TS-InvarNet: Anomaly Detection and Localization based on Tempo-spatial KPI Invariants in Distributed Services

Zijun Hu†, Pengfei Chen*, Guangba Yu†, Zilong He† and Xiaoyun Li†
Sun Yat-Sen University, China
Email: *chenpf7@mail.sysu.edu.cn, †{huzj, yugb5, hezlong, lixy223}@mail2.sysu.edu.cn

***Abstract*—Modern industrial systems are often large-scale distributed systems composed of dozens to thousands of services, leading to difficulty in anomaly detection and localization. KPIs (Key Performance Indicators) record the states of different services and are presented as time series, which reflect the status of the system. However, due to the dynamic and complex periodic patterns embedded in KPIs, pinpointing anomalous behavior of these multivariate time series data quickly and accurately is a challenging problem. The current state-of-the-art deep-learning-based anomaly detection methods model global inter-KPI dependency, causing the limited ability to detect local subtle anomalies and poor interpretability. In practice, interpreting anomalies can accelerate problem localization and further troubleshooting. In this study, we propose *TS-InvarNet*, an interpretable end-to-end anomaly detection and diagnosis framework based on tempo-spatial KPI invariants. Extensive empirical studies on three real-world industrial datasets and a widely-used open-source system demonstrate that *TS-InvarNet* can outperform state-of-the-art baseline methods in detection and diagnosis performance. Specifically, *TS-InvarNet* increases best F1-scores by up to 27% compared to the baselines.**

***Keywords*-Anomaly detection; Root cause analysis; Invariant network; KPIs; Correlation;**

I. Introduction

Modern industrial systems are often large-scale distributed systems composed of dozens to thousands of services running in different machines. For instance, there are more than 3000 services in the WeChat system, which are running over 20000 machines [1]. Each service in the system runs as a set of instances and communicates with other services through HTTP or RPC. Such a massive quantity of services and the complex dependencies among them render distributed systems often fail due to various reasons, such as network delay, hardware failures, and application faults [2]. Once a fault occurs, it may affect the user experience and even cause economic losses and other unexpected consequences. For example, the cost of a 7-hour DNS outage at Facebook is estimated to be as high as $47 billion [3]. Therefore, it is essential to detect the anomaly and locate the root cause at ultra-fast speeds.

The status of the system can often be reflected in KPIs, which record the states of different services and present them as time series. Therefore, closely monitoring and analyzing various KPIs (e.g., CPU load and network usage) collected from each instance of services is a mainstream approach to detect and locate anomalies in academia and industry [4]–[11]. However, anomaly detection is becoming increasingly challenging in large-scale systems due to the increasing data volatility and data modality. In particular, the diverse and complex periodic patterns arise in KPIs, leading to difficulty in anomaly detection and further trouble-shooting. Further, the periodic patterns may be dynamic and sometimes deflect from the normal status. Taking online retailers as an example, the sales amount presents a daily cycle but can change significantly when there are big promotions such as black Friday [12]. Moreover, interpretability, which is often overlooked, should be a key capability for anomaly detection. In practice, anomaly detection is not a separate part but serves for root cause analysis and trouble-shooting. The detailed interpretation helps critical decision-making and speeds up the process for operators to diagnose performance issues.

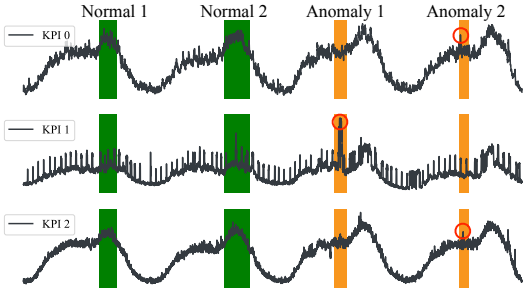


Fig. 1: Some real-world KPI data collected from a larger-scale distributed system in a big company.

Recent work on time series anomaly detection can be categorized into multivariate anomaly detection [4], [13]–[15] and univariate anomaly detection [9], [16]. Univariate anomaly detection approaches, mainly based on one specific KPI, model the temporal dependency but cannot capture complex spatial relationships [4]. They are more prone to identify the normal changes as anomalies, causing more false alarms. In contrast, multivariate anomaly detection approaches can learn the intrinsic connections among KPIs, well representing the system's overall status. Recent popular multivariate anomaly detection are based on deep learning approaches, such as Variational Autoencoders(VAE) [4], [14], [17] and Generative Adversarial Network(GAN) [5], [18], [19]. While deep learning based approaches obtained competitive performance on public datasets, there are concerns about their practical ability for critical decision-making or trouble-shooting because of their limited interpretability [7], [8], [20]. Moreover, the feasibility of deploying and serving computationally expensive VAEs or GANs independently for each server is challenging for the technology companies operating thousands of servers

[7].

In addition, deep learning based approaches mainly focus on the global multivariate time series and learn inter-KPIs dependency while may neglect the importance of subtle local information. As shown in Fig. 1, not every KPI performs abnormally when an anomaly appears in a large system. The value of *KPI 0* and *KPI 2* just perform some sudden increase in *Anomaly 2*, which may not be detected by methods based on modeling global inter-KPI dependency. This is because the *Anomaly 2* is determined as a normal fluctuation of the overall status in those methods. Further, ignoring local information may lead to the limited ability to interpret an anomaly. Though some recent deep learning based methods have provided interpretation based on some raw values generated by the deep neural networks (e.g., the reconstruction probabilities used in [4], [21]), these methods may cause some misinterpretations since anomalies may bring bias to the learned embeddings [8]. They still lose insights into the underlying root causes.

To address the drawbacks of existing work, this paper proposes an interpretable multivariant anomaly detection and localization framework based on tempo-spatial KPI invariants, named *TS-InvarNet*. Our key idea is based on the existence of a stable relationship between KPIs (details shown in Sec.II). *TS-InvarNet* primarily comprises four procedures, including *shape-based clustering*, *invariants mining*, *anomaly detection* and *anomaly localization*. Considering the increasing amount of KPIs, the *shape-based clustering* module first adopts shape-based clustering to exclude repetitive and redundant invariants to accelerate the construction of an invariant network. *Invariants mining* then learns complicated global and local dependencies by tempo-spatial models. Finally, *anomaly detection* detects anomalies by the evolution of the global invariant network while *anomaly localization* performs anomaly localization based on interpreting the changes of local invariants.

In general, our contributions of this paper are four-fold:

- We propose an interpretable end-to-end framework for anomaly detection and localization based on tempo-spatial KPI invariant network in distributed services.
- We introduce a shape-based clustering and a tempo-spatial model to accelerate the build procedure of accurate invariant networks.
- We adopt causal analysis to locate the root cause KPI among many abnormal KPIs. The abnormal KPIs are determined by our method based on the variation of the invariant network without system topology or other extra information.
- We design and implement *TS-InvarNet* to evaluate the effectiveness of our approach with real-world data. The experiment results show that our approach achieves the best F1-score higher than 94.7%, which outperforms state-of-the-art approaches by up to 27%. The ablation studies further demonstrate the great effectiveness of shaped-based clustering in reducing the complexity of mining invariants. The training time of *TS-InvarNet* takes only 1/3 of the time of the original *TS-InvarNet* without clustering.

