Microservice Root Cause Analysis With Limited Observability Through Intervention Recognition in the Latent Space

Zhe Xie
Tsinghua University, BNRist
China, Beijing
xiez22@mails.tsinghua.edu.cn

Shenglin Zhang
Nankai University
China, Tianjin
zhangsl@nankai.edu.cn

Yitong Geng
eBay Inc.
China, Shanghai
eatongeng@163.com

Yao Zhang
eBay Inc.
China, Shanghai
yaozhang@ebay.com

Minghua Ma
Microsoft
USA, Redmond
minghuama@microsoft.com

Xiaohui Nie
Computer Network Information
Center, Chinese Academy of Sciences
China, Beijing
xhnie@cnic.cn

Zhenhe Yao
Tsinghua University, BNRist
China, Beijing
yaozh20@mails.tsinghua.edu.cn

Longlong Xu
Tsinghua University, BNRist
China, Beijing
xull23@mails.tsinghua.edu.cn

Yongqian Sun
Nankai University
China, Tianjin
sunyongqian@nankai.edu.cn

Wentao Li
eBay Inc.
China, Shanghai
wenli@ebay.com

Dan Pei[*]
Tsinghua University, BNRist
China, Beijing
peidan@tsinghua.edu.cn

ABSTRACT

Many failure root cause analysis (RCA) algorithms for microservices have been proposed with the widespread adoption of microservices systems. Existing algorithms generally focus on RCA with ranking single-level (*e.g.* metric-level or service-level) root cause candidates (RCCs) with comprehensive monitoring metrics. However, many heterogeneous RCCs exist with limited observability in real-world microservices systems. Further, we find that the limited observability may result in inaccurate RCA through real-world failures in eBay. In this paper, for the first time, we propose to "model RCCs as latent variables". The core idea is to infer the status of RCCs as latent variables with related monitoring metrics instead of directly extracting features from only the observable metrics. Based on this, we propose LatentScope, an unsupervised RCA framework with heterogeneous RCCs under limited observability. A dual-space graph is proposed to model both observable and unobservable variables, with many-to-many relationships between spaces. To achieve fast inference of latent variables and RCA, we propose the LatentRegressor algorithm, which includes Regression-based Latent-space Intervention Recognition (RLIR) to achieve intervention recognition-based RCA in latent space. LatentScope has been deployed in eBay's production environment and evaluated on both eBay's real-world failures and a testbed dataset. The evaluation results show that, compared with baseline algorithms, our model significantly improves the Top-1 recall by 9.7% − 57.9%. The source code of LatentScope and the dataset are available at https://github.com/NetManAIOps/LatentScope.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**; • **Software and its engineering → Maintaining software**.

KEYWORDS

Root Cause Analysis, Microservices

ACM Reference Format:
Zhe Xie, Shenglin Zhang, Yitong Geng, Yao Zhang, Minghua Ma, Xiaohui Nie, Zhenhe Yao, Longlong Xu, Yongqian Sun, Wentao Li, and Dan Pei. 2024. Microservice Root Cause Analysis With Limited Observability Through Intervention Recognition in the Latent Space. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3637528.3671530

1 INTRODUCTION

As demands of online applications become more complex and diversified, microservice systems are increasingly being used in online service systems due to their scalability and flexibility [3]. However, system failures are inevitable due to frequent change and scale expansion of microservices [26]. Therefore, over the years, an increasing number of root cause analysis (RCA) algorithms [7, 9, 11, 13, 16, 21, 24] have been proposed to localize failure root cause

[*] Corresponding author.

KDD '24, August 25–29, 2024, Barcelona, Spain
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0490-1/24/08.
https://doi.org/10.1145/3637528.3671530

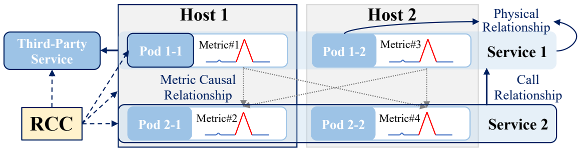


Figure 1: A failure caused by Service 1 in an example microservice system and their corresponding RCCs. RCCs can be heterogeneous and multi-layered.

from *root cause candidates (RCCs)*. In microservice systems, *observability* tools (such as OpenTelemetry[1]) are used to monitor the status of system components by collecting metrics, logs, or traces. RCA algorithms analyze the root causes based on monitoring data and output a ranking for RCCs.

An important challenge in applying RCA algorithms is the limited observability of the RCCs. Observability is crucial for microservices systems [8, 19], and RCA algorithms heavily rely on them to obtain the system status. However, RCCs' observability is often limited in real-world microservice systems for multiple reasons. The first reason comes from the diverse categories of RCCs [5, 11] (see Sec. 2.4). Failures in microservices can be caused by failures in internal components (such as Pods, Services, Hosts in Fig. 1), external components (such as third-party services), external events (such as software changes), etc. RCCs can be heterogeneous, and the components they are associated with can be multi-layered (for example, a Pod is contained within a Host). Different monitoring tools need to be developed separately for heterogeneous RCCs to achieve their observability. This means that fully realizing the observability of RCCs is challenging. Another reason for the limited observability is the absence of direct monitoring metrics for non-observable services (such as third-party services, see 2.3). In such cases, operators attempt to deduce the status of these RCCs through indirectly related metrics (such as Errors related to third-party service calls). Moreover, we find that such limitation may lead to imprecise localizations (Sec. 2.4).

However, existing RCA algorithms usually require RCCs to have good observability. Most RCA algorithms focus only on single-layer RCCs (for example, identifying a specific metric or service), and they are unable to locate heterogeneous RCCs. Some existing algorithms [5, 11] have studied the problem of heterogeneous RCCs and made attempts to tackle it, but they still require good observability for RCCs. Additionally, [5] requires extra expert knowledge to model limited categories of RCCs, making it difficult to extend to other categories of RCCs.

In this paper, for the first time, we propose to **model RCCs as latent variables** and perform RCA for heterogeneous RCCs with the inference of latent variables. The main idea is that latent variables are not required to be observed. Therefore, using latent variables transforms extracting observable features of RCCs into the inference of latent variables. Hence, this modeling approach can be used to model RCCs with limited observability. Since the complete observability of RCCs cannot be guaranteed in practical situations, representing the status of RCCs as latent variables enables accurate modeling of RCCs even under limited observability.

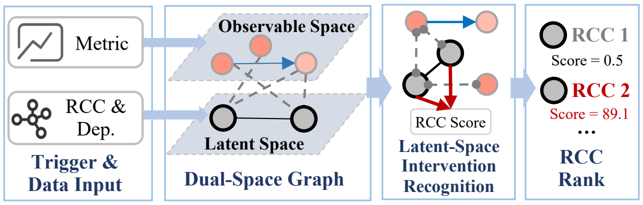


Figure 2: Overall Process of LatentScope

However, there are many challenges in performing RCA with unobservable latent variables: 1) The graph construction for both latent variables (RCCs) and the observable variables (metrics). As shown in Fig. 1, the relationship between RCCs can be heterogeneous (*e.g.*, physical relationship between host and pod, call relationship between services, and causal relationship between metrics). Different relationships imply different causal effects, which the graph should clearly model. 2) Modeling latent RCC variables under limited observability. Latent variables can be inferred through related observable variables. However, the observable data for RCCs can be insufficient under limited observability. 3) Inference the unobservable latent variables with observable data and rank the RCCs. In large-scale microservice systems, there are many RCCs and observable variables. However, the running time of RCA can affect the recovery time. Therefore, efficient algorithms are required to implement the inference.

To address the above challenges, we propose LatentScope, an RCA framework that models RCCs as latent variables with observable monitoring metrics. To address challenge 1, LatentScope builds a dual-space graph for latent and observable variables, respectively. Separating latent and observable variables mitigates the ambiguous causal direction that heterogeneous RCCs and relationships introduce. We introduce indirect metrics in modeling latent RCC variables to address challenge 2, motivated by the RCA process in practice (Sec. 2.4). We also build many-to-many links between latent and observable spaces to achieve this. To address challenge 3, inspired by causal inference with hidden confounders, we infer latent variables in a quantitive approach with causal relationships in a dual-space graph. Motivated by the intervention recognition proposed by CIRCA [9], to enable fast and accurate inference in latent space, we proposed the Regression-based Latent-space Intervention Recognition (RLIR) algorithm to perform quantitative analysis on latent variables of RCCs. Additionally, we introduced LatentRegressor, an enhancement based on the dual-space graph for RLIR, to reduce the computational overhead and mitigate the impact of noises in real-world applications.

