0.25$ as default in our experiments, which is proven to work well in [43], and perform parameter analysis with $\eta = 100, 1000$ and $\gamma = 0.5, 0.75$. We run CausalRCA 10 times and take the average as the result of each experiment.

5) Evaluation metrics: To evaluate localization accuracy, we use two performance metrics: $AC@k$ and $Avg@k$, which are the most commonly used metrics to evaluate rank results [14]. $AC@k$ represents the probability that the top k results given by a method include the real root cause. When the k is small, a higher $AC@k$ indicates that the method is more accurate in identifying the root cause. For each service and each anomaly type, $AC@k$ is calculated as follows:

$$AC@k = \frac{\sum_{i < k} R[i] \in V_{rc}}{\min(k, |V_{rc}|)} \quad (10)$$

where $R[i]$ is the result after ranking all metrics. V_{rc} is the root cause set. $Avg@k$ evaluates the overall performance of a method by computing the average $AC@k$, which is defined as:

$$Avg@k = \frac{1}{k} \sum_{1 \leq j \leq k} AC@j \quad (11)$$

We use $AC@1$, $AC@3$, and $Avg@5$ in our experiments. $AC@1$ evaluates if the top localized root cause is the real one, and it is the most restrictive and accurate metric. $AC@3$ is used to determine if the top three localized results have the real root cause. This metric is less accurate than $AC@1$, but it can

¹¹<https://github.com/alexei-led/pumba>

¹²<https://github.com/cmu-phil/causal-learn>

still help operators quickly reduce root-cause candidates and localize the real ones. Finally, $\text{Avg}@5$ represents the average localization ability. The three metrics are commonly used in the root cause localization task, and they can fairly evaluate localization performance [14], [27].

6) *Statistical testing*: To assess the statistical significance of different RCA methods, we use the one-way analysis of variance (ANOVA) to test the difference between all RCA methods and the t-test to check the pairwise differences [62]. We use $\text{Avg}@5$ as the performance score of each RCA method. ANOVA is a hypothesis-testing framework for determining whether the between-group variation is significant. The F-statistic, calculated as the ratio of the between-group variation to the within-group variation, is used in ANOVA. The p-value associated with the F-statistic indicates the probability of obtaining an F-statistic as extreme as the observed one, assuming the null hypothesis is true. If the p-value is less than the significance level (usually 0.05), we reject the null hypothesis and conclude that there is a statistically significant difference among the RCA methods.

If the ANOVA test indicates a significant performance difference among these methods, we then use a t-test to determine the differences between each pair of RCA methods. The basic idea behind the t-test is to calculate a test statistic called the t-value, which measures the difference between the average performance scores of two methods relative to the variability within each method. Once the t-value has been calculated, it is compared to a critical value from a t-distribution. If the t-value is less than the critical value, we fail to reject the null hypothesis and conclude that there is insufficient evidence to suggest a performance difference between the two methods.

B. Experimental results

We provide the results of three experiments as below: noitemsep, nolistsep

- Coarse-grained faulty service localization based on service latency of all services.
- Fine-grained root cause metric localization in the faulty service based on system-level metrics in the faulty service.
- Fine-grained root cause metric localization with all monitoring metrics in all services

We compare the localization performance of CausalRCA with baseline methods and explain the results.

1) *Coarse-grained faulty service localization*: We evaluate the performance of CausalRCA on localizing the faulty service that initiates performance anomalies. This localization is conducted based on service-level data, which is the latency of all services. Table III shows the localization accuracy compared with baseline methods for different anomalies. We can see that, when compared to baseline methods, CausalRCA has improved localization accuracy in terms of $AC@1$, $AC@3$, and $\text{Avg}@5$ in different anomalies by up to 10%. In addition, for CPU hog, causalRCA has the best performance in terms of $AC@1$, $AC@3$, and $\text{Avg}@5$. The $AC@3$ is 0.7175, which means that there is a 71.75% chance of finding the root

cause in the top three metrics on the ranked list, which is slightly higher than the LiNGAM-based method. For memory leak, CausalRCA continues to outperform in terms of $AC@1$, $AC@3$, and $\text{Avg}@5$. There is a 62.14% possibility of localizing the root cause in the top 3 metrics. For network delay, $AC@3$ is not good enough, but $AC@1$ and $\text{Avg}@5$ are higher than baseline methods. In general, the average $AC@1$ of CausalRCA is 0.2, which means that there is an average 20% possibility that the top 1 metric on the ranked list can be identified as the root cause. The averages $AC@3$ and $\text{Avg}@5$ of CausalRCA for the three anomalies are 0.5749 and 0.5815, respectively. The increase of average $\text{Avg}@5$ is 6.72%, showing the improvement in localizing accuracy compared with baseline methods. We provide statistical testing to show the significant difference between these RCA methods. We obtained a p-value of 0.0003 using the ANOVA method first, showing a significant performance difference between the four RCA methods. We further utilized t-tests to compare the performance differences between each pair of methods, and the resulting p-values are shown in Figure 4. We can see that CausalRCA has a significant difference from baseline methods, while PC-based and GES-based methods have no significant difference.

