



UvA-DARE (Digital Academic Repository)

A fine-grained robust performance diagnosis framework for run-time cloud applications

Xin, R.; Chen, Peng; Grosso, P.; Zhao, Z.

DOI

[10.1016/j.future.2024.02.014](https://doi.org/10.1016/j.future.2024.02.014)

Publication date

2024

Document Version

Final published version

Published in

Future Generation Computer Systems

License

CC BY

[Link to publication](#)

Citation for published version (APA):

Xin, R., Chen, P., Grosso, P., & Zhao, Z. (2024). A fine-grained robust performance diagnosis framework for run-time cloud applications. *Future Generation Computer Systems*, 155, 300-311. <https://doi.org/10.1016/j.future.2024.02.014>

General rights

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (<https://dare.uva.nl>)



A fine-grained robust performance diagnosis framework for run-time cloud applications

Ruyue Xin^a, Peng Chen^{b,*}, Paola Grosso^a, Zhiming Zhao^{a,*}

^a Multiscale Networked Systems (MNS), University of Amsterdam, Amsterdam, Netherlands

^b School of Computer and Software Engineering, Xihua University, Chengdu, China

ARTICLE INFO

Keywords:

Performance diagnosis
Metrics selection
Deep ensemble learning
Causal graph
Fine-grained

ABSTRACT

To maintain the required service quality of time-critical cloud applications, operators must continuously monitor their runtime status, detect potential performance anomalies, and diagnose the root causes of these anomalies effectively. However, existing performance diagnosis methods face challenges such as the need for high-quality labeled data, the low reusability and robustness of performance anomaly detection models, and the absence of real-time fine-grained root cause localization. These challenges make fixing performance issues quickly and developing effective adaptation decisions difficult. We provide a Fine-grained Robust Performance Diagnosis (FIRE) framework to tackle those challenges. The framework offers a metrics selection component to filter noise and improve detection efficiency, an anomaly detection component that assembles several well-selected base models with a deep neural network, and adopts weakly supervised learning considering fewer labels exist in reality. The framework also employs a real-time, fine-grained root cause localization component to locate dependent resource metrics of performance anomalies. Our experiments show that the framework can effectively reduce data noise and achieve the best accuracy and algorithm robustness for performance anomaly detection. In addition, the framework can accurately localize the first root causes, with an average accuracy higher than 0.7 for locating the first four root cause metrics.

1. Introduction

Cloud environments provide elastic and on-demand resources for developing applications [1]. However, because of the inherent dynamism of clouds, performance anomalies, such as degraded response time, can significantly impact the overall user experience quality. In addition, considering complex dependencies and multiple components in cloud applications [2], it is difficult for operators to detect performance anomalies and identify root causes accurately and timely. Traditionally, operators manually diagnose cloud applications, which is complicated and time-consuming. Continuous data collection enables the capture of multiple monitoring metrics, including CPU usage, which offers valuable insights into the real-time state of cloud applications [3]. Therefore, we can consider a performance diagnosis solution for cloud applications that leverage monitoring data to support effective performance anomaly detection and rapid determination of root causes, enabling prompt recovery and loss mitigation.

For an actual cloud application, monitoring data can be identified as service- and resource-level data. Service-level data such as response time can be used to detect performance anomalies when slow response time is defined as an anomaly. However, capturing the

status of underlying cloud environments and exploiting root causes of performance anomalies with only single-variate service-level data is insufficient. Meanwhile, underlying resources affect the performance of cloud applications heavily. Resource-level data mainly includes underlying resource usage like CPU, memory usage. A single resource metric may not precisely reflect the health status of the whole application. Therefore, it is reasonable to detect performance anomalies in cloud applications based on all resource-level monitoring information, which is multivariate time-series data. In addition, when a performance anomaly occurs, we need to pinpoint fine-grained root causes to help operators make decisions for rapid recovery of the cloud application.

Performance diagnosis is defined as detecting abnormal performance phenomena, e.g., degradation, predicting anomalies to forestall future incidents, and localizing the causes of performance anomalies [4]. Recently, studies for performance diagnosis have been developed and mainly focus on performance anomaly detection and root cause localization. For performance anomaly detection, numerous existing methods [5–7] target improving detection accuracy, but their performance is inconsistent in cloud environments. For example, scaling cloud infrastructures will change the data distributions of monitoring

* Corresponding authors.

E-mail addresses: chenpeng@mail.xhu.edu.cn (P. Chen), z.zhao@uva.nl (Z. Zhao).

<https://doi.org/10.1016/j.future.2024.02.014>

Received 21 July 2023; Received in revised form 12 February 2024; Accepted 16 February 2024

Available online 17 February 2024

0167-739X/© 2024 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

data, severely affecting detection performance. Therefore, robust performance anomaly detection is necessary for performance diagnosis to keep performance consistent. As for root cause localization, approaches are still developing [8,9], and most of them focus on service-level or container-level anomalies [10,11], which cannot provide accurate guidance for operators to take actions. For example, only the migration or restart of the entire service can be taken for identified faulty services [12], while service scale-out can be considered when pinpointing indicative faulty metrics. To fill these gaps, we are motivated to develop a performance diagnosis framework to detect performance anomalies with good robustness and identify the root causes at the metric level.

To develop a performance diagnosis framework based on multivariate time-series data, we must address several specific challenges. The primary challenge involves reducing data noise, as monitoring data often contains disruptive noise that can seriously affect diagnostic accuracy. Additionally, our framework prioritizes accurate anomaly detection and the robustness of the detection method across various patterns in monitoring data. Furthermore, another challenge is the real-time localization of fine-grained root causes when anomalies occur to achieve fast application maintenance.

To solve these challenges, the performance diagnosis framework should provide metrics selection methods to filter data noise and reduce data dimensions first. In addition, it is necessary to have good detection performance and fine-grained root cause localization. As for the performance anomaly detection, we mainly focus on our previous work, the deep ensemble method [13], which integrates existing detection methods based on ensemble learning [14]. While the deep ensemble method employs a basic neural network, we consider that incorporating advanced deep learning techniques to enhance detection performance. In addition, prior research solely compared the deep ensemble method to unsupervised detection methods, which is an unfair evaluation. Furthermore, we also explore detection ability by changing the number of labels for training the deep ensemble method. Therefore, we offer extensions and additional evaluation experiments for the deep ensemble method. As for root cause localization, we develop a pipeline including a causal inference method to model anomaly propagation paths and a graph method to determine the root cause node for identifying metric-level root causes. In this paper, we provide a **Fi**ne-grained **R**obust **p**erformance **D**iagnosis (**FIRE**D) framework. Our contributions can be summarized as follows:

- We design an unique integrated framework that combines three crucial components to enhance performance diagnosis: a metrics selection process to filter out useless monitoring metrics, accurate and robust performance anomaly detection, and fine-grained root cause localization.
- We provide an extended version of the deep ensemble method and conduct experiments to assess its enhancements in terms of detection accuracy and robustness. The deep ensemble (LSTM) has the best detection accuracy and robustness. Additionally, we investigate the impact of different amounts of labels on the performance of the deep ensemble method.
- We propose an innovative real-time and fine-grained root cause localization pipeline that leverages a dependency graph and automated random walk execution. This pipeline consistently achieves a high average localization accuracy of over 0.7 when pinpointing root causes, demonstrating its effectiveness.

We organize the rest of the paper as follows. Section 2 reviews existing research about performance diagnosis and introduces anomaly detection and root cause localization methods. In Section 3, a performance diagnosis framework and its main components are introduced in detail. In Section 4, experiments and results for each component of our diagnosis framework are provided. Finally, we provide a discussion and conclusion in Section 5 and Section 6.

2. Related works

Research about performance diagnosis is ongoing rapidly in clouds, e.g., microservice [10], and cloud datacenter [15]. Ibidunmoye et al. [4] review performance anomaly detection and bottleneck identification methods, in which they formulated fundamental research problems, categorized detection methods, and proposed research trends and open challenges. In general, performance diagnosis frameworks include anomaly detection and root cause localization [11,16]. This section will introduce related works on performance anomaly detection and localization methods.

For performance diagnosis, cloud application monitoring data needs to be preprocessed first. Metrics selection is to select metrics related to run-time application status, and it can be used to reduce data dimension and improve detection accuracy and efficiency [17]. In real scenarios, if fewer labels exist, feature selection methods, such as filter, wrapper, and embedded methods, can choose a subset of all features [18]. In the situation that no labels exist, feature extraction methods that create a subset of new features from a combination of existing features can be considered, such as principal components analysis (PCA) [19] and linear discriminant analysis (LDA) [20]. This paper provides metrics selection methods and comparison experiments in Sections 3.3 and 4.2.

2.1. Performance anomaly detection

This section mainly focuses on semi-supervised and unsupervised detection methods because they are more practical for cloud applications by requiring few or no data labels. We will also highlight robustness and ensemble learning research for anomaly detection.