We have deployed LatentScope in eBay's production environment and evaluated it with both real-world failures in eBay and a dataset generated using chaos engineering. The main contributions of this work are as follows:

- For the first time, we propose to model root cause candidates (RCCs) as latent variables and perform RCA for heterogeneous RCCs through latent variables.
- We propose LatentScope, an unsupervised RCA framework with RCCs under limited observability. LatentScope applies a novel dual-space graph to model both observable and unobservable variables, using many-to-many relationships between spaces and

[1] https://opentelemetry.io/

solving latent variables in a dual-space graph to realize RCA for heterogeneous RCCs.

- To perform fast inference on latent variables in RCA, we propose **R**egression-based **L**atent-space **I**ntervention **R**ecognition algorithm (RLIR). We also propose the RCC graph-based enhancement, LatentRegressor, to adapt RLIR to real-world systems.
- We deploy LatentScope in eBay's cluster and evaluate it with both eBay's real-world failures and a testbed dataset. The evaluation results indicate that the proposed algorithm achieves superior performance in both micro and macro evaluation metrics for different categories of root causes. We have made the source code and dataset publically available for further research[2].

2 PRELIMINARY AND MOTIVATION

2.1 Intervention Recognition

2.1.1 Background. Intervention recognition for root causal analysis was proposed by CIRCA [9]. The core idea of intervention recognition is to treat faults as interventions. By using causal inference, CIRCA aims to determine the underlying intervention based on observations on metrics. CIRCA constructs a structural causal graph based on meta-variables that contain groups of monitoring metrics. CIRCA analyzed the propagation of anomalies through quantitative modeling of Causal Bayesian Network (CBN), which brings possibilities for fault analysis of RCCs using latent variables in LatentScope.

2.1.2 Regression-Based Hypothesis Testing. CIRCA uses regression-based hypothesis testing (RHT) among the parents and child metrics in the structural causal graph to realize intervention recognition of the metrics. RHT involves training a regression model for each metric and testing whether a metric follows its expected distribution under normal conditions. However, RHT requires variables to be observable, which cannot be used for intervention recognition in latent space.

2.2 Definitions

This section briefly describes the commonly used conceptions in this paper. A *root cause candidate (RCC)* refers to a potential location or event that can be identified as the root cause in microservices systems. The objective of *root cause analysis (RCA)* algorithms is to rank the RCCs to identify the *root cause* of the failure. If two RCCs share the same type of location or event, they belong to the same *category* of RCCs. *Heterogeneous RCCs* means a set of RCCs not all belonging to the same category. Note that the categories of RCCs are different from the "types of failures" that occur on the RCCs, and the classification of failure types is out of the scope of this paper. *Direct metrics* of an RCC refers to the metrics monitoring the location or event of the RCC. If all the abnormal statuses of an RCC can be reflected by at least one of its direct metrics, then we say the RCC has *complete observability*. Otherwise, the RCC has *limited observability*. *Indirect metrics* of an RCC refers to those direct metrics of other RCCs that may be used to infer its status. We use *latent variables* to model RCCs and treat monitoring metrics as *observable variables* in this paper.

2.3 Empirical Study of Heterogeneous RCCs

Table 1: Some Common Categories of Root Causes in eBay

Category	Percentage	Typical Related Metrics
Third-Party Services	63.59%	Third-Party API Error*
Internal Services	8.76%	Runtime Error
Software Change	7.83%	Change Process, Runtime Error*
Database	5.53%	Markdown Error*

To analyze the categories of RCCs in real-world microservice systems, we conducted an empirical study of common categories of root causes in eBay's dataset. Tab. 1 lists four types of root causes and the proportion of faults associated with each over 6 months. It can be found that the RCCs in eBay's microservice system are heterogeneous, and each category of root causes has a different proportion of the total number of faults. For some RCCs (such as Third-Party Services), it is unable to obtain their monitoring metrics. Under such conditions of limited observability, operators usually use metrics in internal services related to Third-Party Services to monitor their status. Thus, these metrics are *indirect metrics*. Through Tab. 1, it can be found that many categories of RCCs are associated with indirect metrics (marked with *). However, we found that these indirect metrics may not accurately reflect the current status of the components (Sec. 2.3).

2.4 Limitation of Indirect Metrics

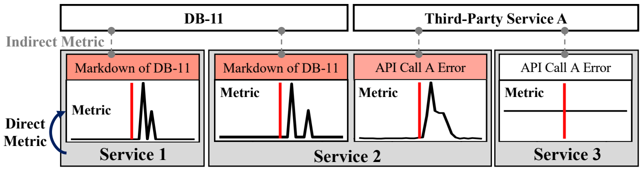


Figure 3: An example showing the inaccuracy of using indirect metrics

To illustrate the inaccuracy with indirect metrics, we analyzed a typical failure case at eBay, as shown in Fig. 3. In this failure, some service metrics (direct metrics of Service 1, 2, 3) exhibited fluctuations, related to Database DB-11 and Third-Party Service A, respectively. However, the root cause of the failure was DB-11. The related metrics in Service 2 are affected by fault propagation, leading to fluctuations in indirect metrics associated with Third-Party Service A. Given the lack of a clear dependency relationship between the DB-11 and Third-Party Service A, it may be difficult for algorithms to identify whether the failure occurs in the database, the third-party service, or Internal Service 2 without additional information. So, the indirect metrics may bring ambiguous information to RCA in real-world applications, resulting in inaccurate localization.

2.5 Motivation of Latent-Space RCC Modeling

To address the challenges brought by the limitation of observability of heterogeneous RCCs, we propose to model heterogeneous RCCs as latent variables. In RCA, metrics are crucial in reflecting the status of RCCs. However, as identified in Sec. 2.3, the observability of RCCs is limited. Furthermore, as observed in Sec. 2.4, the

[2] https://github.com/NetManAIOps/LatentScope

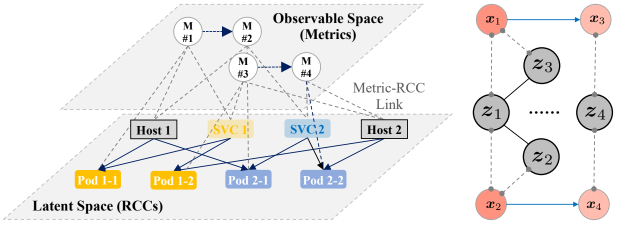
information from indirect metrics is ambiguous and cannot accurately reflect the state of RCCs. Based on this, we propose *modeling RCCs as latent variables*, and the value of the latent variable reflects the state of the RCC. The utilization of latent variables bypasses the need for complete observability of RCCs and transforms the task of extracting observable RCC features into inferring latent variables, enabling effective RCC modeling even when complete observability is not feasible.

However, conducting RCA on these latent variables is challenging. In practice, operators might use additional information to assist in RCA. In Fig. 3, the indirect metrics with DB-11 exhibited significant fluctuations, while only one of the indirect metrics (in Service 2) of Third-Party Service A fluctuated. Thus, the related metrics of different RCCs exhibited different ranges. Motivated by the practice of operators, we can perform RCA by inferring the associated latent RCC variables by analyzing the anomalies in related observable variables (metrics). Through quantitative analysis of latent variables, we can also mitigate the ambiguous information in indirect metrics and achieve more accurate RCA.

3 METHODOLOGY

Based on the idea of "modeling RCCs as latent variables", we propose the LatentScope framework (Fig. 2). Firstly, we propose to use a dual-space graph (latent and observable) to uniformly represent heterogeneous RCCs in the latent-space graph and establish associations between the latent space and observable space. On this basis, we introduce the regression-based latent-space intervention recognition (RLIR) algorithm and its enhancement to achieve fast inference and RCA for latent variables.

3.1 Dual-Space Graph



(a) Dual-Space Graph **(b) Symbols**

Figure 4: Dual-space graph and its symbolic representation

3.1.1 Overview. To accurately model the complex relationship among heterogeneous RCCs, we propose to use a *dual-space graph*, which separates the latent RCC variables from the observable metric variables (Fig. 4a). In the dual-space graph, RCCs are located in the latent layer, and metrics are located in the observable layer. There are different relationships both within the same layer and between different layers. Fig. 4b shows the symbolic representation of the dual-space graph, where $\boldsymbol{x_i}$ represents the nodes in the observable layer, and $\boldsymbol{z_a}$ represents the nodes in the latent layer.

3.1.2 Latent Space and Observable Space. Existing RCA algorithms usually model RCCs within a single-layer graph. However, the relationships among heterogeneous RCCs are distinct (*e.g.*, the relationships between microservices differ from those between containers and hosts), resulting in different fault propagation behaviors. To address this, we model invocation relationships between RCCs with the causal relationships between their related metrics. Firstly, the causal relationships among metrics avoid the ambiguity of causal directions between RCCs [9]. Moreover, modeling heterogeneous RCCs in a unified manner reduces the dependence on expert knowledge [5] about heterogeneous RCCs in algorithm applications.