TABLE III: Localization accuracy of CI-based methods on localizing faulty services (Coarse-grained) for different anomalies

Methods	PC-based	GES-based	LINGAM-based	CausalRCA	Increase
CPU hog					
AC@1	0.1429	0.1429	0.1429	0.1873	4.44%
AC@3	0.2857	0.4286	0.7143	0.7175	0.32%
Avg@5	0.4286	0.4	0.5714	0.6244	5.3%
Memory leak					
AC@1	0	0	0.1429	0.2429	10%
AC@3	0.4286	0.1429	0.5714	0.6214	5%
Avg@5	0.4286	0.2286	0.5429	0.6143	7.14%
Network delay					
AC@1	0.1429	0	0.1429	0.1714	2.85%
AC@3	0.5714	0	0.4286	0.3857	-
Avg@5	0.4857	0.1714	0.4286	0.5057	2%
Average Avg@5	0.4476	0.2667	0.5143	0.5815	6.72%

We analyze the impact of parameters γ and η in causal structure learning on the root cause localization performance of CausalRCA. We use $\gamma = 0.25$ and $eta = 10$ as default, and also set $\gamma = 0.5, 0.75$, and $\eta = 100, 1000$. The results can be found in Figure 5. We can see that $\gamma = 0.25, \eta = 10$ has the best performance in CPU hog and Network latency, and it also has the best average performance of different anomalies. In addition, $\gamma = 0.75, \eta = 10$ performs best for memory leak, because a high γ can prevent too fast convergence and find better solutions, but it usually takes more time. Furthermore, we can see that $\gamma = 0.25, \eta = 1000$ performs poorly on CPU hog anomaly, but relatively well on memory leak anomaly, suggesting that a high η can lead to more variance in localization results.

We then provide a detailed performance of CausalRCA on localizing faulty services with different anomalies in Figure 6. For CPU hog in Figure 6(a), we can see that localization accuracy performs well on services except *payment*, because *payment* is not a CPU-intensive service. For the memory

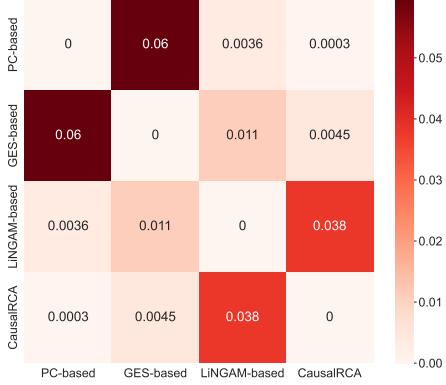


Fig. 4: P-value of RCA methods (Coarse-grained experiment)

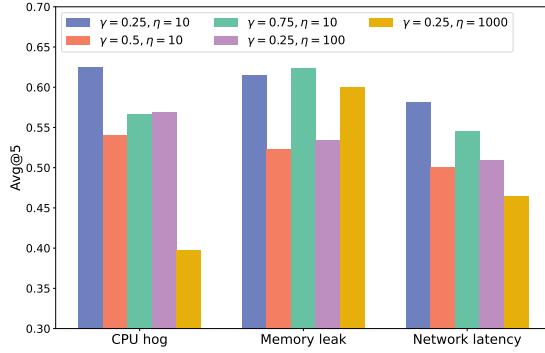


Fig. 5: Localization accuracy with different γ and η (Coarse-grained experiment)

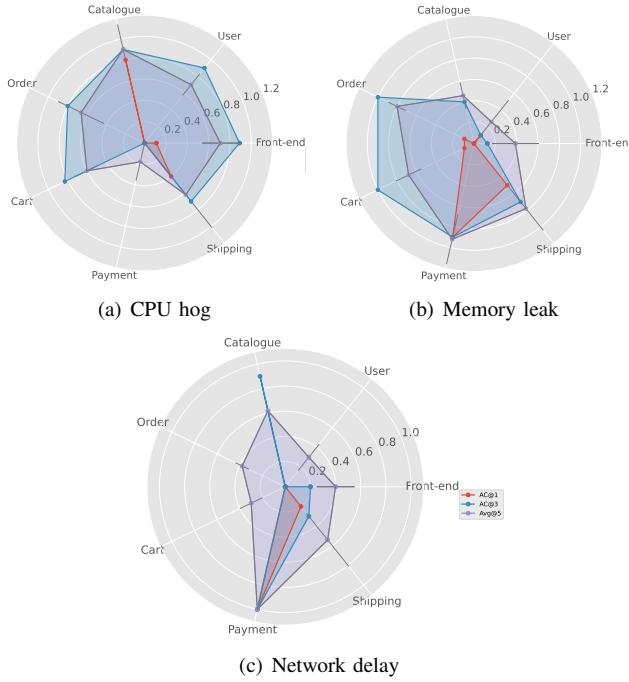


Fig. 6: Performance of CausalRCA on localizing faulty services with different anomalies

leak in Figure 6(b), we can see that *front-end*, *user*, and *catalogue* perform worse than other services. The memory leak issues in these services do not affect their service latency much, making it difficult to identify cause-effect relations between services. In terms of network delay in Figure 6(c), only service *payment* performs well, which explains the poor average localization performance of *AC@3* in Table III. *Payment* service relies heavily on the network, making it easy to localize the network delay issue. We plot the errorbar for *Avg@5* in Figure 6 to represent the variation of our results, and we can see that the standard deviations of many results are not high.