Semi-supervised learning is developed in a situation where fewer labels exist, which is included in weakly supervised learning [21]. For example, Camacho et al. [22] present a semi-supervised approach for anomaly detection. The method extends the unsupervised multivariate statistical network monitoring approach based on PCA by introducing a supervised optimization technique to learn the optimum scaling in the input data. Experiments show that the semi-supervised method performs much better than unsupervised detection.

Unsupervised learning methods work without labels. We classify unsupervised anomaly detection methods into density-based, such as local outlier factor (LOF) [6], distance-based, such as K nearest neighbors (KNN) [23], kernel-based, such as one-class support vector machines (OVSVM) [24], and tree-based, such as isolation forest (IForest) [25] methods. They usually have varying performance for different data [13]. Deep learning-based anomaly detection methods are gaining more attention nowadays because they usually have high detection accuracy [26]. For example, Song et al. [27] provide a deep learning-based anomaly detection method utilizing two parallel graph neural networks for multivariate time-series data. Zhao et al. [28] propose a multivariate KPIs anomaly detection framework which ensembles autoencoders with sRC-RNN, and further provide a dynamic loss function to improve the model training. Unsupervised deep learning methods are developing rapidly and focusing on improving detection accuracy. However, it is vital to consider the time-consuming of training deep learning models. In addition, unsupervised learning methods usually rely heavily on the distribution of the input data. If the data distribution changes significantly, the performance of unsupervised models may degrade.

Researchers usually focus on the data pre-processing phase to improve the robustness of machine learning methods. For example, Bhagoji et al. [29] propose using data transformations, including data dimension reduction via PCA, to enhance the resilience of machine learning methods. These researches show the importance of algorithm robustness, but more methods can be considered for improving algorithm robustness, such as ensemble learning. The fundamental idea behind ensemble learning is to combine multiple detection models with the assumption that the errors of individual models can be compensated

by models of the ensemble [30]. For anomaly detection, ensemble learning can aggregate anomaly scores using techniques such as taking the maximum or computing the average [31].

In summary, weakly supervised learning approaches offer a viable option for performance anomaly detection, particularly in real-world scenarios with limited labeled data. Existing detection methods rarely consider detection accuracy and algorithm robustness simultaneously. In addition, ensemble learning methods are trying to improve detection accuracy by combining the characteristics of multiple models, but most of them integrate detection methods linearly. Therefore, we can consider developing a detection method that integrates existing detection methods in non-linear ways, improving detection accuracy and robustness for cloud applications monitoring data.

2.2. Root cause localization

Studies about localizing root causes for performance anomalies in cloud applications have been developing in recent years. Researchers proposed machine learning, pattern recognition, and graph-based methods with data from cloud applications, such as logs, request execution tracing data, and metrics to localize root causes [32]. Graph-based methods can identify root causes and provide visibility of the issue with a visualized graph. Here, we review graph-based methods as follows.

Graph-based methods identify the root causes by constructing a graph from observational data, and then inferring and ranking the causes based on the graph. Service dependency graphs are usually built based on service deployment and service co-location. For example, Lin et al. [33] provide the Microscope to build the service causal graph based on network connection information. Wu et al. [10] propose MicroRCA to build an attributed graph that models anomaly propagation across services and machines. Cai et al. [34] propose an automatic root cause localization framework, ModelCoder, which includes the deployment graph and service dependency graph building, abnormal node classification, and a fault model to figure out the root cause of faults. Chen et al. [35] introduce the FRL-MFPG, a method that builds the microservice fault propagation graph using link calls and fault correlation and identifies the root cause location through the random walk. These methods, depending on the service dependency graph, can only be used to localize root causes at the service or container level, except at the metric level, which cannot help operators fix performance anomalies accurately.

In addition, the casual graph for monitoring data can be constructed based on causal inference. Chen et al. [36] provide Causeinfer to build the causal graph with the PC (named after the authors, Peter and Clark) algorithm based on the conditional independence test. Meng et al. [16] propose MicroCause, a variant of the PC algorithm that considers the time order of metrics to identify a causal graph of metrics. However, this method also has the time-lag limitations of monitoring data and has a higher computation burden than the PC algorithm. Wu et al. [37] propose MicroDiag, which focuses on applying a DirectLiNGAM to build causal graphs and PageRank to infer root causes, while DirectLiNGAM has a strict non-Gaussian distribution assumption for input data. Ikram et al. [38] introduce the RCD model, which treats anomalies as interventions on the root cause node, utilizing Ψ -PC to identify the interventional target and extract neighboring nodes for determining the actual root causes. The method has good performance on root cause localization, but its execution time can be improved.

We provide a comparison of the above root cause localization research in Table 1. We summarize that the root cause localization pipeline mainly includes two steps: build dependency graph and localize root cause. Furthermore, most research for root cause localization focuses on service level based on deployment graphs or call graphs. Research for metric-level root cause localization based on causal graphs is also developing [39]. To guarantee practicality and accuracy in our diagnosis framework, we develop the root cause localization pipeline

Table 1
Comparison of root cause localization research.

Reference	Year	Build dependency graph	Localize root cause	Root cause level
Microscope [33]	2018	Parallelized PC	BFS	Service level
MicroRCA [10]	2020	Call graph + subgraph extraction	personalized pagerank	Service level
ModelCoder [34]	2021	Deployment graph + service dependency graph	PSO optimized searching	Service level
FRL-MFPG [35]	2023	Link calls + fault correlation analysis + Granger causality	Random walk	Service level
CauseInfer [36]	2016	PC	DFS	Metric level
MicroCause [16]	2021	PCTS	Random walk	Metric level
MicroDiag [37]	2021	DirectLiNGAM	PageRank	Metric level
RCD [38]	2022	Ψ -PC	Extract neighbor nodes	Metric level

including the PC algorithm to build a dependency graph for monitoring data and the random walk to determine the root cause in this paper.

In conclusion, we can implement the performance diagnosis framework to meet requirements in real-life scenarios. As for metrics selection, feature selection and feature extraction methods can be considered based on monitoring data. As for performance detection, we are considering expanding a deep ensemble method with advanced deep learning methods to improve detection accuracy and guarantee robustness. Furthermore, root cause localization based on causal inference and graph methods can be developed to discover root causes in monitoring metrics.

3. Performance diagnosis framework

To build an effective performance diagnosis framework for cloud applications, we first provide the requirements and challenges analysis. Then, we describe our FIRED framework and introduce each component in detail.

3.1. Requirement analysis

Several design requirements must be considered for each component to ensure the development of an effective performance diagnosis system. We identify them from several use cases in the EU project SWITCH¹ and ARTICONF.²

- Data preprocessing to reduce noise. This requirement involves addressing the challenge of noise reduction. Methods should be designed to extract or select the most relevant features that contribute to accurately detecting performance anomalies.
- Anomaly detection for varying performance data. In real-life scenarios, data distributions vary for different performance anomalies, making it necessary to develop an accurate and robust detection method.

¹ <https://www.switchproject.eu/>

² <https://articonf.eu/>

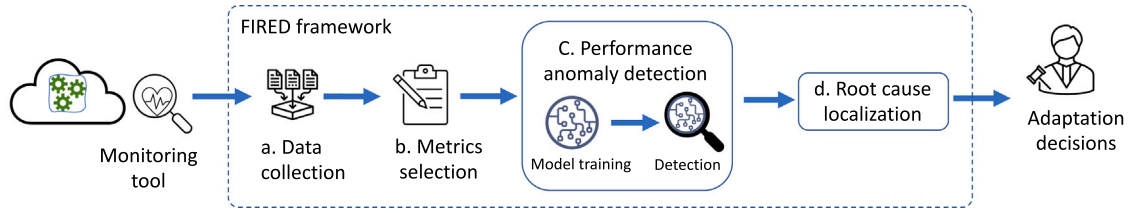


Fig. 1. The FIRED framework for cloud applications has model training and real-time testing. The framework includes a. data collection and b. metrics selection and c. performance anomaly detection. Once an anomaly is detected, d. root cause localization will start to discover the causes of the anomaly. The localization results can be used for the adaptation decisions of cloud applications.

Table 2
Notations and definitions.

Notion	Definition
N	Number of all metrics
n	Number of selected metrics
d	Number of samples in each metric
k	Number of base learners
K^t	Anomaly labels, t is index of timestamps
R_i^t	Collected monitoring data, i is index of metrics, t is index of timestamps.
D_j^t	Data after metrics selection, j is index of data dimensions, t is index of timestamps.
C_k^t	Anomaly scores vector of each base learner, k is index of base learners, t is index of timestamps.
O_k^t	Anomaly scores vector after normalization, k is index of base learners, t is index of timestamps.
G	Dependency graph of metrics

- Root cause localization needs to be fine-grained and accurate. Complex dependencies between services make it challenging to model anomaly propagation paths. In addition, it is difficult to identify the root cause metric considering multiple metrics exist.