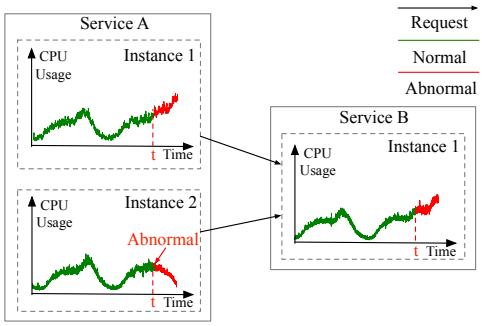


Fig. 2: An example of KPIs' change on fault of Instance 2 in Service A. The red lines signal the occurrence of the anomaly.

The rest of the paper is structured as follows. Section II presents our motivation. Section III introduces the overview of *TS-InvarNet* framework and depicts the design of *TS-InvarNet*. Section IV shows the experimental evaluation. we review the previous related work in section V and Section VI concludes this paper.

II. Motivation

In this section, we first observe the KPIs of different services in a distributed system. Studying the changes in KPIs offers insights to design our anomaly detection and localization framework. Fig. 2 shows three KPIs' changes belonging to three service instances. We present a typical case where a fault occurs at time t on *Instance 2* of *Service A*.

Stable relationships, namely invariants, exist among KPIs. We observe some stable relationships among the KPIs of the distributed services. If the stable correlation holds all the time, it is considered an invariant. More formally, an invariant can be defined as:

Definition 1: Given two variables, X and Y, if there exists a function $f(\cdot)$, which makes $Y = f(X)$. If the equation does not change, $X \leftrightarrow Y$ is defined as an invariant.

As shown in Fig. 2, *Service B* is a down-streaming service of *Service A*. As more users visit the *Service A*, the workload of *Service A* increases and its CPU usage varies with the workload. The *Service A* will send more requests to its downstream *Service B*, leading to an increase in CPU usage of *Service B*. Thus there exist stable correlations between CPU usage of *Service A*'s instances and *Service B*'s instances. Because of the load balancer, there also exists an invariant between CPU usage of two instances in *service A*. Therefore, we get three invariants, namely:
(Instance 1 of A).CPU usage $\leftrightarrow$ *(Instance 2 of A).CPU usage*,
(instance 1 of A).CPU usage $\leftrightarrow$ *(Instance 1 of B).CPU usage*,
(Instance 2 of A).CPU usage $\leftrightarrow$ *(Instance 1 of B).CPU usage*.

When a fault occurs, the invariant between KPIs will be broken. Without more information about systems, a solution to settle the fault is to find the change of patterns between two instances' KPIs compared with history. There used to be three invariants among the three instances' CPU usage. But when a fault occurs in *Instance 2*, two invariants of *Instance 2*'s CPU usage are broken while the third one still

exists. Thus we can determine the *Instance 2* failed. There are more subtle anomalies that only manifest locally in distributed services. If not taken seriously, those anomalies will lead to serious failures. The subtle anomalies can be ignored and are hard to interpret from the view of the overall system. While with invariants, we can capture both the global and local relationships amongst KPIs to tackle these problems.

Invariants can identify anomalies and interpret them. Univariate anomaly detection methods are more prone to raise an error alert and cannot capture the complex spatial relationship. Three KPIs in the case will be detected as anomalous but they cannot figure out where the most anomalous KPI is, not conducive to trouble-shooting. Multivariate anomaly detection methods model the global inter-KPI dependency, well representing the system's overall status. But they pay little attention to local information. The output of these methods is only the overall status of the system. So the three KPIs are also all detected as anomalous but it is hard for them to identify a critical anomalous KPI or to conduct further root cause analysis. An invariant network can be generated by capturing pairwise stable relationships amongst KPIs. The evolution of invariant networks provides some potential global information about abnormal system behaviors, which can help to identify an anomaly. In addition, the violation of the local invariants helps to find the underlying correlation among anomalous KPIs. Invariant network represents the information of the systems' overall status while invariants provide local information to detect subtle anomalies and interpret them. It can further perform root cause analysis based on the interpretation.

III. Approach

A. Overview of TS-InvarNet

Fig. 3 provides an overview of our *TS-InvarNet* framework. *TS-InvarNet* consists of offline and online procedures. In the offline procedure, we adopt a similarity measure to conduct shaped-based clustering, which obtains several groups of KPIs. For univariate time series in each group, we apply a tempo-spatial model to model each pair of time series. Then we use a probability distribution-based judgment method as a threshold to automatically check whether the model will hold all the time. After that, we get an invariant network and we store the invariant network in the form of a compressed sparse row to reduce storage space. The model only needs several observations of each KPI to implement diagnosis in the online procedure. With the threshold obtained in offline procedures, the system will generate an invariant network for every observation. Compared with the network in the normal state of the system, we can obtain a series of evolution networks. We filter the false alarms by discarding the invariants with only a very small proportion of the corrupted ones and we can obtain accurate alarms. For the detected anomalies, we can utilize causal analysis to get the root cause KPI.

B. Invariant reduction

Since the amount of KPIs are tens of thousands, it takes much time to mine invariants in a pairwise way. Some invariants are repetitive and redundant, resulting in more time to observe the evolution of the invariant network. Considering that some time series may be characterized by similar time patterns, we adopt a shape-based clustering approach to classify KPIs. Then we can mine invariants in each cluster to compress the invariant network and reduce the time consumption.

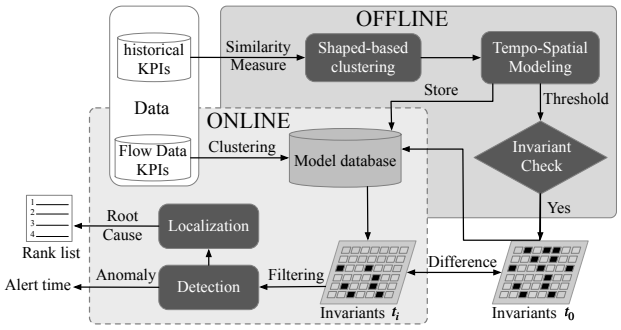


Fig. 3: The Overview of *TS-InvarNet*.

1) Shaped-based Similarity Measure: We adopt Shaped-based Distance (SBD) to measure the distance between each two KPIs [22]. For two time series $\vec{x} = (x_1, ..., x_m)$ and $\vec{y} = (y_1, ..., y_m)$, the similarity of them can be measured well with cross-correlation even if $\vec{x}$ and $\vec{y}$ are not properly aligned. Keeping $\vec{y}$ static, slide $\vec{x}$ over $\vec{y}$ to compute their inner product for each shift s of $\vec{x}$. The shift of $\vec{x}$ can be denoted as follows:

$$\vec{x}_{(s)} = \begin{cases} (\overbrace{0, \ldots, 0}^{|s|}, x_1, x_2, \ldots, x_{m-s}), & s \geq 0 \\ (x_{1-s}, \ldots, x_{m-1}, x_m, \underbrace{0, \ldots, 0}_{|s|}), & s < 0 \end{cases} . \quad (1)$$

Considering all possible shifts $s \in [-m, m]$, the inner-product $CC_s(\vec{x}, \vec{y})$ is defined as Eq.(2). The max value of $CC_s(\vec{x}, \vec{y})$ means that the similarity of $\vec{x}$ and $\vec{y}$ reaches the greatest at the optimal shift s. To normalize the cross-correlation, $NCC_c(\vec{x}, \vec{y})$ is defined as Eq.(3), limited in $[-1, 1]$. The value 1 represents two time series are perfectly similar.