2) *Fine-grained root cause metric localization in the faulty service*: Given the faulty service, we apply CausalRCA to container resource metrics and evaluate its performance on localization accuracy for different anomalies. Table IV shows the localization accuracy of CausalRCA on localizing root cause metric in faulty service compared with baseline methods. For different anomalies, we can see that CausalRCA has improved localization accuracy compared with baseline methods in terms of *AC@1*, *AC@3*, and *Avg@5* by up to 14.29%. In addition, for CPU hog, CausalRCA has the best performance in terms of *AC@3* and *Avg@5*, while the *AC@1* is worse than PC-based methods. For memory leak, the *AC@1* of CausalRCA is worse than LiNGAM-based method, but it still has the best *AC@3* and *Avg@5*. Finally, for network delay, CausalRCA outperforms in terms of *AC@1*, *AC@3*, and *Avg@5*. In general, the average *AC@1* of CausalRCA is 0.2476, which means there is about a 25% possibility of determining the top 1 metric on the ranked list as the root cause. For the three anomalies, the average *AC@3* is 0.719, which means there is a 71.9% possibility to localize the root cause metric in the top 3 metrics on the ranked list, and the average improvement is 10% compared with baseline methods. The average *Avg@5* of CausalRCA is 0.6681, and the average increase is 9.43%. We consider the outperformance of CausalRCA is because resource metrics, such as CPU/memory usage, affect each other, which makes it easier to identify anomaly propagation with CI methods.

TABLE IV: Localization accuracy of CI-based methods on localizing root cause metrics (Fine-grained) in faulty services for different anomalies

Methods	PC-based	GES-based	LiNGAM-based	CausalRCA	Increase
CPU hog	0.4286	0.1429	0	0.2286	-
	0.4286	0.5714	0.7143	0.7286	14.3%
	0.4286	0.6	0.5714	0.67	7%
Memory leak	0	0	0.4286	0.2714	-
	0.1429	0.4286	0.5714	0.7143	14.29%
	0.3429	0.4	0.6286	0.6771	4.85%
Network delay	0	0.1429	0.1429	0.2429	10%
	0.2857	0.5714	0.4286	0.7143	14.29%
	0.2214	0.5143	0.5214	0.6571	13.57%
	Average Avg@5	0.33	0.5048	0.5738	0.6681

We also provide statistical testing to show the significant difference of these RCA methods. We first obtained a p-value of 0.0013 using the ANOVA method, showing a significant performance difference between the four RCA methods. The

p-values obtained from t-tests are shown in Figure 7. We can see that CausalRCA has a significant difference from baseline methods. In comparison, GES-based method has no significant difference with PC-based and LiNGAM-based methods.

We evaluate the impact of parameters γ and η and present the findings in Figure 8. The results indicate that $\gamma = 0.25$ and $\eta = 10$ perform the best in identifying CPU hog anomalies, while $\gamma = 0.25$ and $\eta = 100$ are most effective in detecting memory leak and network latency anomalies. In addition, $\gamma = 0.5$ and $\eta = 10$ outperform $\gamma = 0.5$ and $\eta = 100$ in detecting network latency anomalies and perform better than $\gamma = 0.75$ and $\eta = 10$ across all three types of anomalies. Overall, $\gamma = 0.25$ and $\eta = 10$ have the highest average localization performance. However, increasing γ and η could potentially lead to better solutions and improve localization accuracy.

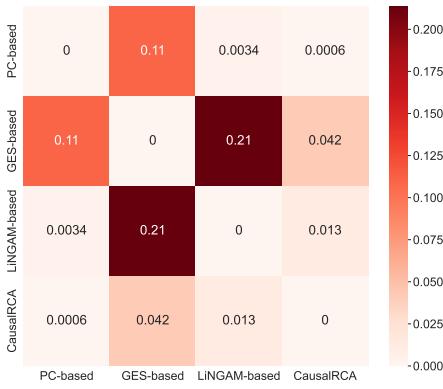


Fig. 7: P-value of RCA methods (Fine-grained experiment)

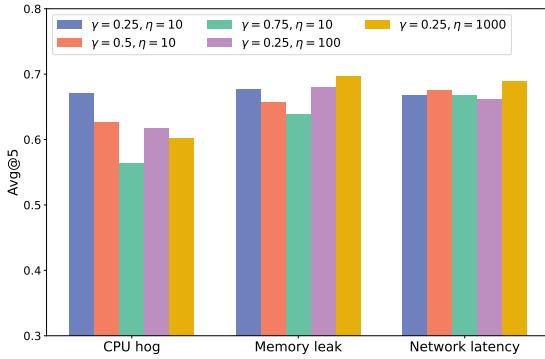
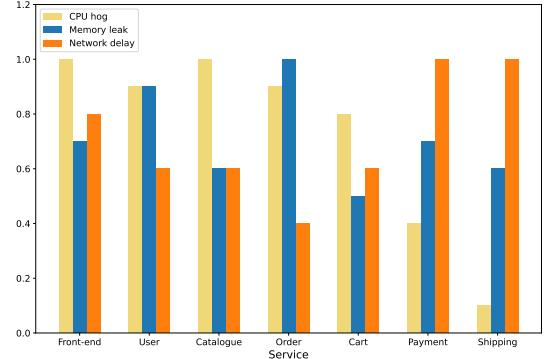
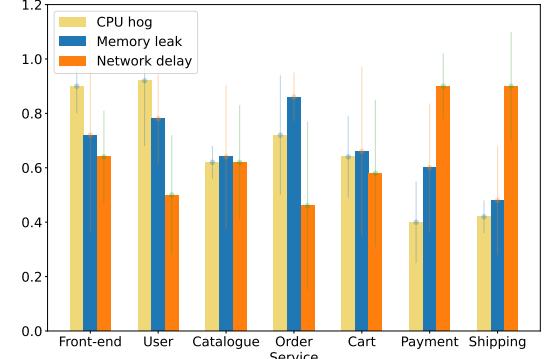