To meet these requirements and tackle challenges, we develop a practical performance diagnosis framework that includes data preprocessing technologies to address data challenges, effective performance anomaly detection aiming for high detection accuracy and good algorithm robustness, and fine-grained root cause localization.

3.2. Framework overview

In this paper, we provide a **FI**ne-grained **RO**bstust **PE**rformance **D**iagnosis (FIRED) framework, which can be seen in Fig. 1. The framework works with several steps. First, (a) we collect multivariate time-series monitoring data continuously. This paper concentrates on resource data, such as CPU and memory usage. In addition, fewer labels to indicate performance anomalies of an application will be used in our framework. For collected data, (b) the metrics selection will filter multivariate data with feature selection or extraction methods to select relevant metrics and reduce data dimensions. Subsequently, (c) selected metrics are used to train the performance anomaly detection method. We also use selected metrics of new data and then test them with the trained detection method. Once an anomaly occurs, (d) we start the root cause localization to discover the causes of the anomaly. The localization results can be used to make adaptation decisions for cloud applications.

The diagnosis framework provides the deep ensemble method developed in our previous work [13] for performance anomaly detection. In this paper, we expand the deep ensemble method with advanced deep neural networks and provide comprehensive experiments to evaluate its detection ability. Furthermore, effective root cause localization enables rapid recovery for detected performance anomalies. We develop the root cause localization method based on causal inference and graph methods. We mainly evaluate localization accuracy, which will be introduced in 3.5. The main notations in this paper are given in Table 2.

3.3. Metrics selection

Multivariate data usually contains noise, introducing unnecessary variance into a developed model. Therefore, metrics selection to identify relevant metrics and reduce the data dimension is needed. Considering fewer labels exist, we can easily select relevant metrics from multiple metrics with filter methods. Pearson’s correlation is generally used to measure the relevance between features, and it provides a fast estimation for feature selection. In addition, the feature extraction method PCA can extract the main features in data and reduce data dimensions without labels. This paper mainly focuses on the feature selection method for application monitoring data. We also compare the performance of the feature selection and extraction methods.

For original data, we apply z-score normalization [40] to scale them first. After normalization, the input data is represented by R_i^t ($i = [1, \dots, N]$ denotes the resource metric index, N represents the total number of metrics, and t is the timestamp). We then provide the metrics selection for the input data, including feature selection and extraction methods. After metrics selection, data D_j^t will be used to diagnose the running status of cloud applications. We define n as data dimensions after reduction, and $j = [1, \dots, n]$ is the data dimension index.

With fewer labels, we can extract related resource metrics automatically. We provide a filter method of correlation analysis for all metrics with these labels. We use K^t for time-series data to represent these labels, and R_i^t represents resource metrics. We calculate Pearson’s correlation of the labels with each resource metric.

$$r_i = \frac{cov(K^t, R_i^t)}{\sigma_{K^t} \sigma_{R_i^t}} \tag{1}$$

A significant test for Pearson’s correlation.

$$t_i = r_i \sqrt{\frac{d-2}{1-r_i^2}} \tag{2}$$

Here, d is the number of timestamps and the sample number of each resource metric. To filter out low correlation metrics, we set the threshold $t_i < 0.05$ and $|r_i| > 0.5$ for all correlation results.

Furthermore, the feature extraction technique known as PCA can be employed to reduce the dimensions of the data with numerous variables while retaining the majority of information. We utilize correlation analysis and PCA on the resource metrics R_i^t . We determine the dimensions

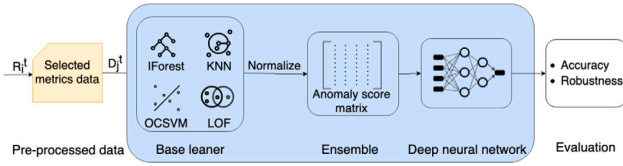


Fig. 2. The deep ensemble pipeline.

after reduction for PCA based on a specified percentage of variance, such as 95%, to retain a significant portion of variance information of original data [41]. We present the comparison of metrics selection methods in Section 4.2.

3.4. Performance anomaly detection

Considering that fewer labels exist in real scenarios, the deep ensemble method works as a weakly supervised learning method to improve detection accuracy. The main idea of the deep ensemble method is integrating existing detection methods with a neural network, and it includes three components: base learner, ensemble, and deep neural network, which can be seen in Fig. 2. The base learner includes four different unsupervised anomaly detection methods. The outputs of these base learners generate an anomaly score matrix. The matrix is used as the input of a deep neural network for training. In this paper, we extend the deep ensemble method by replacing the multi-layer perceptron (MLP) with advanced deep learning methods: long short-term memory (LSTM) [42], and convolutional neural network (CNN) [43]. In our experiments, we evaluate the deep ensemble method for detection accuracy and robustness compared with other detection methods. In addition, we train the deep ensemble method in a weakly supervised manner, which adopts fewer labels for training. We split pre-processed data D_j^t into different amounts of labels as training data and provide experiments to evaluate the impact of different amounts of labels in Section 4.3.2.

3.4.1. Base learner

In order to gain a comprehensive understanding of the characteristics of monitoring data, base learners are designed by selecting four typical methods (IForest, KNN, LOF, OCSVM) because they concentrate on different data features, such as density or distance. We input pre-processed data D_j^t to each base learner. Once the model is initialized and trained with the data, we can obtain the anomaly score vector C_k^t for each base detection method, where k represents the index of the method. In our deep ensemble method, all anomaly score vectors from base learners are assembled, which will be introduced next.

3.4.2. Ensemble

Anomaly score vectors have different scales because they represent different meanings. For instance, KNN calculates the anomaly score based on distance. So, we use z-score normalization [40] to scale them. After normalization, we use O_k^t to represent the anomaly scores vector of each base learner, where k represents the index of the base detection method. Through the integration of these anomaly score vectors, we construct the anomaly scores matrix M as

$$M = \begin{bmatrix} O_1^1 & O_2^1 & O_3^1 & O_4^1 \\ O_1^2 & O_2^2 & O_3^2 & O_4^2 \\ \vdots & \vdots & \vdots & \vdots \\ O_1^t & O_2^t & O_3^t & O_4^t \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

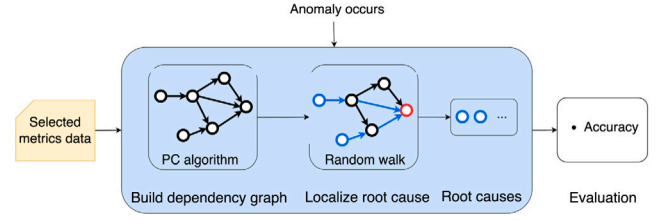


Fig. 3. The root cause localization pipeline.

3.4.3. Deep neural network

In the deep ensemble method, we use an MLP to combine the information extracted by each base learner effectively. The MLP architecture inputs anomaly score matrix M to the input layer. The architecture consists of two hidden layers with numbers of neurons. We use ReLU as the activation function. We set a single neuron for the output layer to produce the probability using the softmax activation function. Let $x = [O_1^t, O_2^t, O_3^t, O_4^t]$ represent the input. We denote $W^{(1)}$, $W^{(2)}$ and $W^{(3)}$ as the weights. We also set the bias for each layer. After the feed-forward calculation, we will get the difference between the predicted and actual results using the cross-entropy error function, also known as the loss function. Optimization aims to minimize the loss function by adjusting the parameters iteratively.

To train the deep ensemble method, we start with preprocessed data, which includes a limited number of data labels. We input the data into four base learners for generating anomaly scores, and then ensemble these scores into a matrix. Following this, the matrix and the subset of labeled data are fed into a neural network, where the objective is to minimize its loss function. Throughout the training process, the neural network refines its parameters to enhance anomaly detection capabilities. Once the model is successfully trained, we introduce testing data to the trained model to effectively identify anomalies within the dataset. It is worth noting this paper provide extensions to replace it with other deep learning methods, LSTM [42], CNN [43]. We evaluate the feasibility of the replacement in our experiments. In addition, we provide experiments to compare the performance of the deep ensemble method with other weakly supervised deep detection methods.

3.5. Root cause localization

Performance anomaly detection allows us to know the status of cloud applications. When an anomaly occurs, localizing the root causes of the anomaly can enable the application to recover effectively. We provide the pipeline of root cause localization for performance anomalies in Fig. 3. The input data consists of selected metrics and anomaly labels. Selected metrics are data after feature selection, which can be identified as CPU-related, memory-related. At the same time, data after feature extraction cannot be used because there is no clear meaning of extracted features. For these selected time-series metrics, we extract their causal relations and build a dependency graph with the PC algorithm. Based on the dependency graph, we use a random walk to find the propagation path and localize root causes. Finally, we evaluate the localization accuracy.