3.1.3 RCC-Metric Link. The RCC-Metric links are the edges connecting the observable space variables and latent space variables in the dual-space graph. As discussed in Sec.2.3, RCA for RCCs with limited observability relies on indirect metrics. However, indirect metrics are likely to introduce ambiguity in localization, due to each indirect metric being associated with multiple RCCs (Root Cause Candidates). Existing approaches typically model the relationship between RCCs and metrics as *one-to-many* [5, 11], where one metric can be related to only one RCC. Therefore, to achieve accurate modeling of both direct and indirect metrics for RCCs, we propose to use many-to-many relationships to model the link between observable and latent space.

3.1.4 Graph Construction. The construction of the dual-space graph is divided into three steps: establishing causal relationships for metrics in the observable space, establishing many-to-many relationships between spaces, and establishing physical relationships for RCCs in the latent space. To build the causal graph for metrics, we follow the structural graph in CIRCA [9], and build the causal graph for the corresponding metrics based on the call relationship among RCCs. We also apply a Pearson correlation filter to remove redundant edges in the structural graph. Research on the construction of the metric-level graph is outside the scope of this paper. The many-to-many relationships (both direct and indirect) between metrics and RCCs and the physical relationships between RCCs are usually predefined according to the architecture of microservice systems or can be extracted through monitoring tools or microservice traces. At eBay, we retrieve the RCC-metric links and the physical relationship between RCCs from a graph database maintained by operators. One of the main challenges in constructing a dual-space graph for large microservice systems is the need for expert knowledge. Experts need to deeply understand the microservice system to construct the graph accurately. For example, when determining the edges of a database, experts need to identify the potential connections between the database and the internal services. However, since the research on different graph construction methods is out of the scope of this paper, we consider it as future work.

3.2 Regression-based Latent-Space Intervention Recognition

3.2.1 Modeling of Variables in the Dual-Space Graph. We denote each metric in the observable space as $\boldsymbol{x_i}$ and the latent variable for RCC a as $\boldsymbol{z_a}$. If there is an RCC-metric link between $\boldsymbol{x_i}$ and $\boldsymbol{z_a}$, we denote this relationship as $a \in \boldsymbol{rcc}(\boldsymbol{x}_i)$, where $\boldsymbol{rcc}(\boldsymbol{x}_i)$ represents the set of the related RCCs for $\boldsymbol{x}_i$ (both direct and indirect).

Based on the intervention recognition in CIRCA [9], we further propose that the intervention of metric $\boldsymbol{x}_i$ can be decomposed into the intervention of parent metrics in the observable layer $\boldsymbol{pa}(\boldsymbol{x}_i)$ and the related RCCs in the latent layer $\boldsymbol{rcc}(\boldsymbol{x}_i)$. Thus, the value of $\boldsymbol{x}_i$ at timestamp t can be represented by f_i:

$$\boldsymbol{x}_i^{(t)} = f_i(\boldsymbol{pa}(\boldsymbol{x}_i)^{(t)}, \boldsymbol{rcc}(\boldsymbol{x}_i)^{(t)}) \tag{1}$$

which implements unified modeling of both latent and observable variables in the dual-space graph.

3.2.2 Latent-Space Intervention Recognition. Eq. 1 models a causal inference problem with hidden confounders. However, regression-based hypothesis testing (RHT) cannot be applied to causal graphs with hidden confounders and thus cannot be used in our task. In existing methods, although there are approaches like CGNN [4], these methods generally take a long time and require a large amount of training data, making them unsuitable for RCA in large-scale microservice systems.

However, due to the nature of RCA tasks, the modeling of structural graphs can be simplified, allowing for faster solution methods. The choice of f_i significantly determines the complexity of computation. As found in [9], choosing a simple linear function as f_i is effective and shows promising results in RCA tasks. Therefore, following the settings in [9], we apply the same assumption to f_i in this paper. If we divide f_i into the observable layer $f_{i,m}$ and the latent layer $f_{i,r}$, we can derive that:

$$\hat{\boldsymbol{x}_i}^{(t)} = f_{i,r}(\boldsymbol{rcc}(\boldsymbol{x}_i))^{(t)} \tag{2}$$

where $\hat{\boldsymbol{x}_i}^{(t)} = \boldsymbol{x}_i^{(t)} - f_{i,m}(\boldsymbol{pa}(\boldsymbol{x}_i))^{(t)}$ is the residual value for $\boldsymbol{x}_i^{(t)}$, which is equivalent to the error value of metric $\boldsymbol{x}_i$ in regression-based hypothesis test (RHT) in CIRCA. In this way, we construct the relationship between $\hat{\boldsymbol{x}}_i$ and the sum of related latent RCC variables, which enables us to model them with regression-based modeling.

3.2.3 RCC-Solvable Condition. Having assumed the linearity of function f_i, the RCC part of it $f_{i,r}$ can be represented by:

$$f_{i,r}(\boldsymbol{rcc}(\boldsymbol{x}_i))^{(t)} = \sum_{z_a \in \boldsymbol{rcc}(x_i)} \alpha_{a,i} \cdot k_{a,i} \cdot z_a^{(t)} \tag{3}$$

where $k_{a,i}$ is the parameter and $\alpha_{a,i}$ is the root cause metric, where $\alpha_{a,i} = 1$ indicates that RCC $\boldsymbol{z}_a$ is anomalous and affects the monitoring metric $\boldsymbol{x}_i$ in the current failure case. Since the RHT calculation in the $f_{i,m}(\boldsymbol{rcc}(\boldsymbol{x}_i))$ term already includes the constant term, it is no longer included in Eq. 3.

To simplify the solution of Eq. 3, we can introduce some conditions for simplification. As mentioned earlier, the RCA task involves certain constraints compared to general causal inference tasks. Specifically, for $\alpha_{a,i}$ in Eq. 3, it should satisfy the following two conditions:

$$\sum_a \max_i(\alpha_{a,i}) = 1, \alpha_{a,i} \in \{0, 1\} \tag{4}$$

$$\forall a, b\ (b \neq a), \max_i(\alpha_{a,i}) = 1 \implies \exists j\ (\alpha_{a,j} = 1 \land z_b \notin \boldsymbol{rcc}(\boldsymbol{x}_j)) \tag{5}$$

The first condition (Eq. 4) is the root cause uniqueness, which represents that in the current failure, there is only one root cause that causes the fluctuation of metrics. The second condition (Eq. 5) is the common cause uniqueness, meaning that for the metrics affected by the root cause, no other RCCs are linked with these metrics simultaneously. We call this the RCC-Solvable (RCC-S) condition.

3.2.4 RLIR Algorithm. RCC-S provides simplified conditions for solving Eq. 3. Under the RCC-S conditions, we propose the Regression-Based Latent-Space Intervention Recognition (RLIR) algorithm to achieve fast intervention recognition for latent RCC variables. The general process of RLIR contains a two-step calculation:

Step 1: Perform linear regression on each $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ if $\exists z_a \in \boldsymbol{rcc}(\boldsymbol{x}_i) \cap \boldsymbol{rcc}(\boldsymbol{x}_j)$:

$$L_{i,j} = \max_t \frac{\boldsymbol{L}(\hat{\boldsymbol{x}_i}, \hat{\boldsymbol{x}_j})^{(t)}}{\hat{\boldsymbol{x}}_i^{(t)}} \tag{6}$$

where $\boldsymbol{L}(\boldsymbol{x}, \boldsymbol{y})$ denotes performing the prediction of $\boldsymbol{x}$ within the linear regression on $\boldsymbol{x}$ with $\boldsymbol{y}$ and $L_{i,j}$ can be seen as the "proportion" of the regression on $\boldsymbol{x}_i$ with $\boldsymbol{x}_j$.

Step 2: For each $\boldsymbol{x}_i$ and $\boldsymbol{z_a} \in \boldsymbol{rcc}(\boldsymbol{x}_i)$:

$$\boldsymbol{M}_{a,i} = \hat{\boldsymbol{x}}_i \cdot \left(1 - \max_{j, \exists z_b \in \boldsymbol{rcc}_j - \boldsymbol{rcc}_i} L_{i,j}\right) \tag{7}$$

where $\boldsymbol{M}_{a,i}$ will serve as a proxy for calculating the root cause score for $\boldsymbol{z}_a$ and $\boldsymbol{rcc}_i$ is the abbreviation for $\boldsymbol{rcc}(\boldsymbol{x}_i)$. Even though we have not precisely solved the exact values of $k_{a,i}$ and $z_{a,i}$, the root cause score $score_{rcc}(z_a)$ can be calculated with Theorem 3.1:

Theorem 3.1. *If the RCC-S condition holds, the score for RCC $\boldsymbol{z}_a$ can be calculated with Eq. 8:*

$$\begin{aligned} score_{rcc}(z_a) &= \max_{i, z_a \in \boldsymbol{rcc}_i} \max_t \boldsymbol{M}_{a,i}^{(t)} \\ &= \max_{i, z_a \in \boldsymbol{rcc}_i} \left(score_m(\boldsymbol{x}_i) \cdot (1 - \max_{j, \exists z_b \in \boldsymbol{rcc}_j - \boldsymbol{rcc}_i} L_{i,j})\right) \end{aligned} \tag{8}$$

where $rcc_{i,j} = \{z_a | z_a \in \boldsymbol{rcc}(\boldsymbol{x}_i) \cap \boldsymbol{rcc}(\boldsymbol{x}_j)\}$.