Fig. 8: Localization accuracy with different γ and η (Fine-grained experiment)

We then provide the performance of CausalRCA on localizing root cause metrics in faulty services with the three anomalies in Figure 9. We can see that $AC@3$ and $AC@5$ have consistent performance. For CPU hog, we can see that localization accuracy is low for *payment* and *shipping* services because they are insensitive to CPU resources. Because the memory leak issue manifests in multiple resource metrics, all services perform well for the memory leak. *Order service* has the best performance because it is highly related to memory usage. For network delay, we can see that *payment*



(a) $AC@3$



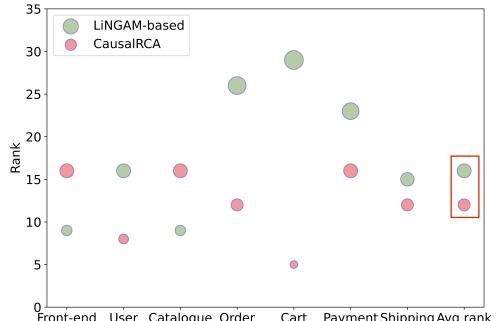
(b) $Avg@5$

Fig. 9: Performance of CausalRCA on localizing root cause metrics in faulty services with different anomalies

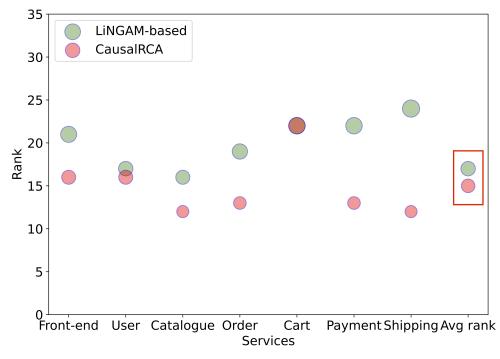
and *shipping* have the best performance because they rely heavily on the network. We plot the errorbar for $Avg@5$ in Figure 9(b), and we can see there are some variances in the localization results, maybe caused by the dynamic nature of cloud environments or random fluctuation of resources in services, showing that the generality of CausalRCA can be explored more in the future.

3) Fine-grained root cause metric localization with all monitoring metrics:

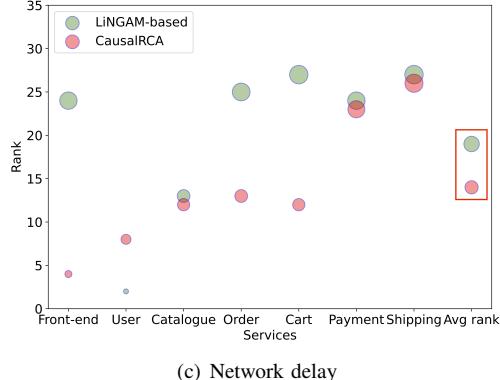
Considering that we are unknown to services and underlying infrastructures of an application, we conduct the fine-grained root cause localization with all monitoring metrics. We apply CausalRCA on all monitoring metrics to localize the root cause metric. We mainly show the ranks of comparison between the LiNGAM-based method and CausalRCA, because PC sometimes fails to extract causal relations between metrics, while GES takes too long to build a causal graph with too many nodes. For this fine-grained root cause localization, it is hard to identify the root cause metric in the top 1 or top 3 metrics, so we use the rank of root cause metrics to evaluate localization performance as shown in Figure 10. We can see that the average rank of CausalRCA is about 13, which is lower than the LiNGAM-based method. The result shows that CausalRCA has better localization performance than the



(a) CPU hog



(b) Memory leak



(c) Network delay

Fig. 10: Ranks of root cause metrics identified by CI-based methods

LiNGAM-based method. However, it is still challenging to extract causal relations between metrics and pinpoint the root cause metric from multiple observable metrics. We apply the t-test to LiNGAM-based and CausalRCA methods and obtain the p-value of 0.0335, showing the significant difference between them. The impact of parameters γ and η is presented in Figure 11. The results indicate that $\gamma = 0.25, \eta = 10$ performs the best in identifying CPU hog and network latency anomalies. For the memory leak anomaly, higher γ and η have better localization accuracy, and $\gamma = 0.75, \eta = 10$ is most effective. On average, the $\gamma = 0.25, \eta = 10$ has the best localization performance, while adjusting γ and η for the memory leak anomaly can improve localization accuracy effectively.