3.5.1. Build dependency graph

The causality between resources and application performance is evident; for example, low network bandwidth will cause high response latency. To extract the relation, causal inference is commonly used in practical applications because of its intuitiveness. The most popular method for constructing a causal graph from observational data is the PC algorithm [44].

Based on the PC algorithm, we build a dependency graph for all selected metrics D_j^t , and the anomaly labels K^t . We define the anomaly labels K^t as an anomaly indicator. In addition, other causal

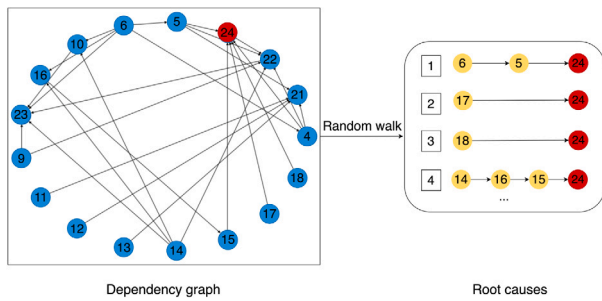


Fig. 4. Visualization of root cause localization pipeline.

inference methods can be used to build the dependency graph, like the additive noise model (ANM) [45]. We also compare their localization performance in our experiments.

3.5.2. Localize root cause

In a dependency graph, there can be many paths that point to the anomaly indicator, making it hard to localize root causes. To solve this problem, we apply a Random Walk algorithm to the dependency graph, which performs well in capturing anomaly propagation. The random walk procedure in a dependency graph is presented in Algorithm 1.

Algorithm 1: Random walk for the causal DAG

Input : DAG G , path length l , start node N

Output: Path points to the start node

```

1 path = [N]
2 while len(path) < l do
3   cur_node = path[-1];
4   if len(list(G.predecessors(cur_node))) > 0 then
5     predecessor =
6       random.sample(list(G.predecessors(cur_node, 1));
7     path.extend(predecessor);
8   else
9     break;
9 return path

```

In this algorithm, we set the anomaly indicator as the start node. Furthermore, we end up with a path pointing to the start node by randomly selecting the predecessors of the current node. We iterate the algorithm many times and get several paths. The last node of each path is regarded as the root cause. By counting and ranking the frequency of root cause nodes, we can get the root cause set and determine the root cause [46].

We provide an example of the root cause localization pipeline in Fig. 4. Nodes 0–23 represent selected metrics, and node 24 represents the anomaly indicator. We first build a dependency graph for these metrics with the PC algorithm. Isolated nodes with no causality relation with others are removed from the dependency graph. Next, we use the random walk algorithm for the dependency graph to get paths pointing to the anomaly indicator and rank all root cause nodes. We can see that there are paths like $6 \rightarrow 5 \rightarrow 24$, $17 \rightarrow 24$, $18 \rightarrow 24$, $14 \rightarrow 16 \rightarrow 15 \rightarrow 24$. After several iterations, we can get the root cause set $\{6, 17, 18, 14, \dots\}$. We then count the number of each node in the root cause set and order them in descending order. The node with the highest frequency will be identified as the root cause.

4. Evaluation and results

We provide different experiments to evaluate each component in the FIRED framework.

- We evaluate metrics selection by comparing the detection performance of base learners with data processed by metrics selection methods.
- We conduct experiments to compare detection performance between the deep ensemble method with other deep detection methods. We also evaluate the impact of different amounts of data labels.
- We check the feasibility of the root cause localization pipeline, compare different causal graph-building methods and observe their time spent.

4.1. Datasets

We apply the decentralized application (DApp) monitoring, SMD, and Vichalana data in our experiments.

4.1.1. DApp monitoring data

The run-time status of a quality critical DApp in cloud environments need to be monitored and diagnosed in real-time. We implement a DApp containing different organizations and peer nodes with Hyperledger Fabric,³ and deploy it on several virtual machines (VMs). In this paper, Prometheus⁴ is utilized for real-time data monitoring and collection, while Caliper⁵ serves the purpose of simulating workload generation.

For the running DApp, various types of monitoring data can be gathered, encompassing service-level information like workloads, blockchain-level data such as transaction numbers, and resource-level data that represents resource usage. Resource-level data can reflect application performance anomalies directly. For example, CPU overload can cause a high response delay. For a DApp, resource usage like high CPU/MEM usage may cause transaction failure. Therefore, we collect resource metrics, including CPU, memory, disk, and network. To simulate transaction failure, we add pressures with *stress-ng*⁶ to a stable running DApp. We focus on inducing I/O anomalies by increasing I/O pressure for 20 min every hour. Throughout a monitoring period of 12 h, we continuously observe the DApp and gather data at 15-second intervals. This experimental setup yields 3237 samples and 229 metrics related to system resources. In addition, an important metric that represents the number of transaction failures can be seen as the anomaly indicator of the DApp. We use this metric to select relevant resource metrics.

4.1.2. Public data

The server machine dataset (SMD), which has been publicly released, is a collection of data obtained by a prominent internet company [47]. This dataset encompasses information from multiple server machines, each recording 38 distinct monitoring metrics. Anomalies in the SMD dataset have been labeled by domain experts.

Vichalana is a dataset intended to detect performance anomalies in API Gateways [48]. This dataset encompasses diverse forms of anomalies, such as instances of high CPU usage, high memory usage. The performance metrics within this dataset have been collected both during normal system operation and under anomalous conditions.

Detailed information about the three datasets used in our experiments can be found in Table 3. The three datasets are all time-series data and structured in a sequence format, where each row represents a specific timestamp, and the columns contain resource information such as the CPU usage, disk space used. The DApp monitoring data is presented in a format of 3237 rows by 229 columns, the SMD data consists of 28,479 rows and 38 columns, while the Vichalana dataset is structured as 45,486 rows by 13 columns. We conduct the metrics selection on the three datasets to reduce noise and improve efficiency, resulting in a dimension reduction to 24, 5, and 6, respectively.

³ <https://www.hyperledger.org/use/fabric>

⁴ <https://prometheus.io/>

⁵ <https://www.hyperledger.org/use/caliper>

⁶ <https://kernel.ubuntu.com/~cking/tarballs/stress-ng/>

Table 3
General information of the three datasets.

Dataset	Number of samples	Number of metrics	Number of metrics after reduction	Anomaly fraction (%)
DApp monitoring data	3237	229	24	28.14
SMD data	28 479	38	5	9.46
Vichalana data	45 486	13	6	6.45

To evaluate our performance diagnosis framework, we only use the DApp monitoring data for the metrics selection evaluation and root cause localization component because we know the detailed information of all metrics. For performance anomaly detection, we use the three datasets.

4.2. Metrics selection evaluation

We conduct experiments to validate the metrics selection component in the performance diagnosis framework. We only use the DApp monitoring data here as we have the detailed information of each metric, for example, CPU usage, system load. The metrics selected by the two methods — correlation analysis and PCA, and all metrics are used as the input to the four base learners. Afterward, we check the effect of metrics selection methods by comparing the detection performances of the four base learners.

4.2.1. Experimental setting

We deploy the DApp on VMs in Azure⁷ with the following specifications: Ubuntu 18.04 as the operating system, 4 CPUs, 16 GB of memory, and 32GiB of storage. On the other hand, the monitor component is deployed on a separate VM similar properties but 2 CPUs, 4 GB of memory.

In our feature selection method, correlation analysis, we apply a threshold of $t_i < 0.05$ and $|r_i| > 0.5$ to filter out metrics with low correlation. As for feature extraction, we set the reduction dimensions to 5 for SMD data, and 6 for Vichalana data.

We evaluate the detection performance of these base learners with F1 score to indicate accuracy, and time spent to indicate efficiency. The F1 score is a metric that takes into account both Precision and Recall, and it can be calculated as below:

$$F1\ score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

4.2.2. Evaluation results

For metrics selection, we apply the correlation analysis and PCA separately on the DApp monitoring data. Fig. 5 shows the correlation analysis result based on r-values in descending order. We calculate the correlation between all monitoring metrics and fewer labels. The results show that metrics like the amount of unevictable memory, iowait have high r-values, which means there is a relationship between these metrics and the occurrence of the transaction failure anomaly. In addition, we reduce data dimensions from 229 to 15 based on PCA method.

We then compare the effects of metrics selection methods based on the performance of base learners. We use the data after correlation analysis, PCA, and without metrics selection as the input of base learners, respectively. The F1 score and time spent of each base learner can be seen in Figs. 6 and 7. In Fig. 6, we can see that after correlation analysis, three base learners, IForest, KNN and OCSVM have the highest F1 scores. For PCA, KNN and LOF have higher F1 scores than without metrics selection. But the F1 scores of IForest and OCSVM are lower than without metrics selection. We can say that metrics selection based

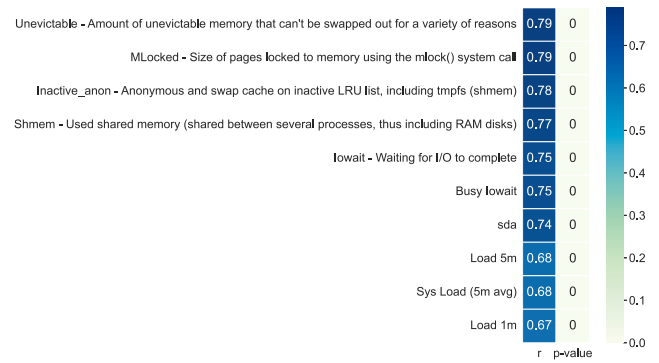


Fig. 5. Top 10 resource metrics with high relevance to performance anomalies.