Proof. Refer to Appendix A. □

Therefore, we can obtain the results of latent-space intervention recognition with Eq. 8.

3.3 Enhance RLIR with LatentRegressor

3.3.1 Motivation. Although RLIR is already capable of solving latent RCC variables, it still has two problems: 1) The current algorithm's computational overhead is still too high. According to Eq. 6 and Eq. 12, the overall time complexity for RLIR is $O(\#m \times (\#m + \#rcc))$, where $\#m$ denotes the number of metrics and $\#rcc$ denotes the number of RCCs. A large number of calculations is still needed in microservices systems with many metrics. 2) RLIR requires linear regression, which might be sensitive to noise in real-world applications.

Algorithm 1: LatentRegressor

```
Data: RCC Relationship
      $G_{RCC} = (V_r, E_r(a, b, \{\boldsymbol{x}_i | a, b \in \boldsymbol{rcc}(\hat{\boldsymbol{x}}_i)\}))$, Metric
      Relationship $G_{metric} = (V_m, E_m(\boldsymbol{x}_i, \boldsymbol{x}_j))$
Result: Root Cause Score for RCCs
$a\_scores \leftarrow \{\}, b\_scores \leftarrow \{\}, results \leftarrow \{\}$;
foreach $b \in V_r$ do
    $\hat{\boldsymbol{x}}_m \leftarrow \arg\max_{\hat{\boldsymbol{x}}_i, b \in \boldsymbol{rcc}(\boldsymbol{x}_i), \forall c \in next(b) - \boldsymbol{rcc}(\boldsymbol{x}_i)} score_m(\hat{\boldsymbol{x}}_i)$;
    $b\_scores[b][\boldsymbol{x}_{max}] \leftarrow score_m(\boldsymbol{x}_m)$;
end
foreach $a \in V_r$ do
    foreach $b \in next(a)$ do
        //Up Step
        $\hat{\boldsymbol{x}}_a \leftarrow \arg\max_{\hat{\boldsymbol{x}}_i, a \in \boldsymbol{rcc}(\boldsymbol{x}_i), b \notin \boldsymbol{rcc}(\boldsymbol{x}_i)} score_m(\hat{\boldsymbol{x}}_i)$;
        $\hat{\boldsymbol{x}}_b \leftarrow \arg\max_{\hat{\boldsymbol{x}}_i, a \in \boldsymbol{rcc}(\boldsymbol{x}_i), b \in \boldsymbol{rcc}(\boldsymbol{x}_i)} score_m(\hat{\boldsymbol{x}}_i)$;
        $score_a \leftarrow score_m(\boldsymbol{x}_b) \cdot R_{a,b}$;
        $score_b \leftarrow score_m(\boldsymbol{x}_b) \cdot (1 - R_{a,b})$;
        //Down Step
        foreach $c \in prev(b) - \{a\}$ do
            $\hat{\boldsymbol{x}}_c \leftarrow \arg\max_{\hat{\boldsymbol{x}}_i, b \in \boldsymbol{rcc}(\boldsymbol{x}_i), c \in \boldsymbol{rcc}(\boldsymbol{x}_i)} score_m(\hat{\boldsymbol{x}}_i)$;
            $score_a \leftarrow score_a \cdot (1 - R_{b,c})$;
        end
        $a\_scores[a] \leftarrow \max(a\_scores[a], score_a)$;
        if $\boldsymbol{x}_b \in b\_scores[b]$ then
            $b\_scores[b][\boldsymbol{x}_b] \leftarrow$
              $\min(b\_scores[b][\boldsymbol{x}_b], score_b)$;
        end
    end
end
foreach $rcc \in V_r$ do
    $results[rcc] \leftarrow \max(a\_scores[rcc], \max(b\_scores[rcc]))$;
end
return $results$;
```

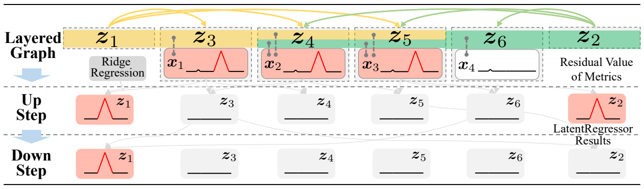


Figure 5: An Example of Process in LatentRegressor

3.3.2 LatentRegressor. To address the challenges, we propose the enhancement algorithm for RLIR, LatentRegressor (Alg. 1). In 1, the calculation of Eq. 12 is simplified. For each edge (E_r) in the RCC layer, we perform a two-step calculation (Up Step and Down Step in 1) between the connected nodes (a and b). After the calculation, we store their scores in a_scores and b_scores respectively and finally take the maximum score for each RCC to obtain the final root cause scores. In this way, LatentRegressor reduces redundant calculations in RLIR (for instance, it’s unnecessary to repeatedly perform regressions with multiple metrics belonging to the same RCC). Additionally, we employ ridge regression ($R_{i,j}$) in LatentRegressor instead of using linear regression for $L_{i,j}$, making it more robust to the noise in Eq. 6. For ease of understanding, each iteration of the LatentRegressor can be viewed as comprising two steps.

The first step is called “Up Step”, which can be viewed as “identifying all the possible parent confounders for the child nodes” in latent space. As shown in Fig. 5, after Step 1, the parent confounders z_1 and z_2 are identified as possible root causes (with fluctuations), while others are eliminated. The second step is the “Down Step”, which “removes the redundant confounders” produced in the Up Step. In Fig. 5, the fluctuation in z_2 is removed. Therefore, only z_1 is identified as the only root cause in the latent space. With LatentRegressor, we only need to perform a regression calculation on the metrics once, significantly reducing computational overhead.

4 EVALUATION

In this section, we focus on the evaluation to answer the following research questions:
RQ1: How does LatentScope perform compared with baselines?
RQ2: Is “modeling RCCs as latent variables” effective in localizing heterogeneous root causes with limited observability?
RQ3: Is the enhancement in LatentRegressor effective in terms of time efficiency and accuracy?

4.1 Evaluation Details

4.1.1 Datasets. We use two datasets for evaluation. Both datasets contain heterogeneous RCCs with limited observability for evaluation.

- **Dataset A.** Collected from eBay’s microservices system containing 66 real-world failure cases over 6 months, labeled by experienced engineers. Dataset A contains over 300 microservices, dozens of databases, hundreds of software changes, more than ten third-party service provider interfaces, and hundreds of thousands of monitoring metrics. In this dataset, most categories of failures exhibit incomplete observability (Sec. 2.3).
- **Dataset B.** Collected from testbed (Online Boutique[3]) with fault injection containing 88 failure cases. Dataset B is comprised of 11 microservices. RCCs can be Pods, Hosts, or Services. We only use metrics data from containers and services and the RCCs have limited observability.

4.1.2 Evaluation Metrics. Following existing RCA research, we use Top-k and MRR for evaluation [13]. We also use macro metrics to balance the weight of different categories of root causes. More details are in Appendix B.

4.1.3 Baselines. We employed the following root cause analysis algorithms as baselines. Most of the baselines cannot perform localization with heterogeneous RCCs. We use metric-level results and take the maximum score of related metrics as the RCC score.
RandomWalk. We use a modified version of random walk [22]. It first performs a random walk on the metric graph, followed by a second random walk in the RCC graph.
MonitorRank. MonitorRank [7] utilizes PageRank for root cause ranking in a causal graph.
MicroScope. MicroScope [12] builds a causal graph with PC algorithm and ranks RCC by similarity after a DFS search.