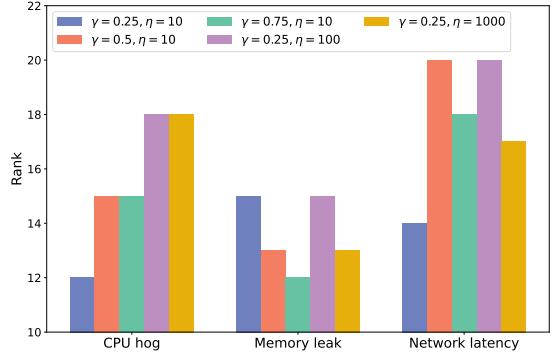


Fig. 11: Ranks of root cause metrics with different parameters γ and η

In conclusion, CausalRCA has a significant difference from baseline methods and better localization accuracy for coarse-grained and fine-grained root cause localization. In addition, we find that $\gamma = 0.25, \eta = 10$ in CausalRCA has the best localization performance on average. However, adjusting parameters can provide more potential for improving localization accuracy. Based on CI methods, we can see that anomaly propagation performs differently in different services; for example, the *payment* service is sensitive to the network delay issue but nonsensitive to the CPU hog issue. As for fine-grained root cause localization, it is still challenging to pinpoint the root cause metric in all monitoring metrics. Therefore, it is more practical to consider the drill-down localization, i.e., identify the faulty service first and then determine the root cause metric in the faulty service.

C. Threats to validity

We analyze threats to our framework from the four categories: construct, internal, conclusion, and external validity based on [63]. The **construct threat** to validity mainly lies in the hyperparameters and evaluation metrics. We provide parameter analysis for two hyperparameters in CausalRCA, and results show that default parameters work well as provided in [43] but tuning parameters carefully has the potential of improving localization accuracy. In addition, we use widely used evaluation metrics and provide statistical testing to evaluate the performance difference of different RCA methods.

The **internal threat** to validity mainly lies in the implementation of the framework, as errors or bugs in the implementation could affect the accuracy of the results. To reduce it, we have used established Python packages and conducted thorough testing. We have also repeated the experiments multiple times to ensure the reliability and consistency of our results. The **conclusion threat** to validity of our framework is related to the types of anomalies used in experiments. As microservice applications have a variety of performance anomalies that can affect the localization results [57], we injected three different types of common and frequent anomalies to evaluate the effectiveness of our framework. We report and discuss the localization results for each individual anomaly type, and the

experimental results demonstrate the superior performance of our framework on these anomalies.

The **external threat** relies on the configuration of microservice applications and the data collection strategies. In this paper, we investigate a specific configuration of a microservice application to evaluate the performance of CausalRCA, which may limit the generality of our framework. However, building complex infrastructures and repeating the experiments on multiple testbeds is extremely expensive, which is impractical for our experiments. In addition, the benchmark microservice application sock-shop is widely used in academia to aid the testing of microservices in clouds [14], [21], and it helps us mitigate this threat. On the other hand, the localization performance of our framework heavily relies on input data. To mitigate the threat, we adopt Prometheus, an open-source tool for real-time monitoring, and collect service-level and resource-level metrics that present the status of a running microservice application. Currently, our framework performs well with anomalies injected over fixed time range anomalies, but the effect of different time ranges for CausalRCA can be explored more.

V. DISCUSSION

This paper provides a framework called CausalRCA for root cause localization of microservice applications. The framework is developed with CI-based methods, including causal structure learning and root cause inference. We provide coarse-grained and fine-grained experiments to evaluate the localization performance of the framework. Our experimental results show that the framework has the best localization accuracy compared with baseline methods. However, some aspects can still be improved.

Our experiments show that CausalRCA performs well on localizing faulty services and root cause metrics in faulty services. However, localizing root cause metrics from all monitoring metrics is very hard. The average rank of root cause metrics is out of ten. We consider the improvement of localization accuracy can be researched more. First, data preprocessing, such as feature reduction, can be considered to reduce training time and improve localization accuracy. Next, we apply a gradient-based method to learn causal structures. The gradient-based method is applied to time-series monitoring data, which may ignore time lags in the original data. We consider that time lags in the data may help improve causal structure learning. For the root cause inference method PageRank, a personalized PageRank [64], which considers the preferences of nodes, can be applied.

This paper mainly focuses on monitoring data to implement root cause localization. Monitoring data has multi-dimensional information and is easy to collect compared with trace and log data. However, trace data and log data contain accurate deployment information and service interactions, which can be used to calibrate the causal graph generated based on monitoring data. At the same time, the causal graph generated with CI methods can extract hidden relations between metrics.

Therefore, we can consider combining different data resources to improve localization accuracy in the future.

This paper mainly focuses on improving localization accuracy of microservices, while efficiency is also important for achieving fast recovery. For the sock-shop benchmark application in this paper, we roughly estimate the time spent of our framework takes tens seconds, showing the cost of time may be lower than service migration [65]. However, for large-scale microservice, e.g., hundreds/thousands of services, the time cost for building the causality graph may be far greater than service migration. We will test the scalability and exact time cost of our framework and pay more attention to reducing training time in the future. For now, we mainly use the data collected in five minutes after the anomaly is detected. In the future, we will consider testing localization performance with different time ranges based on our CausalRCA.