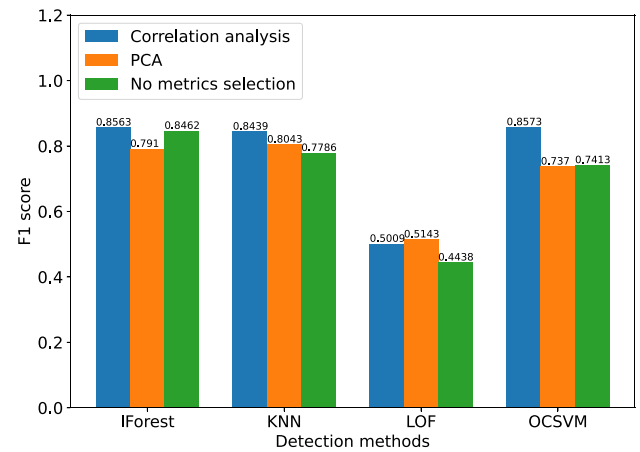


Fig. 6. Detection accuracy by base learners as function of the metrics selection methods.

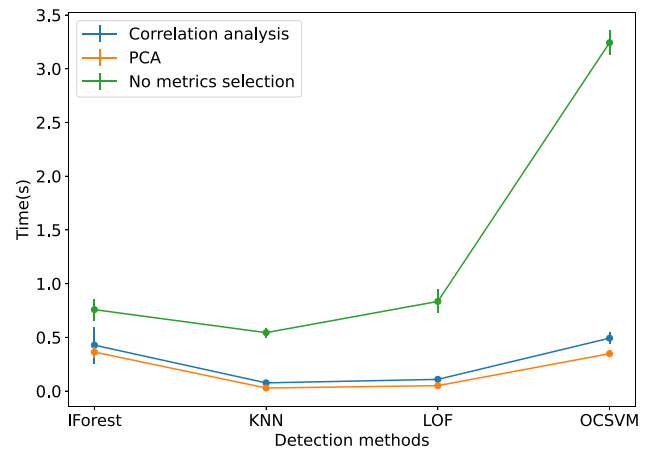


Fig. 7. Time spent by base learners as function of the metrics selection methods.

on correlation analysis improve the detection accuracy. In addition, in Fig. 7, we can see that without metrics selection, the time spent is about 2 to 10 times for each base learner compared with using metrics selection.

In conclusion, we can see that with metrics selection, the detection accuracy is improved and the time spent is reduced compared with no metrics selection. In addition, correlation analysis has better detection accuracy than PCA. Next, we will input data after metrics selection to performance anomaly detection and root cause localization methods.

⁷ <https://azure.microsoft.com/en-us/>

4.3. Performance anomaly detection evaluation

With the three datasets, we conduct experiments to validate detection accuracy and algorithm robustness of the deep ensemble method. We also explore the impact of different amount of labels for the weakly supervised learning method.

4.3.1. Experimental setting

For the deep ensemble method, we design experiments to evaluate detection performance. We use the DApp monitoring data after correlation analysis based on the metrics selection experiments above as the input of the deep ensemble method. For SMD and Vichalana datasets, we use data after PCA as inputs. In our experiments, we combine each base learner with an MLP to train the model and compare its performance with the deep ensemble method. Furthermore, we evaluate the performance of the deep ensemble method with different amounts of labels, taking into consideration the limited availability of labels for monitoring data in real-world scenarios.

The evaluation of these detection models revolves around two key aspects: accuracy and robustness. Detection accuracy is primarily measured using the F1 score. To evaluate the robustness of the detection methods, we conduct tests on different datasets, thereby assessing the consistency of their performance. To quantify the robustness, we calculate the average rank of each detection method. The average rank is inspired by Friedman's M statistic [49], and it is obtained by ordering the average ranks and assigning ranks to the individual method accordingly. The average rank represents a quite useful method for deciding the usability of detection methods [50]. Additionally, we scale all average ranks in the range [0, 1] following the min-max normalization method [40]. The calculation of the robustness score is using the following formula:

$$\text{Robustness score} = \frac{\text{Rank} - \text{Rank}_{\max}}{\text{Rank}_{\min} - \text{Rank}_{\max}} \quad (4)$$

Rank_{\max} represents the highest rank number observed, while Rank_{\min} corresponds to the lowest rank number. In this paper, the Rank_{\max} is 5, and the Rank_{\min} is 1.

The base learners in our approach are configured with specific hyperparameters. Initially, we determine the anomaly fractions for the three datasets. The DApp monitoring data has an approximate anomaly fraction of 0.3. Conversely, we employ the default anomaly fraction of 0.1 for SMD and Vichalana data. For the hyperparameters of each base learner, we specify a tree number of 100 for IForest, set the neighbor number of KNN as 5, and define the neighbor number of LOF as 20.

For the deep ensemble method, we perform the train/test split first. During training, we utilize 50% of the available labels to train the model. We also compare the performance of training with different amounts of labels (10%, 30%, 50%, 70%, 90%). Regarding the hyperparameters of the MLP, the input layer is configured with four neurons to correspond to the four base learners. In addition, we configure each of the two hidden layers with 20 neurons, while the output layer consists of a single neuron. The model training process involves 100 epochs, utilizing a batch size 20. Stochastic gradient descent is performed using the Adam optimizer. We set the learning rate as 10^{-3} . We conduct ten training iterations for the deep ensemble method and present average evaluation metrics, including F1 score and testing times, as final results.

4.3.2. Performance anomaly detection results

A. Comparison with other deep detection methods. We design experiments to compare the detection accuracy and robustness of the deep ensemble method with other deep detection methods. We create deep detection models by combining each base learner with an MLP. Furthermore, we extend the deep ensemble method by replacing MLP with CNN and LSTM and compare their detection performance.

We present a comparison of detection accuracy in Fig. 8. It is evident that the deep ensemble method has the best detection accuracy,

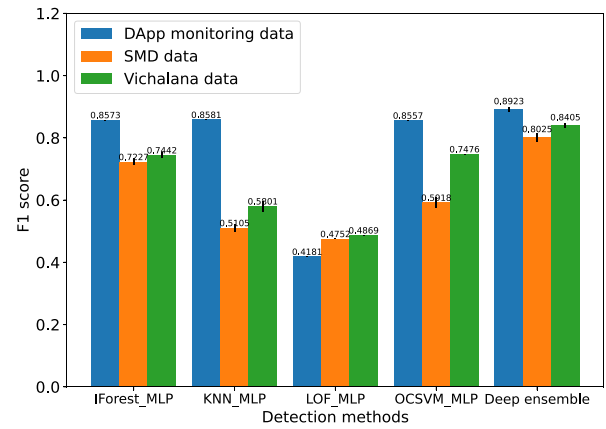


Fig. 8. Detection accuracy of deep detection methods.

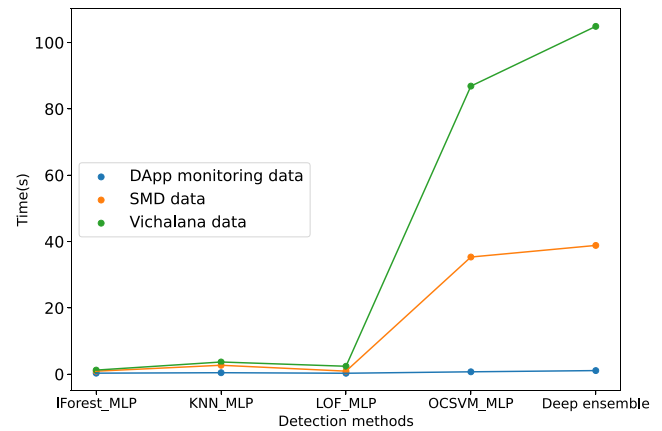


Fig. 9. Time spent of deep detection methods.

showing its stability and adaptability across different datasets. In contrast, other deep detection methods exhibit varying performance. For instance, KNN_MLP performs well on DApp monitoring data. However, because of data characteristics, it has limited effectiveness on SMD and Vichalana data. Furthermore, these deep detection methods heavily rely on the base learner, resulting in only marginal improvements when combined with a deep neural network. For instance, LOF_MLP performs comparatively poorly across all three detection methods. In Fig. 9, we show the test time for each method. We can see that the test time spent for all detection methods on the DApp monitoring data is similar and low, around 1s. For the SMD and Vichalana data, the test time for IForest_MLP, KNN_MLP, and LOF_MLP is relatively low, about 2s. However, for OCSVM_MLP, time spent is relatively high, 38 s for SMD data and 104 s for Vichalana data, respectively. The reason is that OCSVM kernel function calculation is time-consuming, and there is a computational cost for the neural network.