$$CC_s(\vec{x}, \vec{y}) = \begin{cases} \sum_{i=1}^{m-s} x_i \cdot y_{s+i}, & s \geq 0 \\ \sum_{i=1}^{m+s} x_{i-s} \cdot y_i, & s < 0 \end{cases}, \quad (2)$$

$$NCC_c(\vec{x}, \vec{y}) = \frac{CC_s(\vec{x}, \vec{y})}{|\vec{x}| * |\vec{y}|}. \quad (3)$$

According to $NCC_c(\vec{x}, \vec{y})$, SBD is defined as Eq.(4). The value of SBD falls in the range of 0 to 2, where 0 indicates a perfect similarity and the smaller value means the higher shape similarity.

$$SBD(\vec{x}, \vec{y}) = 1 - NCC_c(\vec{x}, \vec{y}). \quad (4)$$

2) Clustering: To extract the most significant clusters from KPIs, an efficient and automatical clustering algorithm is in need. We use HDBSCAN as the base clustering algorithm due to its insensitivity to parameters and no need to predetermine the number of clusters [23]. More importantly, HDBSCAN

can automatically extract the most significant clusters by the simplified clustering hierarchy it generates.

Density-based methods, like DBSCAN [24], use core distance and min points to measure the density of a cluster, requiring a proper global density threshold to obtain meaningful clusters. The threshold is set too low causing too many objects to be considered as noise while too high only one cluster can be obtained. Besides, it is inaccurate to measure the density with a global threshold since the density of clusters is not always the same in a classification problem. HDBSCAN transforms the density space to spread apart objects with low density. It first uses an inexpensive and simple method, the distance from object x to the kth nearest neighbor, $N_k(x)$, to estimate density. Then it defines a new core distance and distance metric between objects as Eq.(5) and Eq.(6), respectively:

$$core_k(x) = d(x, N_k(x)), \tag{5}$$

$$d_{\text{mreach}-k}(a,b) = \max\{\text{core}_k(a), \text{core}_k(b), d(a,b)\}, \tag{6}$$

where $d(a,b)$ means the origin distance of a and b. After being transformed, the dense objects keep the same distance but sparse objects are pushed away from at least the core distance of any other objects. When the clusters have varying local densities, it is hard to obtain proper clusters with the global threshold. HDBSCAN builds the cluster hierarchy over various thresholds. The step can be optimized by Minimum Spanning Tree(MST), which produces a clustering tree containing all partitions in a hierarchical way.

The large and complicated cluster hierarchy can be condensed down into a smaller tree with a little more data attached to each node using minimum cluster size [25]. Then considering the stability of the cluster as a way to determine if it is prominent, define $\lambda = \frac{1}{distance}$. Specifically, $\lambda_{min}(C_i)$ and $\lambda_{max}(C_i)$ are the value when the cluster C_i is generated and when the cluster is split into smaller clusters, respectively. Define $\lambda_{max}(x_j, C_i)$ as the value when the object x_j no longer belongs to the cluster C_i. So the stability of each cluster can be computed as Eq.(7). Traversing the tree bottom-up, update the total stability $\hat{S}_{C_i}$ based on Eq.(8),

$$S_{(C_i)} = \sum_{x_j \in C_i} (\lambda_{max}(x_j, C_i) - \lambda_{min}(C_i)), \tag{7}$$

$$\hat{S}_{(C_i)} = \max\{S_{(C_i)}, \sum_{C_{ij} \in C_{ichildren}} \hat{S}_{(C_{ij})}\}, \tag{8}$$

where $C_{ichildren}$ is the children of C_i. To select C_i or C_i's subtrees depends on the result of updated total stabilities. Concretely, the cluster's stability will be updated as the sum of the children's stabilities if the latter is greater than the former. If the cluster's stability is greater, it will instead be declared as the selected cluster ignoring all its descendants. Once the root node is reached, the currently selected set of clusters is returned as the result. Fig. 4 shows the results of clustering on a dataset of KPIs. We can observe that there are similar variations in the same cluster while the shapes of KPIs in noises are different. We search for invariant relationships in the same cluster. Then select a representative KPI from different clusters to further build invariants with the noise KPIs.

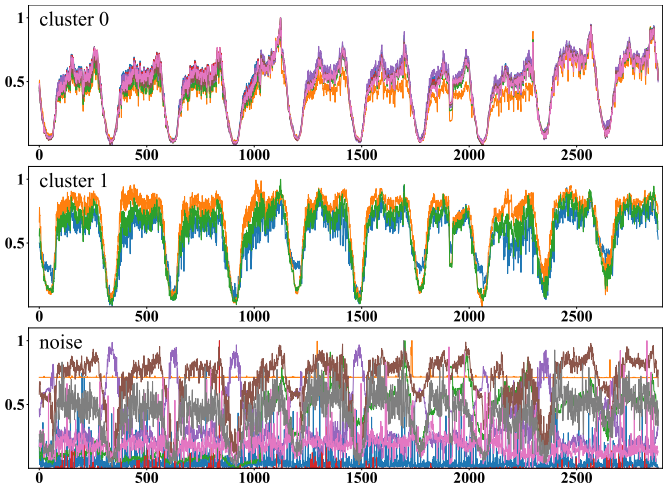


Fig. 4: The results of shaped-based clustering on KPIs collected from an Internet company.

C. Invariant mining

1) Invariant Model: To acquire accurate invariant relationships, the model chosen to evaluate whether two KPIs hold the invariant relationship is essential. Different from the work [26]–[29], *TS-InvarNet* constructs invariants using SARIMAX (Seasonal AutoRegressive Integrated Moving Average with eXogenous regressors model) rather than autoregressive model with exogenous input. SARIMAX can effectively mine the Tempo-Spatial relationship between KPIs with multiple periods and complicated trends. It is a state-space model that not only reflects the internal state of the system but also reveals the relationship between the internal state of the system and external input variables.

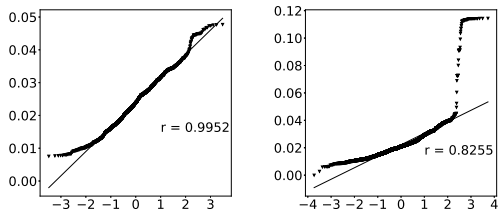
SARIMAX model consists of 5 parts: the seasonal term (S), the autoregressive term (AR), the integrated term (I), the moving-average term (MA), and the eXogenous term (X). The AR term means the value at one time is considered as a weighted sum of past values while the MA term is a weighted sum of past residuals. The S term and the I term are used to differentiate the time series for stationary. For the natural interpretations of the estimated parameters [30], the model is specified as follows:

$$y_t = \beta_t x_t + u_t, \tag{9}$$

$$\phi_p(L)\tilde{\phi}_P(L^s)\Delta^d\Delta_s^D u_t = A(t) + \theta_q(L)\tilde{\theta}_Q(L^s)\epsilon_t. \tag{10}$$

The Eq. (9) is a linear regression and the Eq.(10) simply describes the SARIMAX process followed by the error component. In the Eq.(10), the definition of the parameters is as follows:

- L is an operator which finds the past n observations of the current moment. For example, $y_{t-2} = L^2 y_t$.
- $\phi_p(L)$ and $\tilde{\phi}_P(L^s)$ represent the non-seasonal and the seasonal autoregressive lag polynomial, respectively.



(a) Invariant (b) Non-invariant

Fig. 5: The Q-Q plot of fitness scores of two pair of KPIs.