VI. CONCLUSION

This paper tackles the challenge of localizing root causes of performance anomalies in microservice applications. Root cause localization can be used to help operators achieve fast recovery of microservice applications. Therefore, it is important to guarantee localization accuracy at first. In addition, fine-grained root cause localization, which means identifying both the faulty service and resource related metrics in a faulty service, is necessary. With monitoring data, we provide a CI-based framework named CausalRCA, which can automate localizing root causes with fine granularity and in real time.

The CausalRCA works with causal structure learning and root cause inference components. For causal structure learning, we propose a GNN-based method that uses a deep generative model and applies a variant of the structural constraint to learn the weighted DAG. Compared with other CI methods, the gradient-based method can extract non-linear causal relations between metrics. For root cause inference, we apply PageRank to visit the weighted DAG and return a ranked list of all metrics. We then provide experiments to evaluate the localization performance of CausalRCA.

In our experiments, we conduct three types of experiments: coarse-grained faulty service localization, fine-grained root cause metrics localization in faulty services, and fine-grained root cause metrics localization with all monitoring metrics. Our experimental results show that CausalRCA has better localization accuracy compared with baseline methods. Furthermore, based on CI methods, we can see that anomaly propagation performs differently between services, which gives operators a better understanding of microservice applications. In addition, it is difficult for fine-grained root cause localization with all monitoring metrics because anomalous metrics manifest diverse symptoms in different services. Therefore, fine-grained root cause localization with all monitoring metrics still needs to be improved. However, for microservice applications, we can still consider the drill-down localization, first identifying the faulty service, then pinpointing the root cause metric in the faulty service to identify fine-grained causes.

In the future, we will continue to improve the localization performance of CausalRCA. Hyperparameters tuning can be tested more in the future. The causal structure learning can consider time lag in monitoring data, and the root cause inference can be improved by adding the preferences of nodes. In addition, employing knowledge from trace and log data to calibrate the causal graph may improve localization accuracy and make the causal graph more reasonable. Finally, localization efficiency to achieve fast recovery needs to be tested and improved.

VII. ACKNOWLEDGE

This research work is funded by the EU Horizon 2020 and Horizon Europe research and innovation program under grant agreements 825134 (ARTICONF project), 862409 (BlueCloud project), 824068 (ENVRIFAIR project) and 101094227 (BlueCloud 2026). The work is also supported by LifeWatch ERIC and partially funded by the Science and Technology Program of Sichuan Province under Grant 2020JDRC0067 and 2020YFG0326.