Regarding algorithm robustness, we present the ranking results based on the detection accuracy of all the above methods in Table 4. It is evident that the deep ensemble method exhibits the highest detection accuracy across all three datasets. Conversely, other deep detection methods display inconsistent performance. For instance, IForest_MLP performs well for SMD data, while OCSVM_MLP demonstrates good performance for the DApp monitoring and Vichalana datasets. In addition, LOF_MLP performs quite severely for the three datasets. These findings highlight that the deep ensemble method enhances detection accuracy and algorithm robustness.

As for replacing MLP with CNN and LSTM, we provide the comparison of detection performance in Table 5. It is evident that the

Table 4
Rank result of algorithm robustness.

Method	IForest_MLP	KNN_MLP	LOF_MLP	OCSVM_MLP	Deep_ensemble
DApp monitoring data	3	2	5	4	1
SMD data	2	4	5	3	1
Vichalana data	3	4	5	2	1
Avg rank	2.667	3.333	5	3	1
Robustness score	0.5833	0.4268	0	0.5000	1

Table 5
Performance of different deep ensemble variations.

Detection methods	DApp monitoring data		SMD data		Vichalana data	
	F1 score	Time (s)	F1 score	Time (s)	F1 score	Time (s)
Deep ensemble (MLP)	0.8923 ± 0.0076	1.0887 ± 0.0219	0.8025 ± 0.0101	38.8262 ± 0.0117	0.8405 ± 0.0079	104.8727 ± 0.0757
Deep ensemble (CNN)	0.8839 ± 0.0063	1.1434 ± 0.0913	0.7765 ± 0.0127	36.4284 ± 0.1025	0.8201 ± 0.0059	98.9011 ± 0.2061
Deep ensemble (LSTM)	0.8936 ± 0.0031	1.3831 ± 0.1192	0.8196 ± 0.0087	39.8886 ± 0.3167	0.8483 ± 0.0021	93.1235 ± 0.0999

Table 6
Impact of amounts of labels on the DApp monitoring, SMD, and Vichalana data.

Number of labels	DApp monitoring data		SMD data		Vichalana data	
	F1 score	Time (s)	F1 score	Time (s)	F1 score	Time (s)
10% labels	0.8796 ± 0.0042	1.6796 ± 0.016	0.7729 ± 0.0092	37.3878 ± 0.0103	0.825 ± 0.0283	111.5429 ± 1.0363
30% labels	0.8881 ± 0.0046	1.6797 ± 0.0087	0.7902 ± 0.0163	37.4847 ± 0.1181	0.8374 ± 0.0099	111.4794 ± 0.3982
50% labels	0.8923 ± 0.0076	1.6869 ± 0.0219	0.8025 ± 0.0101	38.8262 ± 0.0117	0.8405 ± 0.0079	105.8727 ± 0.0757
70% labels	0.8948 ± 0.005	1.6868 ± 0.0227	0.8115 ± 0.0088	38.2883 ± 0.0262	0.8457 ± 0.0099	100.3362 ± 0.1859
90% labels	0.8968 ± 0.0047	1.6957 ± 0.0326	0.8142 ± 0.0149	38.3515 ± 0.1824	0.8469 ± 0.0071	101.5478 ± 0.0628

deep ensemble (LSTM) has the highest detection accuracy, which is reasonable because LSTM can extract long-term dependencies in data suitable for time-series data. The deep ensemble (CNN) performs worst because the pooling layer in CNN will compress information, which does not suit time-series data very well. As for spent time, we can see that these deep learning methods take similar time for test data. In contrast, the deep ensemble (MLP) method takes less time because of less computation. In comparison, the deep ensemble (CNN) is slightly faster than LSTM because we use fewer parameters in CNN. Therefore, the deep ensemble method can be extended easily, and replacing the MLP with LSTM can improve detection accuracy.

B. Impact of different amounts of labels. The deep ensemble needs to train with fewer labels. We use 50% labels for all the experiments above. Here, we design an experiment to test the impact of amounts of labels to evaluate the detection ability of the deep ensemble method. The results can be seen in Table 6.

For the three datasets, we set different amounts of labels (10%, 30%, 50%, 70%, 90%) to train the deep ensemble method. We can see that with only 10% labels for training and testing for all data, the deep ensemble method has a higher F1 score than other deep detection methods. In addition, more labels are used for training, and the F1 score is higher, which is easy to explain given that more samples with labels provide more information to learn. Besides, the time spent for each dataset in the table is similar, and this shows that the amounts of labels for training the model have little effect on the test time. To conclude, the deep ensemble method can achieve superior performance with fewer labels, such as 10%, to train, and the trained models can be used on other data with high detection accuracy.

In summary, we evaluate the performance anomaly detection in the FIRED diagnosis framework. Our experiments show that the deep ensemble method has the best detection accuracy and algorithm robustness compared to other deep detection methods. Furthermore, the deep ensemble (LSTM) demonstrates the highest detection accuracy among the variations. In addition, as a weakly supervised learning method, the deep ensemble method can get superior detection performance with fewer labels, such as 10% of labels.

4.4. Root cause localization evaluation

We conduct experiments to validate the feasibility of root cause localization in the performance diagnosis framework. Our experiments

are implemented based on DApps monitoring data because we have a clear description of each metric. We will identify which metrics in the DApps monitoring data cause performance anomalies.

4.4.1. Experimental setting

We apply root cause localization methods on the DApps monitoring data. In Table 7, we provide 24 selected metrics and 1 anomaly indicator of DApp monitoring data. We classify them into CPU/MEM/NET/Disk related metrics. For the DApp we are monitoring, we add I/O pressure to inject anomalies. The root causes are I/O related, as shown in Table 7. As for methods in the localization pipeline, we set α in the PC algorithm as 0.05, and the iteration for the random walk is 500.

To evaluate the accuracy of root cause localization, we use two performance metrics: $AC@k$ and Avg . These two metrics are most commonly used to evaluate the rank result of the root cause localization task [10,16]. $AC@k$ represents the probability that top k results localized by algorithms include the real root causes for a given anomaly. When the k is small, the higher $AC@k$ indicates the algorithm identifies the actual root cause more accurately. We calculate $AC@k$ as follows:

$$AC@k = \frac{\sum_{i < k} R[i] \in V_{rc}}{\min(k, |V_{rc}|)} \quad (5)$$

where $R[i]$ is the result of the rank of all metrics for the anomaly. V_{rc} is the root cause set of the anomaly. $Avg@k$ evaluates the overall performance of the localization algorithm by computing the average $AC@k$. The calculation is as follows:

$$Avg@k = \frac{1}{k} \sum_{1 \leq j \leq k} AC@j \quad (6)$$

We use $k = 1, 2, 3, 4$ in our experiments to give a comprehensive evaluation of localization accuracy.

4.4.2. Root cause localization results

When an anomaly is detected, we initiate root cause localization. Anomaly injection yields multiple 20-minute anomaly periods. We select 5 of them randomly and compare the localization accuracy based on two dependency graph-building methods: the PC and the ANM algorithms. The comparison result is shown in Fig. 10. To evaluate the localization accuracy, we use the Avg metric with different values of k due to the uncertain number of predicted root causes. We can see that

Table 7
Description of selected resource metrics.

Index	Type	Metric	Ground truth
0	Memory related	Unevictable - Amount of unevictable memory that cannot be swapped out for a variety of reasons	
1	Memory related	Size of pages locked to memory using the mlock() system call	
2	Memory related	Inactive_anon - Anonymous and swap cache on inactive LRU list, including tmpfs (shmem)	
3	Memory related	Shmem - Used shared memory (shared between several processes, thus including RAM disks)	
4	CPU related	Iowait - Waiting for I/O to complete	✓
5	CPU related	Busy Iowait	✓
6	Disk related	sda	✓
7	CPU related	Load 5 m	
8	CPU related	Sys Load (5 m avg)	
9	CPU related	Load 1 m	
10	CPU related	CPU Busy	
11	Memory related	Pagesout - Page out operations	
12	Disk related	sda - Successfully written bytes	✓
13	Disk related	sda - Written bytes	✓
14	Disk related	sda - discard	✓
15	Network related	OutOctets - Sent octets	
16	Network related	trans eth0	
17	Disk related	Processes blocked waiting for I/O to complete	✓
18	Memory related	Dirty - Memory which is waiting to get written back to the disk	
19	CPU related	Sys Load (15 m avg)	
20	CPU related	Load 15 m	
21	Memory related	Writeback - Memory which is actively being written back to disk	
22	Disk related	sda - Writes completed	✓
23	CPU related	Idle	
24	Anomaly indicator	txn_fail_label	

the PC algorithm performs better for different anomaly points than the ANM algorithm. In addition, for anomaly_point_1 and anomaly_point_2, we can see that the ANM algorithm does not discover real root causes because the dependency graph does not extract causality relations from data. Therefore, the PC algorithm has better localization accuracy and more stable performance for the DApp monitoring data.