- $\Delta^d \Delta_s^D u_t$ is the result of differencing the time series d times and seasonally defferencing D times.
- $A(t)$ represents the trend polynomial.
- $\theta_q(L)$ and $\tilde{\theta}_Q(L^s)$ denote the non-seasonal and the seasonal moving average lag polynomial, respectively.
- ϵ_t represents the residuals between the observed value and estimation for the current period.

The parameters are solved mainly by maximum likelihood estimate (MLE) [31]. For two time series $\vec{x} = (x_1, ..., x_t)$ and $\vec{y} = (y_1, ..., y_t)$, We can obtain parameters Φ as follows:

$$\Phi = (\beta_t, \phi_1, \ldots, \phi_p, \tilde{\phi}_1, \ldots \tilde{\phi}_q, \theta_1, \ldots, \theta_P, \tilde{\theta}_1, ..\tilde{\theta}_Q). \quad (11)$$

2) Invariants Extracting: Based on parameter Φ and new observations, we can calculate the estimation $\hat{y}(\Phi|t)$. To evaluate how well the learned model fits the observations, we use RMSE (Root of Mean Square Error) to calculate a fitness score. Here we use a sliding window technique and set the length of a window as m. When the window slide w observations, the $RMSE$ will be calculated once as Eq.(12) shows:

$$F_{score} = RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}. \quad (12)$$

A static threshold does not work well to determine whether an invariant relationship exists since the scales of different KPIs differ and the fluctuation of origin data would have an influence on F_{score}. We employ the Quantile-Quantile (Q-Q) plot of fitness scores to assess the appropriate fit and find the optimal distribution is normal distribution. So we assess an invariant relationship by checking whether the distribution of the fitness scores follows the normal distribution. In more detail, if the points on the Q-Q plot are approximately in the vicinity of a straight line, the scores conform to the normal distribution and thus there exists an invariant. The slope of the straight line is the standard deviation and the intercept is the mean. Fig. 5 shows the Q-Q plot of the fitness scores of two pair of KPIs. We can determine whether they are an invariant or not with Q-Q plot.

D. Anomaly Detection

The combination of a large number of invariants could effectively characterize large, dynamic, and complex distributed systems. When new observations arrive, we compute an anomaly score for each pair of invariants. If new scores belong to the extreme value of distribution we identified in the offline process, we determine that the invariant is violated. However, one broken invariant may not imply an occurring anomaly because of noise or some sudden fluctuation of time series. So it is quite straightforward to consider observing from the whole invariant network. Only when a certain number of invariants are broken can the whole system be considered anomalous. The Eq.(13) illustrates how to determine an anomaly, where $Invar_{t_0}$ is the invariant network obtained from offline process, $Invar_{t_1}$ is the invariant network at time t_1 in the online process, sum is the number of invariants and τ is the preset threshold.

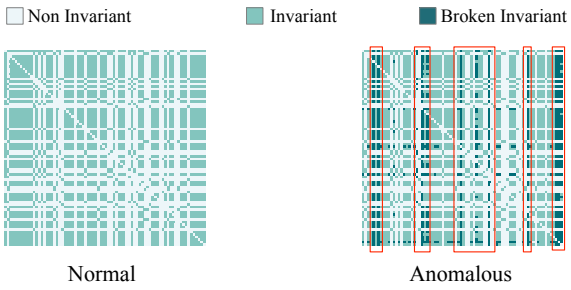


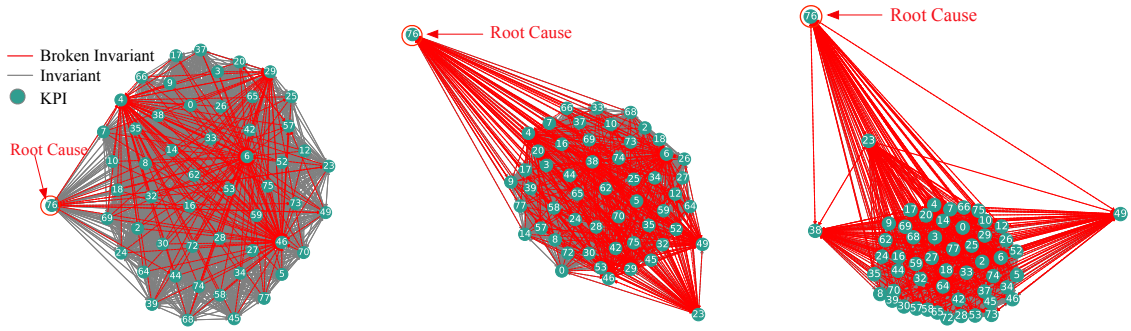
Fig. 6: The invariant network in different states of a system.

$$\frac{sum(Invar_{t_1} - Invar_{t_0})}{sum(Invar_{t_0})} > \tau. \quad (13)$$

E. Anomaly Interpretation and Localization

It is beneficial for problem localization to interpret a detected anomaly by finding a group of anomalous KPIs. Some KPIs may be affected by failures and perform abnormally, while some will not. The fault may break most invariants of an anomalous KPI. So we interpret the anomaly based on the proportion of broken invariants to the total invariants. Fig. 6 shows an invariant network in different states of a distributed system, where the dark green in the anomalous graph represents broken invariants. Most broken invariants present a continuous trend while some appear sporadically. The latter may be false alarms, which should be discarded to reduce the complexity of subsequent analysis. Specifically, we calculate the percentage of broken invariant of each KPI, ϵ_i and rank ϵ_i. Consider a higher ranking of ϵ_i as a more anomalous KPI and the ϵ_i below the average as false alarms.

The solution to localize the root cause KPI cannot be simply converted to find the anomalous KPI appearing earliest in the entity. Fig. 7 shows the evolution of an invariant network after a failure occurred. From Fig. 7(a) we can observe the red edges around nodes 4, 6, 29, and 46 are more than others, indicating they are the first to show the most anomalous behaviors. However, as the anomaly spreads, the broken invariants of nodes 76, 38, 23, and 49 exceed others, implying they have a strong association with the failure. In fact, they are the KPIs monitoring the root cause service. The above observation validates our claim and we can further explain the observations. For one thing, the duration of some



(a) Failure occurred initially (b) Failure has occurred for a while (c) Failure impact is almost over

Fig. 7: **The evolution of an invariant network when a failure occurred**. The broken invariants of node 76 and 23 grows with time indicating there are strong correlation between them and the failure. In fact, Node 76 represents CPU of a service's instance injected with CPU fault and 38 represents its workload. Node 23 and 49 represent workload of another two instances of the service. The rapidly propagating failure makes it difficult to distinguish the true root cause KPI.

failures is relatively short but failures can rapidly propagate. Since KPIs are sampled at a coarse granularity, it seems like the root cause KPI and the affected KPIs behave abnormal simultaneously. For another, as there is possibility of false alarms, it is difficult to distinguish between a root cause KPI and a false alarm. Another significant conclusion that can be drawn from the above observations is that the invariants of the root cause service's KPIs are indeed broken most.

We can determine the probable root cause KPIs based on the number of broken invariants. But it is difficult to further determine the root cause KPI because there are strong correlations between the probable root cause KPIs. In Fig. 7, the nodes 38 and 76 represent the KPIs monitoring workload and CPU of the root cause service's instance, respectively. While the nodes 23 and 49 represent the KPIs monitoring workload of another two instances of the root cause service. To identify further which KPI is more likely to be the root cause, we use Granger causality test [32], a hypothesis testing statistical method, to capture causality among anomalous KPIs.