[3] https://github.com/GoogleCloudPlatform/microservices-demo

Table 2: Comparison with Baselines and Ablation Study

			Micro				Macro				
Dataset	Category	Model	Top@1	Top@5	MRR	MRR@5	Top@1	Top@5	MRR	MRR@5	Avg. Time (s)
A	Baselines	RandomWalk	0.5606	0.5606	0.5683	0.5606	0.1888	0.1888	0.2117	0.1888	7.1
		MonitorRank	0.5000	0.6970	0.5993	0.5745	0.2415	0.3443	0.3228	0.2666	5.9
		MicroScope	0.3030	0.5303	0.4030	0.3851	0.2168	0.2934	0.2759	0.2445	5.4
		CloudRanger	0.2632	0.5802	0.4045	0.3813	0.0939	0.2094	0.1564	0.1358	603.3
		MicroCause	0.2391	0.8478	0.5163	0.5116	0.1433	0.8158	0.3841	0.3707	302.5
		TrinityRCL	0.0303	0.1212	0.0607	0.0494	0.1250	0.3486	0.1858	0.1788	14.6
	Study of Latent Variables	CIRCA-Avg	0.5152	0.8333	0.6512	0.6288	0.2832	0.8335	0.4952	0.4869	11.3
		CIRCA-Max	0.4697	**0.8923**	0.6473	0.6402	0.4243	**0.9118**	0.6228	0.6167	11.3
	Ours	**LatentScope (RLIR Only)**	0.5606	0.8615	0.6828	0.6641	0.4133	0.8699	0.5789	0.5729	136.4
		LatentScope	**0.6154**	**0.8923**	**0.7324**	**0.7205**	**0.6302**	**0.9118**	**0.7430**	**0.7372**	11.6
B	Baselines	RandomWalk	0.1379	0.4912	0.2902	0.2498	0.1273	0.5795	0.3210	0.2841	3.0
		MonitorRank	0.1724	0.5632	0.3146	0.2816	0.1923	0.4313	0.2983	0.2587	5.1
		MicroScope	0.0920	0.5977	0.3123	0.2739	0.1026	0.6404	0.3495	0.3118	0.2
		CloudRanger	0.1707	0.6849	0.3906	0.3703	0.1740	0.7452	0.3936	0.3786	3.2
		MicroCause	0.1724	0.7241	0.4101	0.3771	0.1862	0.7908	0.4151	0.3851	71.9
		TrinityRCL	0.0517	0.1264	0.1169	0.0795	0.1189	0.2809	0.2185	0.1775	6.1
	Study of Latent Variables	CIRCA-Avg	0.2045	0.8295	0.4589	0.4186	0.2246	0.7698	0.4332	0.3841	10.7
		CIRCA-Max	0.2159	0.8636	0.4633	0.4393	0.2804	0.8847	0.4537	0.4344	10.7
	Ours	**LatentScope (RLIR Only)**	0.3258	**0.9299**	0.4882	0.4706	0.3063	**0.9353**	0.4745	0.4586	586.3
		LatentScope	**0.3750**	0.9205	**0.6064**	**0.5953**	**0.4337**	0.9287	**0.6491**	**0.6394**	10.9

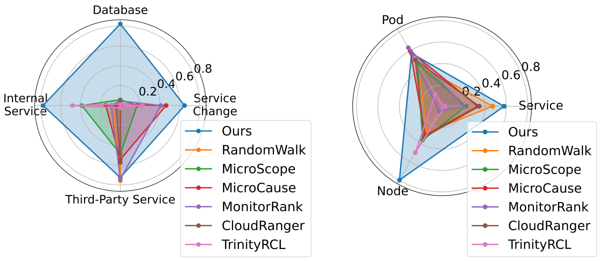
MicroCause. MicroCause [16] proposes Path Condition Time Series (PCTS) algorithm for causal discovery and utilizes second-order random walk to address delay in failure propagation.

CloudRanger. CloudRanger [21] constructs a causal graph with PC and employs a second-order random walk for root cause ranking.

TrinityRCL. TrinityRCL [5] proposes a multi-granular RCA algorithm based on a random walk, but limited in services, hosts, metrics, and codes levels. We modified TrinityRCL by creating virtual hosts and services to adapt it to more RCCs in our datasets.

CIRCA. CIRCA [9] proposes intervention-recognition RCA at metric level. It is equivalent to a variation of LatentScope without a latent layer. Thus, we detail the comparison of CIRCA in RQ2.

4.2 RQ1: Comparison with Baselines



(a) Dataset A **(b) Dataset B**

Figure 6: Micro MRR for Different Categories of Root Cause

The comparison with baselines is shown in Tab. 2. The experimental results demonstrate that our LatentScope outperforms the baseline algorithm in both Micro and Macro metrics. This indicates that our algorithm, compared to existing ones, can effectively improve performance in heterogeneous root cause localization tasks. It is noteworthy that our algorithm has achieved a significant improvement in the Macro metrics. In algorithms like MonitorRank, MicroCause, and MicroScope, root cause localization is conducted solely at the metric-level causal graphs. Due to the complexity brought by heterogeneous RCCs and the limitation of observability, even though they might accurately locate causes at the metric level, they may not perform well in heterogeneous root cause localization tasks. As for Random Walk, despite employing a dual-space graph, the complex relationships between RCCs make it challenging to accurately quantify the transition matrix. LatentScope uses latent-space intervention recognition to perform precise modeling of latent RCC variables and achieves accurate RCA in most cases.

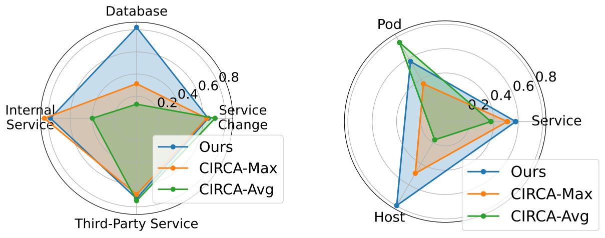
Additionally, Tab. 2 shows the runtime of LatentScope. It can be observed that the average running time of LatentScope is acceptable compared to the baseline algorithms. Due to the significant reduction in computational complexity by LatentRegressor (which will be further discussed in Sec. 4.4), LatentScope is more suitable for large-scale microservice systems compared to algorithms like CloudRange and MicroCause which require causal discovery.

To further explore why our algorithm outperforms the baseline algorithm, Fig. 6 demonstrates the micro MRR in different categories of root causes. In Dataset A, most algorithms can accurately locate faults in third-party services but perform poorly in other cases. In Dataset B, LatentScope also shows an advantage in locating root causes in node and service cases. This indicates that the design of LatentScope framework effectively models heterogeneous RCCs in different datasets.

We can also find that there is a significant difference in Micro Top@1 between Dataset A and B. The main reason for this is that Dataset B has more limitations in observability compared to Dataset A. Dataset A is collected from a real microservices system and includes more comprehensive observability metrics. Despite the limited observability in Dataset B, LatentScope demonstrated robust performance and significantly outperformed the baselines.

This indicates that our algorithm is well-suited to work with limited observability in the microservices systems.

4.3 RQ2: Ablation Study of Latent Layer



(a) Dataset A

(b) Dataset B

Figure 7: Ablation Study of Latent Layer

We conduct an ablation study to investigate the effectiveness of "modeling RCCs as latent variables". To achieve this, we replaced the latent layer and LatentRegressor by applying two types of aggregation directly on the metric layer: the maximum value of related metrics (CIRCA-Max) and the average value of related metrics (CIRCA-Avg). As mentioned earlier, CIRCA is equivalent to the variation of LatentScope without the latent layer. The removal of latent variables leaves only the metric layer, so it is equivalent to CIRCA with different aggregation methods. The results of the experiment are shown in Tab. 2 and Fig. 7. We can find that our algorithm significantly outperforms the compared algorithms in evaluation metrics. Moreover, Fig. 7 indicates that LatentScope achieves good results in almost all categories of cases, while the compared methods typically achieve good results in cases with only certain categories of root causes. This might be due to their difficulty distinguishing the specific level of the heterogeneous RCCs (*e.g.*, host and pod) under limited observability. Refer to the case studies in Sec. 5.2 for details to further understand the process and effectiveness of the latent variables under limited observability.

4.4 RQ3: Ablation Study of LatentRegressor

Table 3: Avg. Running Time (s) of RLIR and LatentRegressor

Dataset	LatentRegressor	RLIR	Dataset	LatentRegressor	RLIR
A	0.7	124.7	B	0.2	579.1

In order to investigate whether LatentRegressor can effectively improve accuracy and reduce running time, we conducted an ablation study. We compare the running time of RLIR (LatentRegressor without enhancement) and the LatentRegressor in Tab. 3 and their accuracy in Tab. 2. The comparison results indicate that the proposed enhancement in LatentRegressor not only achieves much faster speed but also achieves higher accuracy in most evaluation metrics, demonstrating the effectiveness of LatentRegressor in terms of both accuracy and time efficiency with the reduced regression calculations and the use of Ridge regression.