REFERENCES

- [1] A. Balalaie, A. Heydarnoori, P. Jamshidi, Microservices architecture enables devops: Migration to a cloud-native architecture, *Ieee Software* 33 (3) (2016) 42–52.
- [2] R. Xin, H. Liu, P. Chen, Z. Zhao, Robust and accurate performance anomaly detection and prediction for cloud applications: a novel ensemble learning-based framework, *Journal of Cloud Computing* 12 (1) (2023) 1–16.
- [3] Z. Li, N. Zhao, M. Li, X. Lu, L. Wang, D. Chang, X. Nie, L. Cao, W. Zhang, K. Sui, et al., Actionable and interpretable fault localization for recurring failures in online service systems, *arXiv preprint arXiv:2207.09021* (2022).
- [4] B. Gregg, Systems performance: enterprise and the cloud, Pearson Education, 2014.
- [5] O. Ibdumoye, F. Hernández-Rodríguez, E. Elmroth, Performance anomaly detection and bottleneck identification, *ACM Computing Surveys (CSUR)* 48 (1) (2015) 1–35.
- [6] Z. Chen, Y. Kang, L. Li, X. Zhang, H. Zhang, H. Xu, Y. Zhou, L. Yang, J. Sun, Z. Xu, et al., Towards intelligent incident management: why we need it and how we make it, in: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1487–1497.
- [7] P. Chen, H. Liu, R. Xin, T. Carval, J. Zhao, Y. Xia, Z. Zhao, Effectively detecting operational anomalies in large-scale iot data infrastructures by using a gan-based predictive model, *The Computer Journal* 65 (11) (2022) 2909–2925.
- [8] Y. Song, R. Xin, P. Chen, R. Zhang, J. Chen, Z. Zhao, Identifying performance anomalies in fluctuating cloud environments: A robust correlative-gnn-based explainable approach, *Future Generation Computer Systems*Doi: 10.1016/j.future.2023.03.020 (2023).
- [9] P. Chen, Y. Qi, P. Zheng, D. Hou, Causeinfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems, in: *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, IEEE, 2014, pp. 1887–1895.
- [10] C. Wu, N. Zhao, L. Wang, X. Yang, S. Li, M. Zhang, X. Jin, X. Wen, X. Nie, W. Zhang, et al., Identifying root-cause metrics for incident diagnosis in online service systems, in: *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, IEEE, 2021, pp. 91–102.
- [11] L. Wu, J. Tordsson, A. Acker, O. Kao, Microras: Automatic recovery in the absence of historical failure data for microservice systems, in: *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, IEEE, 2020, pp. 227–236.
- [12] C. Engelbert, Rebooting your services for monitoring is so 2015, <https://www.instana.com/blog/rebooting-your-services-for-monitoring-is-so-2015/>, August 3, 2020.
- [13] L. Wu, J. Tordsson, E. Elmroth, O. Kao, Causal inference techniques for microservice performance diagnosis: Evaluation and guiding recommendations, in: *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, IEEE, 2021, pp. 21–30.
- [14] L. Wu, J. Tordsson, E. Elmroth, O. Kao, Microrca: Root cause localization of performance issues in microservices, in: *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, IEEE, 2020, pp. 1–9.
- [15] Z. Li, N. Zhao, S. Zhang, Y. Sun, P. Chen, X. Wen, M. Ma, D. Pei, Constructing large-scale real-world benchmark datasets for aiops, *arXiv preprint arXiv:2208.03938* (2022).
- [16] P. Vergadia, Introduction to google cloud's operations suite, <https://cloud.google.com/blog/topics/developers-practitioners/introduction-google-clouds-operations-suite/>, Dec 3, 2021.
- [17] J. Soldani, A. Brogi, Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey, *ACM Computing Surveys (CSUR)* 55 (3) (2022) 1–39.
- [18] P. Aggarwal, A. Gupta, P. Mohapatra, S. Nagar, A. Mandal, Q. Wang, A. Paradkar, Localization of operational faults in cloud applications by mining causal dependencies in logs using golden signals, in: *International Conference on Service-Oriented Computing*, Springer, 2020, pp. 137–149.
- [19] M. Kim, R. Sumbaly, S. Shah, Root cause detection in a service-oriented architecture, *ACM SIGMETRICS Performance Evaluation Review* 41 (1) (2013) 93–104.
- [20] J. Weng, J. H. Wang, J. Yang, Y. Yang, Root cause analysis of anomalies of multitier services in public clouds, *IEEE/ACM Transactions on Networking* 26 (4) (2018) 1646–1659.
- [21] J. Lin, P. Chen, Z. Zheng, Microscope: Pinpoint performance issues with causal graphs in micro-service environments, in: *International Conference on Service-Oriented Computing*, Springer, 2018, pp. 3–20.
- [22] Z. Guan, J. Lin, P. Chen, On anomaly detection and root cause analysis of microservice systems, in: *International Conference on Service-Oriented Computing*, Springer, 2018, pp. 465–469.
- [23] P. Wang, J. Xu, M. Ma, W. Lin, D. Pan, Y. Wang, P. Chen, Cloudranger: Root cause identification for cloud native systems, in: *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, IEEE, 2018, pp. 492–502.
- [24] M. Ma, W. Lin, D. Pan, P. Wang, Ms-rank: Multi-metric and self-adaptive root cause diagnosis for microservice applications, in: *2019 IEEE International Conference on Web Services (ICWS)*, IEEE, 2019, pp. 60–67.
- [25] M. Ma, W. Lin, D. Pan, P. Wang, Self-adaptive root cause diagnosis for large-scale microservice architecture, *IEEE Transactions on Services Computing* (2020).
- [26] M. Ma, J. Xu, Y. Wang, P. Chen, Z. Zhang, P. Wang, Automap: Diagnose your microservice-based web applications automatically, in: *Proceedings of The Web Conference 2020*, 2020, pp. 246–258.
- [27] Y. Meng, S. Zhang, Y. Sun, R. Zhang, Z. Hu, Y. Zhang, C. Jia, Z. Wang, D. Pei, Localizing failure root causes in a microservice through causality inference, in: *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, IEEE, 2020, pp. 1–10.
- [28] L. Wu, J. Tordsson, J. Bogatinovski, E. Elmroth, O. Kao, Microdiag: Fine-grained performance diagnosis for microservice systems, in: *2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*, IEEE, 2021, pp. 31–36.
- [29] P. Chen, Y. Qi, D. Hou, Causeinfer: Automated end-to-end performance diagnosis with hierarchical causality graph in cloud environment, *IEEE transactions on services computing* 12 (2) (2016) 214–230.
- [30] P. Spirtes, C. N. Glymour, R. Scheines, D. Heckerman, *Causation, prediction, and search*, MIT press, 2000.
- [31] S. Shimizu, T. Inazumi, Y. Sogawa, A. Hyvärinen, Y. Kawahara, T. Washio, P. O. Hoyer, K. Bollen, Directlingam: A direct method for learning a linear non-gaussian structural equation model, *The Journal of Machine Learning Research* 12 (2011) 1225–1248.
- [32] A. Gholami, A. K. Srivastava, Comparative analysis of ml techniques for data-driven anomaly detection, classification and localization in distribution system, in: *2020 52nd North American Power Symposium (NAPS)*, IEEE, 2021, pp. 1–6.
- [33] L. Wang, N. Zhao, J. Chen, P. Li, W. Zhang, K. Sui, Root-cause metric location for microservice systems via log anomaly detection, in: *2020 IEEE International Conference on Web Services (ICWS)*, IEEE, 2020, pp. 142–150.