Granger Causality test is informally defined as follows. If with the history of Y, the error δ_{xy} in predicting X is less than the error δ_x in predicting X only with the history X, we can say the KPI Y is Granger-causing the KPI X. We check whether the test is valid via F-test [33]. If the p-value is below a critical value (i.e., 0.05), the null hypothesis (i.e., Y does not granger-cause X) is rejected [34]. After applying the Granger Causality tests to anomalous KPIs, we can obtain a causality graph. In the causality graph, nodes denote KPIs and edges represent causal relations among KPIs. We rank the nodes according to outdegree centrality. The probability that the node is a root cause KPI depends on the outdegree of the node. After the step, the root cause KPI 76 in Fig. 7 can be determined.

IV. Experiment

A. Experiment Setup

1) Dataset: We conduct experiments on three real-world industrial datasets to illustrate the performance of anomaly detection. Since few open-source datasets cover scenarios where the anomaly propagates widely with labels of root cause KPIs, we use one open-source microservice system to validate the performance of anomaly interpretation and localization.

TABLE I: The detailed infomation of Industrial Datasets

Dataset	Services	KPIs	Training Days	Testing Days	Anomaly Ratio(%)
Dataset1	30	19	20	25	5.25
Dataset2	30	19	20	25	20.26
Dataset3	13	11	5	2	0.55

Industrial Datasets: Two datasets, namely Dataset1 and Dataset2, are open-source datasets [1], which are collected from 30 online service systems. They are sampled once every five minutes and collected for 5 weeks. Dataset3 is collected from online cloud servers in a game business of a big company, which are selected from 13 cloud servers that suffered from failures. A total of 11 KPIs are monitored, such as CPU usage, disk usage, and memory usage. The KPIs are collected at the interval of one minute for 7 days. We did not use the well-known dataset SMD from [4] since it is of low quality as [7], [8], [20] claimed. Its labeled anomalies are Gaussian outliers of high acuteness, which can be easily detected [20]. The detailed information of each dataset is listed in Table I.

Hipster-shop microservice system: Hipster-shop is an open-source microservice system, which is a widely-used benchmark designed to aid demonstration and testing of microservices technologies [35], [36]. It contains 10 microservices, such as *product*, *cart* and *currency*, where users can browse products, add them to the cart and purchase them. Equipped with a workload generator, Hipster-shop simulates concurrent users of that application. The service of *front end* and *product* receive more requests while *checkout* and *payment* fewer, which is conformed with the workload law.

To mimic anomaly propagation issues, Hipster-Shop is

[1]dataset, https://github.com/NetManAIOps/JumpStarter

injected with two types of faults. To simulate the latency problem and CPU exhaustion problem, we delay service instance's network packets to stimulate Network Jam and consume CPU heavily to stimulate CPU exhaustion by Chaosblade[2], a chaos engineering tool. In our experiments, each fault is injected for 5 minutes. The interval between injection operations is nearly 60 minutes to reduce the mutual influence of different injection operations. We injected 32 faults, including 12 CPU faults and 20 latency faults to Hipster-Shop.

2) Benchmark Model: We compare *TS-InvarNet* with two state-of-the-art models based on deep learning, Omnianomaly [4] and USAD [5]. The former model adopts a stochastic recurrent neural network to deal with temporal dependence and learn representations of input data. The latter model adopts encoder-decoder architecture within an adversarial training framework. We also select a novel anomaly detection model, RANSynCoders [7], which uses an architecture of multiple encoders-decoders to infer anomalies. To compare with methods not based on deep learning, we select JumpStarter [6], a multivariate time series anomaly detection approach based on Compressed Sensing (CS). As for the problem localization part, we are not going to compare *TS-InvarNet* with other approaches [37]–[39] because they all require system topology or other information to generate a dependency graph while *TS-InvarNet* only uses KPIs.

3) Evaluation Metrics and Threshold Selection: For each observation in multiple time series, an anomaly detection approach generates a result indicating whether an anomaly has occurred. We use Precision (P), Recall (R), and F1 score (F1) to evaluate anomaly detection performance of all models:

$$P = \frac{TP}{TP + FP}, R = \frac{TP}{TP + FN}, F1 = \frac{2 \cdot P \cdot R}{P + R}. \quad (14)$$

where TP is the True Positives, FP is the False Positives and FN is the False Negatives.

Practically, an anomaly occurs at one timestamp but its impact of it will last for some time. So it is more important to identify anomalies in a contiguous segment than point-wise anomalies. Thus we adopt a point-adjust approach [4], which is widely used to evaluate the anomaly detection [4]–[6], [8], [40], [41]. In this approach, if any observation of an anomalous segment in the ground truth is detected, we consider the whole anomalous segment as detected correctly.

4) Implementation: The *TS-InvarNet* is implemented using Python 3.7. The parameters of S, AR, I, MA in SARIMAX model are set as 1440, 3, 1, and 3, respectively. These parameters can be tuned in different services. The sliding window size $w = 50$. We conduct our experiments on a server with Intel CPU @ 2.10GHz and 128GB memory.

B. Overall Performance

1) Anomaly Detection: To demonstrate the overall performance of *TS-InvarNet*, we compare it with benchmark models. To distinguish anomaly detection methods' performance using different threshold selection methods, we first try various thresholds on the detection results and select the posterior optimal threshold. This approach, called best F1 score, can help to determine a preset threshold and provide feedback for deploying anomaly detection method, which has been widely adopted as an ideal threshold selection [4]–[8]. Table II lists the obtained results with best F1 score on different datasets. We note that *TS-InvarNet* yields superior performance for most datasets. The exception is for Dataset2, where JumpStarter performs 1.64% better, which we will further discuss below. On the averaged performance over all datasets, *TS-InvarNet* is the best performing approach exceeding by 7% over the state-of-the-art [4], [5].

As we mentioned before, the best F1 score is just an ideal threshold selection, only theoretically serving to help. But not all the baseline models provide a mechanism to select thresholds in practice. While considering a good anomaly detection method does not overly rely on a posteriori threshold selection, we use a simple widely used threshold, 3-sigma rule with a sliding window [6], to evaluate the actual performance of all models in practice. Table II lists the average best F1 scores, F1 scores, Precision and Recall of *TS-InvarNet*, and baseline models on different datasets. As expected, the results show that there is a discrepancy between the best F1 scores and dynamic 3-sigma scores. We note a significant drop in performance of OmniAnomaly [4] by using dynamic 3-sigma scores. We try the automatic threshold selection method POT, adopted by OmniAnomaly. It still underperforms and we conjecture the difficulty in selecting an optimal threshold leads to its different performances between with and without best F1. And as [7] said, OmniAnomaly's superiority in the SMD dataset does not carry over to other datasets. As for high recall of OmniAnomaly with POT, we do not think it is worthwhile because we find POT tends to achieve high recall at the price of quite low precision. Most F1 scores in Dataset3 are lower than 30% while some even are 5%. Practically, it is of little help for operators to analyze failures. In fact, we can adjust threshold selections to adapt to the goals of high recall or F1 score in different scenes. Here we primarily consider the F1 score.