5 DEPLOYMENT AND CASE STUDY

5.1 Deployment Details

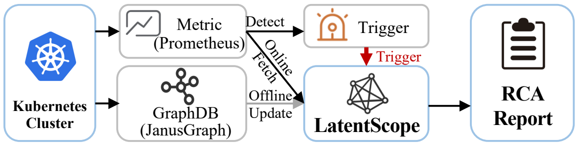


Figure 8: Study of Deployment of LatentScope

To investigate the effectiveness of LatentScope in real-world applications, we deploy LatentScope in eBay's production cluster. The overall process of the deployed algorithm is illustrated in Fig. 8. Refer to Appendix D for detailed system information. LatentScope is deployed as a microservice application in eBay's Kubernetes cluster with the limitation of 2 CPU cores. Upon the occurrence of a failure, LatentScope is triggered and fetches the metrics. After performing LatentRegressor, the ranking is output to the operators. During the deployment on eBay's cluster, due to the slow speed of metric queries and retrieval, we made some trade-offs in the implementation. When an anomalous SLI (service level indicator) is detected, we determine the failing domain based on the triggered SLI (such as the user experience of buying/selling in eBay) and only extract metrics related to services in that domain to reduce data retrieval time.

5.2 Case Studies

The quantitative evaluation of the deployed algorithm has been discussed in Sec. 4. In this section, we perform 2 case studies to further demonstrate the working process of LatentScope.

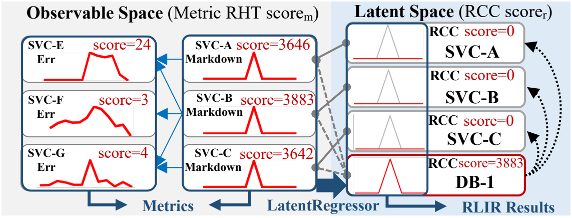


Figure 9: Case study of a database failure case

5.2.1 Case Study 1: Online Database Failure in eBay. In this case, we perform a case study (Fig. 9) on a real-world database failure in eBay with the result of the deployed algorithm. As shown in Fig. 9, the database failure in DB-1 led to anomalies in the metrics of several services through metric-RCC links (grey lines) and failure propagation in the metrics (blue lines). However, there are direct metrics for internal services, but only indirect metrics for DB-1, which has limited observability. In most existing RCA algorithms, the relationship between RCCs and metrics is usually one-to-many. In such scenarios, the fault could be incorrectly located in one of the internal services based on the direct metrics, which are anomalous, rather than in the actual root cause DB-1. With LatentScope, both the failure propagation in observable metrics and latent RCCs can be modeled through the dual-space graph and latent-space intervention recognition. The latent variables for RCCs A, B, and C are considered to be normal (red lines), while the root cause DB-1 shown in a spike, is more likely to be considered as the root cause. This case demonstrates that LatentScope can be effectively applied to accurately infer fault root causes under limited observability.

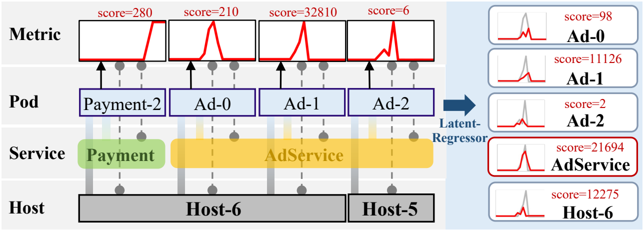


Figure 10: Case study of a service failure case

5.2.2 Case Study 2: Service Failure. To understand the working process of LatentScope in a complex case with multi-layered RCCs, we perform another case study (Fig. 10) with a service failure case in Dataset B. This case demonstrates a complex many-to-many relationship between metrics and RCCs, where each metric is related to RCCs of three types: Pod, Service, and Host. RCA in such a scenario is typically complex. In existing metric-level RCA algorithms, even if we can accurately locate the root cause metric, it is difficult to directly determine which corresponding RCC is the root cause due to the multi-granularity of RCCs. Service-level root cause localization (such as trace-based RCA) may also fail to judge faults occurring in a specific Pod and may incorrectly locate a fault affecting an individual Service when a Host is entirely failing. The red lines in the figure illustrate the computational results of the LatentRegressor algorithm. It can be observed that the RLIR process in LatentRegressor "corrects" the original metrics (grey) through the many-to-many relationship between RCCs and metrics, eliminating the influence in non-root cause RCCs. For the root cause (AdService), the "corrected" results show a higher root cause score, making it the most likely fault root cause. This demonstrates that the quantitative solution of latent variables for the root cause by RLIR is feasible and effective. It also demonstrates the importance of using quantitative modeling in RCA.

6 THREATS TO VALIDITY

The threat to the internal validity of this paper is the implementation of the baselines. We implemented RandomWalk, MonitorRank, MicroScope, CloudRanger, and MicroCause by ourselves, as there are no publicly available implementations.

The threat to external validity mainly lies in this paper in the evaluation datasets. The failures in eBay may not represent the performance in other systems. To reduce this threat, we also collect Dataset B from the testbed and use it for evaluation.

7 RELATED WORKS

Unsupervised Root Cause Analysis. Current unsupervised root cause analysis approaches can be divided into metric-based, event-based, log-based, and trace-based [17]. The widely adopted workflow in metric-based approaches [2, 9, 12, 15, 16, 18, 21–23] is building a causal graph on the metrics and ranking the metrics to localize the failure root cause. Event-based approaches like Groot [20] and Nezha [25] construct event graphs according to rules or features extracted from events and rank the events with the event graph. Compared with metric-based and event-based approaches, log-based [1] and trace-based [6, 10, 13, 14] use additional features in logs and traces to achieve more accurate ranking. However, these methods typically focus only on root cause identification at a single level with a high requirement for observability, making it difficult to address the RCA of heterogeneous RCCs with limited observability.

Heterogeneos Root Cause Localization. Recently, some RCA algorithms have focused on addressing RCA with heterogeneous RCCs. Dejavu [11] proposed the concept of actionable root cause analysis, which defines RCC with the failure type as a "failure unit" and localizes heterogeneous RCCs with graph neural networks. However, Dejavu is a supervised algorithm that requires manual labeling and end-to-end training, making it difficult to adapt to the frequent dynamic changes of the microservices system. TrinityRCL [5] introduces a ”multi-granularity” RCA algorithm. TrinityRCL uses random walks to locate RCCs at four levels: service, host, metric, and code. However, TrinityRCL relies on strong expert knowledge to set transition probabilities between different types of RCCs, limiting its scalability. Additionally, these methods require RCCs to have complete observability.

8 CONCLUSION

In this paper, we focus on the challenges of RCA with heterogeneous RCCs with limited observability in real-world microservice systems. To address the challenges, we conduct an empirical study of eBay's failure and propose the idea of "modeling RCCs with latent variables". Based on the idea, we propose LatentScope, an unsupervised RCA framework for heterogeneous RCCs. LatentScope introduce a dual-space graph to model both observable metrics and latent RCC variables, with a many-to-many relationship between the spaces. To infer the latent variables and achieve RCA in the dual-space graph, we propose latent-space intervention recognition (RLIR) and its enhancement, LatentRegressor. We deployed LatentScope in eBay's production cluster and evaluated it with both eBay's real-world failure cases and testbed failure cases. Experimental results show that our algorithm achieves effective improvement in unsupervised root cause localization tasks. In particular, compared with baseline algorithms, LatentScope significantly improves the accuracy in different categories of root causes, indicating the effectiveness of our algorithm in solving RCA tasks with heterogeneous RCCs.

ACKNOWLEDGMENTS

We thank Huai Jiang and Mingjie Li for their helpful suggestions. This work is supported by the National Natural Science Foundation of China under Grant 62272249, 62072264, and 62302244, and the Beijing National Research Center for Information Science and Technology (BNRist) key projects.

REFERENCES

[1] Pooja Aggarwal, Ajay Gupta, Prateeti Mohapatra, Seema Nagar, Atri Mandal, Qing Wang, and Amit Paradkar. 2021. Localization of Operational Faults in Cloud Applications by Mining Causal Dependencies in Logs Using Golden Signals. In *Service-Oriented Computing – ICSOC 2020 Workshops*, Hakim

Hacid, Fatma Outay, Hye-young Paik, Amira Alloum, Marinella Petrocchi, Mohamed Reda Bouadjenek, Amin Beheshti, Xumin Liu, and Abderrahmane Maaradji (Eds.). Vol. 12632. Springer International Publishing, Cham, 137–149. https://doi.org/10.1007/978-3-030-76352-7_17

[2] Pengfei Chen, Yong Qi, Pengfei Zheng, and Di Hou. 2014. CauseInfer: Automatic and Distributed Performance Diagnosis with Hierarchical Causality Graph in Large Distributed Systems. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. IEEE, Toronto, ON, Canada, 1887–1895. https://doi.org/10.1109/INFOCOM.2014.6848128

[3] Paolo Di Francesco, Ivano Malavolta, and Patricia Lago. 2017. Research on architecting microservices: Trends, focus, and potential for industrial adoption. In *2017 IEEE International conference on software architecture (ICSA)*. IEEE, 21–30.