We also show the detailed localization performance of the PC algorithm in Table 8. We calculate the detection accuracy of AC@1, AC@2, AC@3, AC@4, and Avg and present the localization results. We can see that the real root cause can be localized directly for different anomaly points. As for anomaly_point_2 and anomaly_point_3, we can see that

Table 8
Root cause localization accuracy for anomalies with the PC algorithm.

Metric	AC@1	AC@2	AC@3	AC@4	Avg	Time (s)
Anomaly_point_1	1	0.5	–	–	0.75	0.843
Anomaly_point_2	1	1	1	0.75	0.9375	0.797
Anomaly_point_3	1	0.5	0.667	0.75	0.729	0.591
Anomaly_point_4	1	–	–	–	1	0.688
Anomaly_point_5	1	0.5	–	–	0.75	0.913

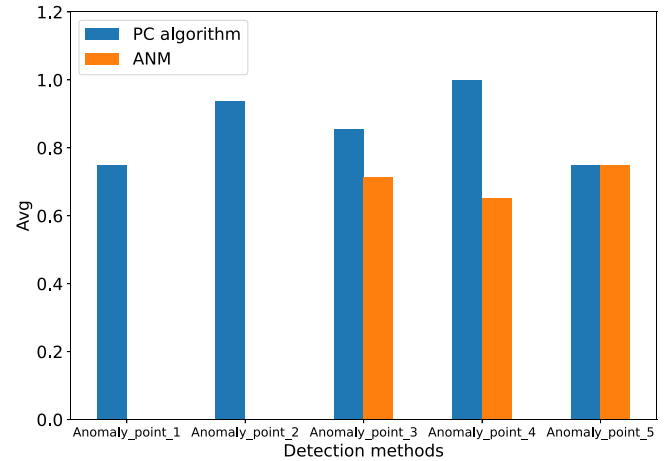


Fig. 10. Root cause localization accuracy based on different dependency graph building methods.

multiple root causes are discovered, including many real root causes, so the localization accuracy is high. Also, we provide the time spent on building a dependency graph and localizing root causes; we can see that the localization for these anomalies can be completed within 1s, which means that the localization can be done in real time with given data.

In conclusion, our experiments demonstrate the feasibility of real-time root cause localization in the performance diagnosis framework. In addition, we build the dependency graph and localize root causes at the fine-grained. Our experiments also show that the localization based on the PC algorithm is accurate and fast for the DApp monitoring data.

5. Discussion

For performance diagnosis, this paper introduces the FIRED framework, which includes metrics selection, performance anomaly detection, and root cause localization components. We design several experiments to validate the FIRED framework. Nevertheless, areas in the framework and experiments can still be further improved.

Regarding metrics selection, we will consider combining correlation analysis and PCA to extract useful information in data. Additionally, it is worth noting that the effectiveness of the deep ensemble method heavily relies on the chosen base learners, as their outputs are integrated as inputs for subsequent processing. This paper selects four typical base detection methods manually. Future research will be beneficial to explore automated approaches for selecting suitable base learners based on the data distributions. Furthermore, for the deep ensemble (LSTM) method, which demonstrates the highest detection accuracy, we intend to conduct hyperparameter tuning to optimize its performance. In the root cause localization pipeline, we will develop advanced methods to build the dependency graph, based on graph autoencoder [51], to extract complex causal relations and improve localization accuracy.

This paper concentrates on resource metrics, while there are many other monitoring data for an application. For example, DApps have blockchain-level data like transaction numbers in each peer node or

committed blocks, which can be used for service anomaly detection and root cause localization. In addition, recovery from performance anomalies is also essential to ensuring the running of a cloud application. Based on diagnosis results, an automatic response, such as scaling VMs or migrating services to solve anomalies in real-time before users realize it is needed. We can see that an automatic operation system, including monitoring, diagnosis, and recovery, is needed for a cloud application. This paper provides a novel performance diagnosis framework for the operation system. More work needs to be done for cloud application performance management in the future.

6. Conclusion

In this paper, we present an integrated performance diagnosis framework named FIRED, which can effectively detect performance anomalies and localize the root causes of cloud applications. The deep ensemble method achieves better detection accuracy and robustness. The proposed root cause localization method can accurately identify root causes in a metric granularity. The FIRED framework focuses on weakly supervised learning, considering fewer labels exist in real scenarios and provides metrics selection to reduce data dimensions and improve diagnosis performance. We provide experiments to evaluate the effect of metrics selection, and results show that it can help reduce time spent and improve detection accuracy.

We introduce the deep ensemble method as a solution for performance detection, addressing detection accuracy and algorithm robustness requirements. Our survey shows that various detection methods have been developed, each focusing on different data features. Consequently, we provide and expand the deep ensemble method that leverages ensemble learning to integrate existing detection methods non-linearly. Our experiments compare the deep ensemble method with other deep detection methods, demonstrating that it achieves the highest detection accuracy and exhibits superior algorithm robustness. Furthermore, we investigate the impact of label quantities in the deep ensemble method, revealing that the method performs exceptionally well even with a limited number of labels.

We provide the root cause localization pipeline to identify the root causes of performance anomalies accurately and in real-time. The pipeline includes building the dependency graph with the PC algorithm, and localizing and ranking root causes with a random walk. We apply the localization pipeline to the DApp monitoring data. We compare the PC and ANM algorithms to build the dependency graph, and the results show that the PC algorithm has an average localization accuracy higher than 0.7. Our experiments also demonstrate the feasibility of real-time root cause localization based on the PC algorithm. More research into improving localization accuracy can be considered in the future.

In the future, we will focus on improving the performance diagnosis framework. For performance anomaly detection, accuracy, robustness, and efficiency of the deep ensemble method can be explored more. As for root cause localization, graph methods to extract non-linear relations between monitoring data need to be developed. In addition, diagnosis results based on the FIRED framework can be used for exploiting operation strategies to implement rapid recovery. Therefore, an automatic operation system for cloud applications that includes monitoring, diagnosis, and adaptation strategies can be developed.

CRedit authorship contribution statement

Ruyue Xin: Writing – original draft, Methodology, Data curation, Conceptualization. **Peng Chen:** Validation, Funding acquisition, Conceptualization. **Paola Grosso:** Writing – review & editing, Supervision. **Zhiming Zhao:** Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This research work is funded by the EU Horizon 2020 and Horizon Europe research and innovation program under grant agreements 825134 (ARTICONF project), 824068 (ENVRIFAIR project) and 101094227 (BlueCloud 2026). The work is also supported by LifeWatch ERIC, and Dutch NWO LTER-LIFE project. The work is also partially funded by the Science and Technology Program of Sichuan Province under Grant 2020JDR0067 and 24NSFTD0025.

References

- [1] H. Zhou, X. Ouyang, Z. Ren, J. Su, C. de Laat, Z. Zhao, A blockchain based witness model for trustworthy cloud service level agreement enforcement, in: *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, IEEE, 2019, pp. 1567–1575.
- [2] P. Chen, Y. Xia, S. Pang, J. Li, A probabilistic model for performance analysis of cloud infrastructures, *Concurr. Comput.: Pract. Exper.* 27 (17) (2015) 4784–4796.
- [3] H. Zhao, Y. Wang, J. Duan, C. Huang, D. Cao, Y. Tong, B. Xu, J. Bai, J. Tong, Q. Zhang, Multivariate time-series anomaly detection via graph attention network, in: *2020 IEEE International Conference on Data Mining, ICDM, IEEE, 2020*, pp. 841–850.
- [4] O. Ibidunmoye, F. Hernández-Rodríguez, E. Elmroth, Performance anomaly detection and bottleneck identification, *ACM Comput. Surv.* 48 (1) (2015) 1–35.
- [5] M. Hu, Z. Ji, K. Yan, Y. Guo, X. Feng, J. Gong, X. Zhao, L. Dong, Detecting anomalies in time series data via a meta-feature based approach, *IEEE Access* 6 (2018) 27760–27776.
- [6] M.M. Breunig, H.-P. Kriegel, R.T. Ng, J. Sander, LOF: identifying density-based local outliers, in: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 2000, pp. 93–104.
- [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, *Comput. J.* 65 (11) (2022) 2909–2925.
- [8] Y. Gan, Y. Zhang, K. Hu, D. Cheng, Y. He, M. Pancholi, C. Delimitrou, Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices, in: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 19–33.
- [9] Y. Song, R. Xin, P. Chen, R. Zhang, J. Chen, Z. Zhao, Autonomous selection of the fault classification models for diagnosing microservice applications, *Future Gener. Comput. Syst.* 153 (2024) 326–339.
- [10] 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.
- [11] G. Yu, P. Chen, H. Chen, Z. Guan, Z. Huang, L. Jing, T. Weng, X. Sun, X. Li, MicroRank: End-to-end latency issue localization with extended spectrum analysis in microservice environments, in: *Proceedings of the Web Conference 2021*, 2021, pp. 3087–3098.
- [12] 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.
- [13] 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, *J. Cloud Comput.* 12 (1) (2023) 1–16.
- [14] A. Galicia, R. Talavera-Llames, A. Troncoso, I. Koprinska, F. Martínez-Álvarez, Multi-step forecasting for big data time series based on ensemble learning, *Knowl.-Based Syst.* 163 (2019) 830–841.
- [15] A. Roy, D. Bansal, D. Brumley, H.K. Chandrappa, P. Sharma, R. Tewari, B. Arzani, A.C. Snoeren, Cloud datacenter sdn monitoring: Experiences and challenges, in: *Proceedings of the Internet Measurement Conference 2018*, 2018, pp. 464–470.
- [16] 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.