All methods, especially Jumpstarter, present the lowest performance on Dataset3 without the best F1. The anomaly ratio of Dataset3, 0.55%, is far lower than 5.25% and 20.26%. The sampling in Jumpstarter completes data compression while weakening the features of anomaly, leading to bad performance in datasets with a low anomaly ratio. Besides, Jumpstarter obtains a worse F1 score without optimal threshold on Dataset3 while TS-InvarNet can still perform well. Another reason for the poor results of deep learning methods on Dataset3 is the short training time. The training time of Dataset3 is 5 days, 1/4 of the others while deep learning methods require a long period of training time, indicating it is unsuitable for newly updated systems. RANSynCoders shows relatively poor results in all datasets. We attribute it to two reasons. Firstly, it uses a network to learn the phase shifts across signals and generate a synchronized representation of the raw time series. However, the phase shifts among

[2]Chaosblade, https://github.com/chaosblade-io/chaosblade

TABLE II: Performance Comparisons of *TS-InvarNet* and Baselines on Different Datasets

Method	Dataset1				Dataset2				Dataset3				Average			
	F1-best	F1	P	R	F1-best	F1	P	R	F1-best	F1	P	R	F1-best	F1	P	R
TS-InvarNet	**96.96**	**75.35**	70.30	84.19	92.59	**91.71**	93.00	94.81	**94.59**	**61.81**	59.83	67.34	**94.71**	**76.29**	74.38	82.11
JumpStarter	86.85	71.92	68.04	83.98	**94.23**	82.81	91.79	78.26	74.40	28.76	21.25	56.79	85.16	61.16	60.36	73.01
OmniAnomaly	69.59	11.57	60.46	14.34	88.43	35.34	49.28	22.94	88.62	9.12	46.51	10.40	82.21	18.68	52.08	15.89
OmniAnomaly(POT)	-	31.38	27.51	**88.61**	-	60.99	53.01	**99.95**	-	44.1	34.45	**100**[1]	-	45.49	38.32	**96.19**
RANSynCoders	56.00	48.84	57.91	49.77	85.00	59.57	73.02	53.82	61.00	54.50	52.73	64.52	67.33	54.30	61.22	56.04
USAD	92.00	70.26	**93.09**	78.48	92.00	90.32	81.40	85.28	79.00	60.45	**86.91**	67.83	87.67	73.68	**87.13**	77.20

[1] High recall can be obtained easily by setting low threshold. Low precision indicates more false alarms, hindering subsequent troubleshooting.

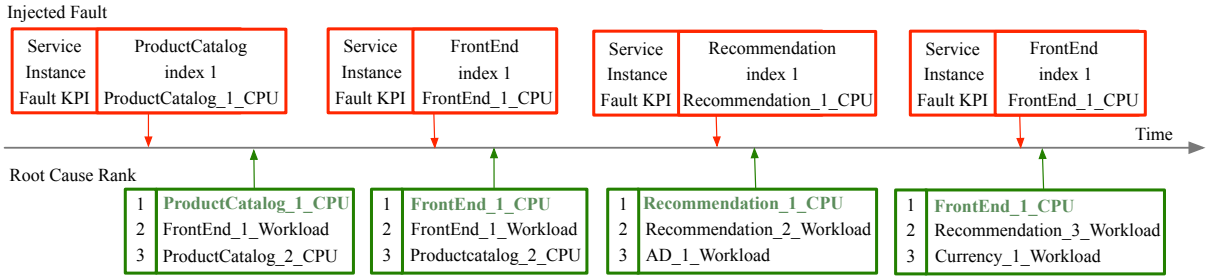


Fig. 8: Examples of fault injections and the root cause KPIs ranks of candidates on Hipster-Shop.

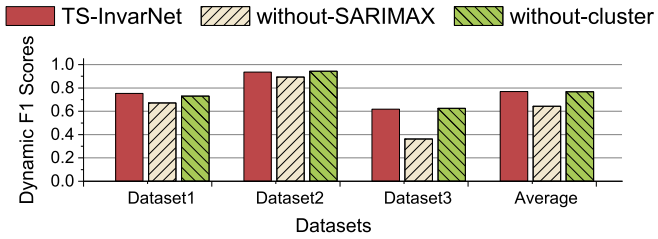


Fig. 9: Average $F1$ for TS-InvaNet and its variants.

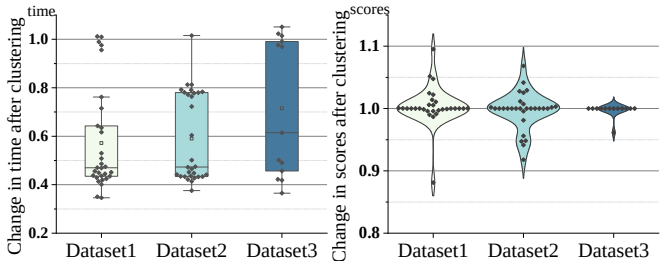


Fig. 10: Change in time and scores after clustering.

different signals may be the same due to similar business behaviors. Secondly, the spectral analysis may lose part of the temporal information due to extracting dominant frequencies. Meanwhile, it may drop spatial dependence to minimize the reconstruction losses through feeding random subsets of synchronous time series to the autoencoders. The lack of temporal and spatial information leads to its bad performance. Overall, USAD performs good performances on all datasets. But *TS-InvarNet* still outperforms it by 7% with best F1 scores and 3% without it. The relatively low best F1 score of USAD indicates its limited potential to achieve very high accuracy in anomaly detection.

2) Anomaly Interpretation and Localization: We evaluate the problem localization on hipster-shop, which provided a ground-truth of root cause KPIs. We detected 31 faults in 32 and the root cause KPIs all are interpreted as the most anomalous KPIs. The remaining one shows a slight abnormality and it does not propagate to other instances. We further analyze the root cause and the results show *TS-InvarNet* can locate 75% root cause KPIs within the top 5 candidates. Fig. 8 shows some examples of fault injections and the root cause KPIs ranks output by *TS-InvarNet*.

We take one of the injected faults as an example to illustrate our approach in detail. A CPU fault was injected into one instance of a product catalog service. After 2 observations, we detected the number of broken invariants increasing from 0 to 511, accounting for more than ten percent of the total invariants. We ranked the KPIs by the number of broken invariants and filtered the KPIs whose broken invariants were far below the average. The KPIs in the top ranking are interpreted as the most anomalous KPIs. We found those KPIs mainly monitor the workload and CPU of ProductCatalog service and its downstream services, such as Recommendation service or FrontEnd service. With outdegree centrality in the casual graph, we ranked the anomalous KPIs and the first in the ranking is the KPI monitoring the CPU of the instance we injected a CPU fault.

C. Ablation Study

In this section, we conduct an ablation study using several variants of *TS-InvarNet* to further demonstrate the effectiveness of the designs. As shown in Fig. 9, *TS-InvarNet* outperforms *without-SARIMAX* on all datasets, which demonstrates the capabilities of SARIMAX model to learn temporal

TABLE III: Average storage of different methods on datasets

	TS-InvarNet	OmniAnomaly	RanSynCoder	USAD
Storage	292K	4.9G	390K	36M

TABLE IV: Maximum computing time of one observation on different datasets

	Dataset1	Dataset2	Dataset3
time(s)	1.92	1.92	0.2253

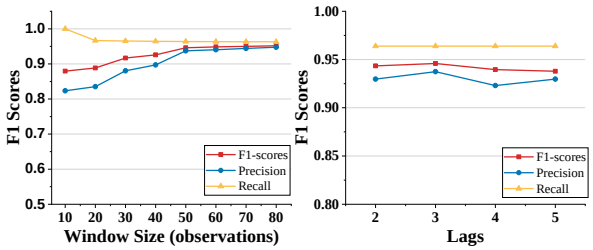


Fig. 11: F1-score of *TS-InvarNet* with different parameters.

and spatial relationships from complicated multivariate time series. Moreover, *TS-InvarNet* has a similar performance with *without-cluster*. It indicates the spatial information discarded by shaped-based clustering may be redundant and clustering does not have an influence on the performance of *TS-InvarNet*.