[4] Olivier Goudet, Diviyan Kalainathan, Philippe Caillou, Isabelle Guyon, David Lopez-Paz, and Michele Sebag. 2018. Learning functional causal models with generative neural networks. *Explainable and interpretable models in computer vision and machine learning* (2018), 39–80.

[5] Shenghui Gu, Guoping Rong, Tian Ren, He Zhang, Yongda Yu, Xian Li, Jian Ouyang, and Chunan Chen. 2023. TrinityRCL: Multi-Granular and Code-Level Root Cause Localization Using Multiple Types of Telemetry Data in Microservice Systems. *IEEE Transactions on Software Engineering* 49, 5 (May 2023), 3071–3088. https://doi.org/10.1109/TSE.2023.3241299

[6] Xiaofeng Guo, Xin Peng, Hanzhang Wang, Wanxue Li, Huai Jiang, Dan Ding, Tao Xie, and Liangfei Su. 2020. Graph-Based Trace Analysis for Microservice Architecture Understanding and Problem Diagnosis. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, Virtual Event USA, 1387–1397. https://doi.org/10.1145/3368089.3417066

[7] Myunghwan Kim, Roshan Sumbaly, and Sam Shah. 2013. Root Cause Detection in a Service-Oriented Architecture. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*. ACM, Pittsburgh PA USA, 93–104. https://doi.org/10.1145/2465529.2465753

[8] Bowen Li, Xin Peng, Qilin Xiang, Hanzhang Wang, Tao Xie, Jun Sun, and Xuanzhe Liu. 2022. Enjoy your observability: an industrial survey of microservice tracing and analysis. *Empirical Software Engineering* 27 (2022), 1–28.

[9] Mingjie Li, Zeyan Li, Kanglin Yin, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2022. Causal Inference-Based Root Cause Analysis for Online Service Systems with Intervention Recognition. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, Washington DC USA, 3230–3240. https://doi.org/10.1145/3534678.3539041

[10] Zeyan Li, Junjie Chen, Rui Jiao, Nengwen Zhao, Zhijun Wang, Shuwei Zhang, Yanjun Wu, Long Jiang, Leiqin Yan, Zikai Wang, Zhekang Chen, Wenchi Zhang, Xiaohui Nie, Kaixin Sui, and Dan Pei. 2021. Practical Root Cause Localization for Microservice Systems via Trace Analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, Tokyo, Japan, 1–10. https://doi.org/10.1109/IWQOS52092.2021.9521340

[11] Zeyan Li, Nengwen Zhao, Mingjie Li, Xianglin Lu, Lixin Wang, Dongdong Chang, Xiaohui Nie, Li Cao, Wenchi Zhang, Kaixin Sui, Yanhua Wang, Xu Du, Guoqiang Duan, and Dan Pei. 2022. Actionable and Interpretable Fault Localization for Recurring Failures in Online Service Systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, Singapore Singapore, 996–1008. https://doi.org/10.1145/3540250.3549092

[12] Jinjin Lin, Pengfei Chen, and Zibin Zheng. 2018. Microscope: Pinpoint Performance Issues with Causal Graphs in Micro-service Environments. In *Service-Oriented Computing*, Claus Pahl, Maja Vukovic, Jianwei Yin, and Qi Yu (Eds.). Vol. 11236. Springer International Publishing, Cham, 3–20.

[13] Dewei Liu, Chuan He, Xin Peng, Fan Lin, Chenxi Zhang, Shengfang Gong, Ziang Li, Jiayu Ou, and Zheshun Wu. 2021. MicroHECL: High-Efficient Root Cause Localization in Large-Scale Microservice Systems. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, Madrid, ES, 338–347. https://doi.org/10.1109/ICSE-SEIP52600.2021.00043

[14] Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, and Dan Pei. 2020. Unsupervised Detection of Microservice Trace Anomalies through Service-Level Deep Bayesian Networks. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, Coimbra, Portugal, 48–58. https://doi.org/10.1109/ISSRE5003.2020.00014

[15] Xianglin Lu, Zhe Xie, Zeyan Li, Mingjie Li, Xiaohui Nie, Nengwen Zhao, Qingyang Yu, Shenglin Zhang, Kaixin Sui, Lin Zhu, and Dan Pei. 2022. Generic and Robust Performance Diagnosis via Causal Inference for OLTP Database Systems. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, Taormina, Italy, 655–664. https://doi.org/10.1109/CCGrid54584.2022.00075

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

[17] Jacopo Soldani and Antonio Brogi. 2023. Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey. *Comput. Surveys* 55, 3 (April 2023), 1–39. https://doi.org/10.1145/3501297

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

[19] Muhammad Usman, Simone Ferlin, Anna Brunstrom, and Javid Taheri. 2022. A survey on observability of distributed edge & container-based microservices. *IEEE Access* (2022).

[20] Hanzhang Wang, Zhengkai Wu, Huai Jiang, Yichao Huang, Jiamu Wang, Selcuk Kopru, and Tao Xie. 2021. Groot: An event-graph-based approach for root cause analysis in industrial settings. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 419–429.

[21] Ping Wang, Jingmin Xu, Meng Ma, Weilan Lin, Disheng Pan, Yuan Wang, and Pengfei Chen. 2018. CloudRanger: Root Cause Identification for Cloud Native Systems. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, Washington, DC, USA, 492–502. https://doi.org/10.1109/CCGRID.2018.00076

[22] Jianping Weng, Jessie Hui Wang, Jiahai Yang, and Yang Yang. 2018. Root cause analysis of anomalies of multitier services in public clouds. *IEEE/ACM Transactions on Networking* 26, 4 (2018), 1646–1659.

[23] Li Wu, Johan Tordsson, Jasmin Bogatinovski, Erik Elmroth, and Odej Kao. 2021. MicroDiag: Fine-grained Performance Diagnosis for Microservice Systems. In *2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*. IEEE, Madrid, Spain, 31–36. https://doi.org/10.1109/CloudIntelligence52565.2021.00015

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

[25] Guangba Yu, Pengfei Chen, Yufeng Li, Hongyang Chen, Xiaoyun Li, and Zibin Zheng. 2023. Nezha: Interpretable Fine-Grained Root Causes Analysis for Microservices on Multi-modal Observability Data. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 553–565.

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

A PROOF OF THEOREM 3.1

LEMMA A.1. *If the RCC-S condition holds,* $\max\limits_{j,b\in rcc_i - rcc_{i,j}} L_{i,j} \cdot \hat{\boldsymbol{x}}_i = \alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a$ *if* $\alpha_{a,i} = 1$ *and* $a \neq b$.

PROOF. According to Eq. (5) in the RCC-S condition, given $a \neq b$ and $\alpha_{a,i} = 1$, we can find an $\boldsymbol{x}_j$ satisfying $\alpha_{a,j} = 1 \wedge \boldsymbol{z}_b \notin \boldsymbol{rcc}(\boldsymbol{x}_j)$. And according to Eq. (4) in the RCC-S condition, $\forall k, b \neq a, \alpha_{b,k} = 0$. We have $\hat{\boldsymbol{x}}_i = \alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a$ and $\hat{\boldsymbol{x}}_j = \alpha_{a,j} \cdot k_{a,j} \cdot \boldsymbol{z}_a$, which gives that:

$$\frac{\hat{\boldsymbol{x}}_i}{\alpha_{a,i} \cdot k_{a,i}} = \boldsymbol{z}_a = \frac{\hat{\boldsymbol{x}}_j}{\alpha_{a,j} \cdot k_{a,j}}, \tag{9}$$

indicating that $\hat{\boldsymbol{x}}_i$ and $\hat{\boldsymbol{x}}_j$ are linear correlated. Thus, we can derive that $L_{i,j} \cdot \hat{\boldsymbol{x}}_i = \hat{\boldsymbol{x}}_i = \alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a$. And since the maximum value of $L_{i,j} \cdot \hat{\boldsymbol{x}}_i$ is no larger than $\alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a$, the lemma can be proved. □

LEMMA A.2. *If the RCC-S condition holds,* $\boldsymbol{M}_{a,i} = \alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a$, *where* $\boldsymbol{M}_{a,i}$ *is defined in Eq. 7.*