- [17] S. Khalid, T. Khalil, S. Nasreen, A survey of feature selection and feature extraction techniques in machine learning, in: 2014 Science and Information Conference, IEEE, 2014, pp. 372–378.
- [18] G. Chandrashekar, F. Sahin, A survey on feature selection methods, *Comput. Electr. Eng.* 40 (1) (2014) 16–28.
- [19] J. Yang, D. Zhang, A.F. Frangi, J.-y. Yang, Two-dimensional PCA: a new approach to appearance-based face representation and recognition, *IEEE Trans. Pattern Anal. Mach. Intell.* 26 (1) (2004) 131–137.
- [20] X. Wei, W.B. Croft, LDA-based document models for ad-hoc retrieval, in: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2006, pp. 178–185.
- [21] Z.-H. Zhou, A brief introduction to weakly supervised learning, *Natl. Sci. Rev.* 5 (1) (2018) 44–53.
- [22] J. Camacho, G. Macia-Fernandez, N.M. Fuentes-García, E. Saccenti, Semi-supervised multivariate statistical network monitoring for learning security threats, *IEEE Trans. Inf. Forensics Secur.* 14 (8) (2019) 2179–2189.
- [23] S. Ramaswamy, R. Rastogi, K. Shim, Efficient algorithms for mining outliers from large data sets, in: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, 2000, pp. 427–438.
- [24] B. Schölkopf, J.C. Platt, J. Shawe-Taylor, A.J. Smola, R.C. Williamson, Estimating the support of a high-dimensional distribution, *Neural Comput.* 13 (7) (2001) 1443–1471.
- [25] F.T. Liu, K.M. Ting, Z.-H. Zhou, Isolation forest, in: 2008 Eighth Ieee International Conference on Data Mining, IEEE, 2008, pp. 413–422.
- [26] S. Qi, J. Chen, P. Chen, P. Wen, X. Niu, L. Xu, An efficient GAN-based predictive framework for multivariate time series anomaly prediction in cloud data centers, *J. Supercomput.* (2023) 1–26.
- [27] 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 Gener. Comput. Syst.* 145 (2023) 77–86.
- [28] N. Zhao, B. Han, R. Li, J. Su, C. Zhou, A multivariate KPIs anomaly detection framework with dynamic balancing loss training, *IEEE Trans. Netw. Serv. Manag.* (2022).
- [29] A.N. Bhagoji, D. Cullina, C. Sitawarin, P. Mittal, Enhancing robustness of machine learning systems via data transformations, in: 2018 52nd Annual Conference on Information Sciences and Systems, CISS, IEEE, 2018, pp. 1–5.
- [30] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*, Chapman and Hall/CRC, 2019.
- [31] C.C. Aggarwal, S. Sathé, Theoretical foundations and algorithms for outlier ensembles, *Acm Sigkdd Explor. Newslett.* 17 (1) (2015) 24–47.
- [32] P. Notaro, J. Cardoso, M. Gerndt, A survey of AIOps methods for failure management, *ACM Trans. Intell. Syst. Technol.* 12 (6) (2021) 1–45.
- [33] 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.
- [34] Y. Cai, B. Han, J. Li, N. Zhao, J. Su, Modelcoder: A fault model based automatic root cause localization framework for microservice systems, in: 2021 IEEE/ACM 29th International Symposium on Quality of Service, IWQOS, IEEE, 2021, pp. 1–6.
- [35] Y. Chen, D. Xu, N. Chen, X. Wu, FRL-MFPG: Propagation-aware fault root cause location for microservice intelligent operation and maintenance, *Inf. Softw. Technol.* 153 (2023) 107083.
- [36] P. Chen, Y. Qi, D. Hou, CauseInfer: Automated end-to-end performance diagnosis with hierarchical causality graph in cloud environment, *IEEE Trans. Serv. Comput.* 12 (2) (2016) 214–230.
- [37] 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.
- [38] A. Ikram, S. Chakraborty, S. Mitra, S. Saini, S. Bagchi, M. Kocaoglu, Root cause analysis of failures in microservices through causal discovery, *Adv. Neural Inf. Process. Syst.* 35 (2022) 31158–31170.
- [39] R. Xin, P. Chen, Z. Zhao, Causalrca: Causal inference based precise fine-grained root cause localization for microservice applications, *J. Syst. Softw.* 203 (2023) 111724.
- [40] C. Saranya, G. Manikandan, A study on normalization techniques for privacy preserving data mining, *Int. J. Eng. Technol. (IJET)* 5 (3) (2013) 2701–2704.
- [41] H. Abdi, L.J. Williams, Principal component analysis, *Wiley Interdiscip. Rev.: Comput. Statist.* 2 (4) (2010) 433–459.
- [42] A. Graves, J. Schmidhuber, Framewise phoneme classification with bidirectional LSTM and other neural network architectures, *Neural Netw.* 18 (5–6) (2005) 602–610.
- [43] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, *Adv. Neural Inf. Process. Syst.* 25 (2012) 1097–1105.
- [44] M. Kalisch, P. Bühlman, Estimating high-dimensional directed acyclic graphs with the PC-algorithm, *J. Mach. Learn. Res.* 8 (3) (2007).
- [45] P.O. Hoyer, D. Janzing, J.M. Mooij, J. Peters, B. Schölkopf, et al., Nonlinear causal discovery with additive noise models, in: NIPS, Vol. 21, Citeseer, 2008, pp. 689–696.
- [46] 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.
- [47] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, D. Pei, Robust anomaly detection for multivariate time series through stochastic recurrent neural network, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 2828–2837.
- [48] D. Geethika, M. Jayasinghe, Y. Gunarathne, T.A. Gamage, S. Jayathilaka, S. Ranathunga, S. Perera, Anomaly detection in high-performance API gateways, in: 2019 International Conference on High Performance Computing & Simulation, HPCS, IEEE, 2019, pp. 995–1001.
- [49] H.R. Neave, P.L. Worthington, *Distribution-Free Tests*, Unwin Hyman, London, 1988.
- [50] S.M. Abdulrahman, P. Brazdil, J.N. van Rijn, J. Vanschoren, Speeding up algorithm selection using average ranking and active testing by introducing runtime, *Mach. Learn.* 107 (2018) 79–108.
- [51] I. Ng, S. Zhu, Z. Chen, Z. Fang, A graph autoencoder approach to causal structure learning, 2019, arXiv preprint arXiv:1911.07420.



Ruyue Xin got her Doctor degree in the Multiscale Networked Systems (MNS) research group, University of Amsterdam, the Netherlands. She received the M.Sc. degree in the School of Systems Science, Beijing Normal University, Beijing, China in 2019. Her research interests include AIOps, cloud computing, causality inference.



Peng Chen is currently a professor with School of Computer and Software Engineering, Xihua University, China. He also was a visiting scholar in the Multiscale Networked Systems (MNS) research group, University of Amsterdam, the Netherlands in 2021. He received the B.E. degree in computer science from University of Electronic Science and Technology of China, Chengdu, China in 2001, the M.Sc. degree in computer software and theory from Peking University, Beijing, China in 2004 and Ph.D. degree in computer science from Sichuan University, Chengdu, China in 2017. His research interests include machine learning and service computing.



Paola Grosso is currently a professor with the University of Amsterdam where she leads the Multiscale Networked Systems research group (mns-research.nl). Her work focus on the creation of sustainable and secure e-infrastructures and relying on the provisioning and design of programmable networks. She has an extensive list of publications on the topic and contributes to several national and international projects.



Zhiming Zhao, is currently an associate professor in the Multiscale Networked Systems (MNS) research group, University of Amsterdam, the Netherlands. He leads a team on “Quality Critical Distributed Computing” in the System and Networking Lab (SNE). His research focuses on innovative programming and control models.