- [34] C. S. Calude, G. Longo, The deluge of spurious correlations in big data, *Foundations of science* 22 (3) (2017) 595–612.
- [35] Á. Brandón, M. Solé, A. Huélamo, D. Solans, M. S. Pérez, V. Muntés-Mulero, Graph-based root cause analysis for service-oriented and microservice architectures, *Journal of Systems and Software* 159 (2020) 110432.
- [36] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th international conference on evaluation and assessment in software engineering, 2014, pp. 1–10.
- [37] S. Tuli, G. Casale, N. R. Jennings, Pregan: Preemptive migration prediction network for proactive fault-tolerant edge computing, in: IEEE INFOCOM 2022-IEEE Conference on Computer Communications, IEEE, 2022, pp. 670–679.
- [38] J. Peters, D. Janzing, B. Schölkopf, Elements of causal inference: foundations and learning algorithms, The MIT Press, 2017.
- [39] K. Sachs, O. Perez, D. Pe'er, D. A. Lauffenburger, G. P. Nolan, Causal protein-signaling networks derived from multiparameter single-cell data, *Science* 308 (5721) (2005) 523–529.
- [40] D. M. Chickering, Optimal structure identification with greedy search, *Journal of machine learning research* 3 (Nov) (2002) 507–554.
- [41] S. Shimizu, P. O. Hoyer, A. Hyvärinen, A. Kerminen, M. Jordan, A linear non-gaussian acyclic model for causal discovery., *Journal of Machine Learning Research* 7 (10) (2006).
- [42] X. Zheng, B. Aragam, P. K. Ravikumar, E. P. Xing, Dags with no tears: Continuous optimization for structure learning, *Advances in Neural Information Processing Systems* 31 (2018).
- [43] Y. Yu, J. Chen, T. Gao, M. Yu, Dag-gnn: Dag structure learning with graph neural networks, in: International Conference on Machine Learning, PMLR, 2019, pp. 7154–7163.
- [44] P. Upadhyaya, K. Zhang, C. Li, X. Jiang, Y. Kim, Scalable causal structure learning: New opportunities in biomedicine, arXiv preprint arXiv:2110.07785 (2021).
- [45] A. Demir, T. Koike-Akino, Y. Wang, D. Erdoğmuş, Eeg-gat: Graph attention networks for classification of electroencephalogram (eeg) signals, in: 2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC), IEEE, 2022, pp. 30–35.
- [46] S. Beamer, K. Asanovic, D. Patterson, Direction-optimizing breadth-first search, in: SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, IEEE, 2012, pp. 1–10.
- [47] F. Spitzer, Principles of random walk, Vol. 34, Springer Science & Business Media, 2001.
- [48] C. Ridings, M. Shishigin, Pagerank uncovered, Technical Paper for the Search Engine Optimization Online Community (2002).
- [49] J. Bruna, W. Zaremba, A. Szlam, Y. LeCun, Spectral networks and locally connected networks on graphs, arXiv preprint arXiv:1312.6203 (2013).
- [50] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:1609.02907 (2016).
- [51] D. P. Kingma, M. Welling, Auto-encoding variational bayes, arXiv preprint arXiv:1312.6114 (2013).
- [52] D. P. Bertsekas, Nonlinear programming, *Journal of the Operational Research Society* 48 (3) (1997) 334–334.
- [53] L. Page, S. Brin, R. Motwani, T. Winograd, The pagerank citation ranking: Bringing order to the web, Tech. rep., Stanford InfoLab (1999).
- [54] L. Beccetti, C. Castillo, The distribution of pagerank follows a power-law only for particular values of the damping factor, in: Proceedings of the 15th international conference on World Wide Web, 2006, pp. 941–942.
- [55] M. Waseem, P. Liang, M. Shahin, A. Di Salle, G. Márquez, Design, monitoring, and testing of microservices systems: The practitioners' perspective, *Journal of Systems and Software* 182 (2021) 111061.
- [56] O. Ninio, Practical guide on setting up prometheus and grafana for monitoring your microservices, <https://komodor.com/blog/setting-up-prometheus-and-grafana-for-monitoring-your-microservices/>, Sep 28, 2022.
- [57] L. Mariani, C. Monni, M. Pezzé, O. Riganelli, R. Xin, Localizing faults in cloud systems, in: 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST), IEEE, 2018, pp. 262–273.
- [58] C. Sauvanaud, K. Lazri, M. Kaâniche, K. Kanoun, Anomaly detection and root cause localization in virtual network functions, in: 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2016, pp. 196–206.
- [59] H. Kang, H. Chen, G. Jiang, Peerwatch: a fault detection and diagnosis tool for virtualized consolidation systems, in: Proceedings of the 7th international conference on Autonomic computing, 2010, pp. 119–128.
- [60] H. Jayathilaka, C. Krintz, R. Wolski, Detecting performance anomalies in cloud platform applications, *IEEE Transactions on Cloud Computing* 8 (3) (2018) 764–777.
- [61] T. Huang, P. Chen, R. Li, A semi-supervised vae based active anomaly detection framework in multivariate time series for online systems, in: Proceedings of the ACM Web Conference 2022, 2022, pp. 1797–1806.
- [62] J. Demšar, Statistical comparisons of classifiers over multiple data sets, *The Journal of Machine learning research* 7 (2006) 1–30.
- [63] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer Science & Business Media, 2012.
- [64] G. Jeh, J. Widom, Scaling personalized web search, in: Proceedings of the 12th international conference on World Wide Web, 2003, pp. 271–279.
- [65] X. Chen, Y. Bi, X. Chen, H. Zhao, N. Cheng, F. Li, W. Cheng, Dynamic service migration and request routing for microservice in multi-cell mobile edge computing, *IEEE Internet of Things Journal* (2022).