PROOF. Denote $L_i = \max\limits_j L_{i,j}$ and $L_{i,a} = \max\limits_{j,z_a \in rcc_i - rcc_{i,j}} L_{i,j}$. When $\forall z_a \in rcc_i, \alpha_{a,i} = 0$, all $L_i, L_{i,j,a}$ and $\alpha_{a,i}$ are zero. In this case, Eq. (7) clearly holds. If $\alpha_{a,i} = 1$, according to Eq. (4), $\forall b, i, \alpha_{b,i} = 0$, thus $L_{i,j,a} = 0$. We can derive that:

$$\begin{aligned}(L_i - L_{i,j,a}) \cdot \hat{\boldsymbol{x}}_i &= \alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a + \sum_{b \neq a} \alpha_{b,i} \cdot k_{b,i} \cdot \boldsymbol{z}_b - 0 \\ &= \alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a\end{aligned} \tag{10}$$

If $\alpha_{a,i} = 0$ but $\exists b \neq a, \alpha_{b,i} = 1$, according to Lemma A.1, it can be derived that:

$$\begin{aligned}L_i - L_{i,j,a} &= \sum_{z_c \in rcc_i} \alpha_{c,i} \cdot k_{c,i} \cdot \boldsymbol{z}_c - \alpha_{b,i} \cdot k_{b,i} \cdot \boldsymbol{z}_b \\ &= \alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a + \sum_{c \notin \{a,b\}} \alpha_{c,i} \cdot k_{c,i} \cdot \boldsymbol{z}_c + 0 \\ &= \alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a + 0 = \alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a\end{aligned} \tag{11}$$

Therefore, Eq. 7 holds under the RCC-S condition, thus the RLIR algorithm solves the equation. □

THEOREM 3.1. *If the RCC-S condition holds, the score for RCC* $\boldsymbol{z}_a$ *can be calculated with:*

$$\begin{aligned}score_{rcc}(\boldsymbol{z}_a) &= \max_{i,z_a \in \boldsymbol{rcc}_i} \max_t \boldsymbol{M}_{a,i}^{(t)} \\ &= \max_{i,z_a \in \boldsymbol{rcc}_i} \left(score_m(\boldsymbol{x}_i) \cdot (1 - \max_{j,\exists z_b \in rcc_j - rcc_i} L_{i,j}) \right)\end{aligned} \tag{12}$$

where $rcc_{i,j} = \{\boldsymbol{z}_a | \boldsymbol{z}_a \in \boldsymbol{rcc}(\boldsymbol{x}_i) \cap \boldsymbol{rcc}(x_j)\}$.

PROOF. According to Lemma A.2, the value of $\alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a^{(t)}$ can be linear represented by $\boldsymbol{x}_i^{(t)}$. Therefore, the score of $\alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a$ can be calculated with:

$$\begin{aligned}score_{rcc}(\boldsymbol{z}_{a,i}) &= \max_{t=t_1}^{t_2} \frac{\alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a^{(t)} - \boldsymbol{\mu}(\alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a)_{[t_0:t_1]}}{\sigma(\alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a)_{[t_0:t_1]}} \\ &= \max_{t=t_1}^{t_2} \left(\frac{\lambda_i \cdot \hat{\boldsymbol{x}}_i^{(t)} - \boldsymbol{\mu}(\lambda_i \cdot \hat{\boldsymbol{x}}_i)_{[t_0:t_1]}}{\sigma(\lambda_i \cdot \hat{\boldsymbol{x}}_i)_{[t_0:t_1]}} \right)\end{aligned} \tag{13}$$

where $score_{rcc}(\boldsymbol{z}_{a,i})$ denotes the score of $\boldsymbol{z}_a$ on the monitoring metric $\boldsymbol{x}_i$ and $\lambda_i = 1 - 1 - \max_{j,a \in rcc_j - rcc_{i,j}}$.

According to the proof in Lemma A.2, when $\alpha_{a,i} = 0$, we have $1 - \max_{j,a \in rcc_j - rcc_{i,j}} = 0$, thus $\lambda_i = 0$. And when $\alpha_{a,i} = 1$, we have $\lambda_i = 1$. Thus we have:

$$\begin{aligned}score_{rcc}(\boldsymbol{z}_{a,i}) &= \max_{t=t_1}^{t_2} \left(\frac{\lambda_i \cdot \boldsymbol{x}_i^{(t)} - \boldsymbol{\mu}(\lambda_i \cdot \boldsymbol{x}_i)_{[t_0:t_1]}}{\sigma(\lambda_i \cdot \boldsymbol{x}_i)_{[t_0:t_1]}} \right) \\ &= \max_{t=t_1}^{t_2} \left(\frac{\hat{\boldsymbol{x}}_i^{(t)} - \boldsymbol{\mu}(\hat{\boldsymbol{x}}_i)_{[t_0:t_1]}}{\sigma(\hat{\boldsymbol{x}}_i)_{[t_0:t_1]}} \right) \cdot \lambda_i \\ &= score_m(\boldsymbol{x}_i) \cdot \left(1 - \max_{j,a \in rcc_j - rcc_{i,j}} \right)\end{aligned} \tag{14}$$

And since we have:

$$\begin{aligned}score_{rcc}(\boldsymbol{z}_a) &= \max_{t=t_1}^{t_2} \left(\frac{\boldsymbol{z}_a^{(t)} - \boldsymbol{\mu}(\boldsymbol{z}_a)_{[t_0:t_1]}}{\sigma(\boldsymbol{z}_a)_{[t_0:t_1]}} \right) \\ &= \max_{t=t_1}^{t_2} \left(\frac{\alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a^{(t)} - \boldsymbol{\mu}(\alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a)_{[t_0:t_1]}}{\sigma(\alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a)_{[t_0:t_1]}} \right) \\ &\forall \alpha_{a,i} = 1\end{aligned} \tag{15}$$

According to Eq. 4, if $\boldsymbol{z}_a$ is the root cause of the failure, then we have $\max(\alpha_{a,i}) = 1$, which indicates that at least one $\alpha_{a,i}$ is non-zero. Thus we have:

$$\begin{aligned}score_{rcc}(\boldsymbol{z}_a) &= \max_i \max_{t=t_1}^{t_2} \left(\frac{\alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a^{(t)} - \boldsymbol{\mu}(\alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a)_{[t_0:t_1]}}{\sigma(\alpha_{a,i} \cdot k_{a,i} \cdot \boldsymbol{z}_a)_{[t_0:t_1]}} \right) \\ &= \max_{i,z_a \in \boldsymbol{rcc}_i} \left(score_m(\boldsymbol{x}_i) \cdot (1 - \max_{j,a \in rcc_j - rcc_{i,j}} L_{i,j}) \right)\end{aligned} \tag{16}$$

When $\max(\alpha_{a,i}) = 0$, then $\forall i, score_{rcc}(\boldsymbol{z}_a) = score_{rcc}(\boldsymbol{z}_{a,i}) = 0$. Therefore, the original equation holds. □

B EVALUATION METRICS

To comprehensively and accurately evaluate the root cause localization performance of our algorithm and baseline algorithms across multi-level RCCs, following existing similar work [13], we choose two types of metrics: Top-k and MRR. Top-k refers to the ranking of the root cause in the output of the localization algorithm. MRR is the mean of the reciprocals of all the rankings. Furthermore, since RCCs ranked too low are often overlooked in practical applications, we incorporated MRR@k, which means considering only the top k results and calculating their MRR.

Additionally, in our root cause localization task, different algorithms may exhibit significant biases in localizing various types of root causes (for instance, some algorithms might be more inclined to localize fine-grained RCCs). However, in our datasets, due to the limitations in the number of failures and data collection, the quantity of failure cases for different types of root causes is imbalanced. Therefore, to better evaluate the localization performance of algorithms across all types of root causes, we calculate the unweighted performance (micro) of the algorithm for all failure cases, and the weighted performance (macro) that balances the algorithm's performance across different categories.

C IMPLEMENTATION DETAILS

We use Python to implement the LatentScope model. In Dataset A, we preprocessed the data by filtering out monitoring metrics irrelevant by their business domain (*e.g.* shipping domain and checkout domain) according to the triggered metrics. Additionally, we apply a Pearson correlation filter to remove metrics unrelated to the specific fault occurrence. The preprocess method and the filter are applied to all the baselines for fair comparison.

D DEPLOYMENT DETAILS

In eBay, the microservices system is orchestrated with Kubernetes[4] and the monitoring metrics are collected with MetricBeat[5] and stored with Prometheus[6]. A GraphDB based on JanusGraph[7] is utilized for storing the relationships between RCCs and metrics, from where LatentScope retrieves data offline for constructing the dual-space graphs. LatentScope is deployed on a host with Intel(R) Xeon(R) Gold 6138 CPU with the limitation of 2 CPU cores and 4 GB memories.

[4] https://kubernetes.io/

[5] https://www.elastic.co/beats/metricbeat

[6] https://prometheus.io/

[7] https://janusgraph.org/