Fig. 10 presents the change in time and scores after shaped-based clustering on each entity of all datasets. In the left box plot, most points are around 0.3-0.4 and a small amount around 1. This validates clustering significantly faster *TS-InvarNet*. In the right violin Plot, most points are around 1. It indicates clustering has little influence on the performance in anomaly detection, which further validates our conclusion that shaped-based clustering not only does not weaken the performance of anomaly detection but also remarkably reduces the running time.

D. Feasibility Study

The implementation of *TS-InvarNet* can be divided into two phases, namely *Offline* and *Online*. The *Offline* step stores invariant models of an entity, invariant check thresholds, which are used for online anomaly detection and problem localization. Compared with state-of-the-art methods [4], [5], [7], *TS-InvarNet* can perform with less storage as Table III shows. Though the storage of RanSynCoder is close to *TS-InvarNet*, its overall performance is much worse than *TS-InvarNet*. We need to emphasize that despite the costs of hard disks and storage capacity consumed seems to be of little importance, there are some specific scenarios where storage space of the model becomes particularly important such as Edge Computing and IoT (Internet-of-Things).

In the *Online* step, as a new observation x_t arrives, we can obtain anomaly scores of the entity by the models and utilize the invariant check threshold to count the number of broken invariants. If the number is higher than a preset threshold, then we declare the observation x_t an anomaly. For a detected anomaly, we can apply anomaly interpretation to know specific anomalous KPIs and causal analysis to find the root cause KPI. If the faults are not repaired in time, it will cause enormous economic losses and serious consequences. So it is more crucial to know the online computing time. Table IV shows the maximum computing time of anomaly detection of one observation on different datasets, which illustrates the ability of *TS-InvarNet* to respond quickly.

In Fig. 11, we show *TS-InvarNet*'s sensitivity to some key parameters that can have an impact on the performance of *TS-InvarNet*. As the window size increases from 10 to 80 observations, each evaluation score gradually stabilizes. Before the window size reaches 50 observations, the F1 score increases and after that it becomes stable. Thus the window size is preset as 50. In the meantime, we also note that when the window size reaches 30, the F1 score is more than 0.9. As for the order, we set the non-seasonal AR order and MA order to be equal and resize them at the same time. We find the order has little influence on the performance. When we increase the order from 2 to 4, each evaluation score has slight variations. When the order is 3, the F1 score is slightly higher than the others. Therefore, we set the order to 3.

V. Related Work

A. Anomaly detection

Anomaly detection is a complex task, which has been an active topic. Traditional anomaly detection approaches mainly includes SVM [42], KNN [43], HMMs [44], Kalman filters [45], which are unable to handle with complex dynamic time series. The shapelet learning-based approaches learn features that are representative of the normal class from a training set [46], which shows good performance when the subsequences indeed contain relevant information. However, it is limited in scenarios where statistics or spectral features over the whole time series determine the class. Traditional invariant-based approach for anomaly detection uses ARX model to mine invariant relationship [26], which is time-consuming and not applicable for time series with periodicity and trend. Besides, it is not robust to check invariants with a fixed threshold. We adopt shaped-based clustering to reduce the complexity of mining invariants and use SARIMAX to model accurately invariants in complicated KPIs. A probability distribution-based helps to automatically check invariants. JumpStarter [6] is a jump-starting anomaly detection approach based on Compressed Sensing (CS) with a short initialization time.

Recently, deep learning approaches attract widespread interest. Donut [17] proposed a univariate anomaly detection method based on VAE for seasonal KPIs with local variations. Microsoft combines SR and CNN models to detect anomalies in univariate time series [40], proving the possibility of using visual saliency in anomaly detection. However, these univariate anomaly detection methods only model temporal dependency but cannot consider the overall system's status, more prone to false alarms. Omnianomaly [4] adopts a stochastic recurrent neural network to deal with temporal dependence and

learn robust multivariate time series representations. LSTM-VAE [47] combined VAE and LSTM by replacing the feed-forward network in a VAE with LSTM. Combining the advantages of autoencoders and adversarial training, USAD [5] adopts an encoder-decoder architecture within an adversarial training framework. RANSynCoders [7] uses an architecture of multiple encoders-decoders to infer and localize anomalies. The methods can well represent the system's overall status but lose insights into the local correlations between KPIs, leading to poor performance in detecting subtle anomalies and interpretability.

B. Root Cause Analysis

Root cause analysis based on KPIs usually builds a dependency graph with different methods. After monitoring KPIs to detect anomalies, these works conduct problem localization with the combination of KPIs and the dependency graph. CauseInfer constructs a causality graph and infer the root causes of performance problems along the causal paths in the graph with statistical methods [48]. Microscope [37] captures the real service dependency through capturing and parsing the network-related system calls and locates the root cause by comparing the similarity between SLO KPIs and the abnormal service. MicroRCA [38] extracts an anomalous subgraph from an attribute graph including service and KPIs. Automap [39] generates a causality graph between services and selecting the proper KPIs, and identifies the root cause by random walking. These works all require system topology or other extra information to build a dependency graph.

VI. Conclusion

This paper designs and implements *TS-InvarNet*, an interpretable end-to-end anomaly detection and localization framework based on tempo-spatio KPI invariants. Through shape-based clustering, *TS-InvarNet* can generate an invariant network quickly. Then *TS-InvarNet* keeps track of the evolution of the invariant network to diagnose the system. When an anomaly is detected, *TS-InvarNet* can interpret the anomaly and conduct causal analysis to localize the root cause KPIs. The experimental evaluations on three real-world industrial datasets show that *TS-InvarNet* can identify the problem accurately, which outperforms some state-of-the-art approaches. The experiment on one open-source microservice system shows that *TS-InvarNet* has the ability to localize the root cause KPI. Moreover, *TS-InvarNet* is lightweight enough to be deployed in large-scale systems.

Acknowledgment

The research is supported by the National Key Research and Development Program of China (2019YFB1804002), the Key-Area Research and Development Program of Guangdong Province (No. 2020B010165002), the National Natural Science Foundation of China (No. U1811462), the Basic and Applied Basic Research of Guangzhou (No. 202002030328), and the Natural Science Foundation of Guangdong Province (No. 2019A1515012229). The corresponding author is Pengfei Chen.

References

[1] H. Zhou, M. Chen, and et al., "Overload control for scaling wechat microservices," in *SoCC*, 2018, pp. 149–161.

[2] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," ser. ESEC/FSE 2019. Association for Computing Machinery, 2019, p. 683–694.

[3] R. Ascierto, "Too big to fail? facebook's global outage," https://www.datacenterdynamics.com/en/opinions/too-big-to-fail-facebooks-global-outage/, accessed Jan 6, 2022.

[4] Y. Su, Y. Zhao, and et al., "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *SIGKDD*, 2019, pp. 2828–2837.

[5] J. Audibert, P. Michiardi, and et al., "Usad: Unsupervised anomaly detection on multivariate time series," in *SIGKDD*, 2020, pp. 3395–3404.

[6] M. Ma, S. Zhang, and et al., "Jump-starting multivariate time series anomaly detection for online service systems," in *ATC*, 2021, pp. 413–426.

[7] A. Abdulaal, Z. Liu, and et al., "Practical approach to asynchronous multivariate time series anomaly detection and localization," in *SIGKDD*, 2021, pp. 2485–2494.

[8] Z. Li, Y. Zhao, and et al., "Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding," in *SIGKDD*, 2021, pp. 3220–3230.

[9] J. Gao, X. Song, and et al., "Robusttad: Robust time series anomaly detection via decomposition and convolutional neural networks," *arXiv preprint arXiv:2002.09545*, 2020.

[10] Q. Wen, J. Gao, and et al., "Robusttrend: A huber loss with a combined first and second order difference regularization for time series trend filtering," in *IJCAI*, 2019, pp. 3856–3862.

[11] Q. Wen, K. He, and et al., "Robustperiod: Robust time-frequency mining for multiple periodicity detection," in *SIGMOD*. ACM, 2021, pp. 2328–2337.

[12] R. Taft, N. El-Sayed, and et al., "P-store: An elastic database system with predictive provisioning," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 205–219.

[13] S. Hawkins, H. He, and et al., "Outlier detection using replicator neural networks," in *International Conference on Data Warehousing and Knowledge Discovery*. Springer, 2002, pp. 170–180.

[14] L. Dai, T. Lin, and et al., "Sdfvae: Static and dynamic factorized vae for anomaly detection of multivariate cdn kpis," ser. WWW '21. Association for Computing Machinery, 2021, p. 3076–3086.

[15] P. Malhotra, A. Ramakrishnan, and et al., "Lstm-based encoder-decoder for multi-sensor anomaly detection," *arXiv preprint arXiv:1607.00148*, 2016.

[16] D. Liu, Y. Zhao, and et al., "Opprentice: Towards practical and automatic anomaly detection through machine learning," in *IMC*, 2015, pp. 211–224.

[17] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng *et al.*, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *Proceedings of the 2018 world wide web conference*, 2018, pp. 187–196.

[18] F. Di Mattia, P. Galeone, M. De Simoni, and E. Ghelfi, "A survey on gans for anomaly detection," *arXiv preprint arXiv:1906.11632*, 2019.

[19] D. Li, D. Chen, B. Jin, L. Shi, J. Goh, and S.-K. Ng, "Mad-gan: Multivariate anomaly detection for time series data with generative adversarial networks," in *International Conference on Artificial Neural Networks*. Springer, 2019, pp. 703–716.

[20] R. Wu and E. Keogh, "Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress," *TKDE*, 2021.

[21] C. Zhang, D. Song, and et al., "A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 1409–1416.

[22] J. Paparrizos and L. Gravano, "k-shape: Efficient and accurate clustering of time series," in *SIGMOD*, 2015, pp. 1855–1870.

[23] R. J. Campello, D. Moulavi, and et al., "Density-based clustering based on hierarchical density estimates," in *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 2013, pp. 160–172.

[24] M. Ester, H.-P. Kriegel, and et al., "A density-based algorithm for discovering clusters in large spatial databases with noise." in *KDD*, vol. 96, no. 34, 1996, pp. 226–231.

[25] L. McInnes, J. Healy, and et al., "hdbscan: Hierarchical density based clustering," *Journal of Open Source Software*, vol. 2, no. 11, p. 205, 2017.

[26] G. Jiang, H. Chen, and et al., "Discovering likely invariants of distributed transaction systems for autonomic system management," in *2006 IEEE International Conference on Autonomic Computing*, 2006, pp. 199–208.

[27] Jiang, Guofei and Chen, Haifeng and et al., "Efficient and scalable algorithms for inferring likely invariants in distributed systems," *TKDE*, vol. 19, no. 11, pp. 1508–1523, 2007.

[28] W. Cheng, K. Zhang, and et al., "Ranking causal anomalies via temporal and dynamical analysis on vanishing correlations," in *SIGKDD*, 2016, pp. 805–814.

[29] L. Aniello, C. Ciccotelli, and et al., "Automatic invariant selection for online anomaly detection," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2016, pp. 172–183.

[30] S. Seabold and J. Perktold, "statsmodels: Econometric and statistical modeling with python," in *9th Python in Science Conference*, 2010.

[31] I. J. Myung, "Tutorial on maximum likelihood estimation," *Journal of mathematical Psychology*, vol. 47, no. 1, pp. 90–100, 2003.

[32] C. W. Granger, "Investigating causal relations by econometric models and cross-spectral methods," *Econometrica: journal of the Econometric Society*, pp. 424–438, 1969.

[33] M. Tiku, "Tables of the power of the f-test," *Journal of the American Statistical Association*, vol. 62, no. 318, pp. 525–539, 1967.

[34] J. Thalheim, A. Rodrigues, and et al., "Sieve: Actionable insights from monitored metrics in distributed systems," in *Middleware*, 2017, pp. 14–27.

[35] G. Yu, P. Chen, and et al., "Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments," in *WWW*, 2021, pp. 3087–3098.

[36] Yu, Guangba and Chen, Pengfei and et al., "Microscaler: Automatic scaling for microservices with an online learning approach," in *ICWS*. IEEE, 2019, pp. 68–75.

[37] J. Lin, P. Chen, and et al., "Microscope: Pinpoint performance issues with causal graphs in micro-service environments," in *ICSOC*. Springer, 2018, pp. 3–20.

[38] L. Wu, J. Tordsson, and et al., "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.

[39] M. Ma, J. Xu, and et al., "Automap: Diagnose your microservice-based web applications automatically," in *Proceedings of The Web Conference 2020*, 2020, pp. 246–258.

[40] H. Ren, B. Xu, and et al., "Time-series anomaly detection service at microsoft," in *SIGKDD*, 2019, pp. 3009–3017.

[41] L. Shen, Z. Li, and et al., "Timeseries anomaly detection using temporal hierarchical one-class network," *Advances in Neural Information Processing Systems*, vol. 33, pp. 13 016–13 026, 2020.

[42] A. Rodriguez, D. Bourne, and et al., "Failure detection in assembly: Force signature analysis," in *2010 IEEE International Conference on Automation Science and Engineering*. IEEE, 2010, pp. 210–215.

[43] S. Ando, T. Thanomphongphan, and et al., "Ace: anomaly clustering ensemble for multi-perspective anomaly detection in robot behaviors," in *ICDM*. SIAM, 2011, pp. 1–12.

[44] D. Park, Z. Erickson, and et al., "Multimodal execution monitoring for anomaly detection during robot manipulation," in *ICRA*. IEEE, 2016, pp. 407–414.

[45] J.-i. Furukawa, T. Noda, and et al., "Estimating joint movements from observed emg signals with multiple electrodes under sensor failure situations toward safe assistive robot control," in *ICRA*. IEEE, 2015, pp. 4985–4991.

[46] L. Beggel, B. X. Kausler, M. Schiegg, M. Pfeiffer, and B. Bischl, "Time series anomaly detection based on shapelet learning," *Computational Statistics*, vol. 34, no. 3, pp. 945–976, 2019.

[47] D. Park, Y. Hoshi, and et al., "A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1544–1551, 2018.

[48] P. Chen, Y. Qi, P. Zheng, and D. Hou, "Causeinfer: Automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 1887–1895.