Jump-Starting Multivariate Time Series Anomaly Detection for Online Service Systems

Minghua Ma
Tsinghua University, BNRist

Shenglin Zhang *
Nankai University

Junjie Chen
Tianjin University

Jun Xu
Georgia Tech

Haozhe Li, Yongliang Lin
Nankai University

Xiaohui Nie
Tsinghua University, BNRist

Bo Zhou, Yong Wang
CNCERT/CC

Dan Pei
Tsinghua University, BNRist

Abstract

With the booming of online service systems, anomaly detection on multivariate time series, such as a combination of CPU utilization, average response time, and requests per second, is important for system reliability. Although a collection of learning-based approaches have been designed for this purpose, our empirical study shows that these approaches suffer from long *initialization time* for sufficient training data. In this paper, we introduce the *Compressed Sensing* technique to multivariate time series anomaly detection for rapid initialization. To build a jump-starting anomaly detector, we propose an approach named *JumpStarter*. Based on domain-specific insights, we design a shape-based clustering algorithm as well as an outlier-resistant sampling algorithm for *JumpStarter*. With real-world multivariate time series datasets collected from two Internet companies, our results show that *JumpStarter* achieves an average F1 score of 94.12%, significantly outperforming the state-of-the-art anomaly detection algorithms, with a much shorter initialization time of twenty minutes. We have applied *JumpStarter* in online service systems and gained useful lessons in real-world scenarios.

1 Introduction

In recent years, online service systems based on cloud computing, *e.g.*, online office, e-commerce, are becoming increasingly popular. For example, the number of daily meeting participants of the video conferencing app Zoom jumps to over 300 million before May 2020 [31]. Due to the complexity and the large scale of online service systems, automatic anomaly detection is of ultimate importance to guarantee their reliability [36]. To closely monitor the quality of service, online service providers or cloud computing platforms, such as Microsoft and AWS, continuously collect the monitoring data of each performance metric (*e.g.*, CPU utilization, average response time, and requests per second) at equally spaced intervals [3, 33]. The monitoring data of a metric form a univariate time series, and thus that of a service system, which has multiple metrics, constitutes a multivariate time series.

Traditional multivariate time series anomaly detection approaches are typically based on detecting univariate time series [15, 32, 36]. However, operators are concerned about the status of the overall service rather than that of a specific metric [28]. Because the univariate time series anomaly detection cannot capture the complex temporal relationships among different univariate time series [28], they tend to cause alert storms [5]. To address this problem, recent works [12, 22–24, 28, 34] use deep learning techniques to build learning models for multivariate time series anomaly detection. For example, the state-of-the-art approach, OmniAnomaly [28], utilizes a stochastic recurrent neural network model to learn the temporal relationships of multivariate time series.

Learning-based approaches are hardly applicable in practice because they usually require a long period of training data. Online service systems are deployed or changed very frequently to deploy new features, fix bugs [35], *etc.* In large service providers, such as Google [3] and Baidu [35], it is reported that thousands of software changes are deployed every day. Due to these software changes, the data distribution of multivariate time series can change dramatically, which is called the expected concept drift [17]. For example, when operators conduct a software change to deploy a service to more instances, the metric "Requests Per Second" in each instance will drop significantly as shown in Figure 1. This is expected to operators, and they do not need to roll back the software change. After the change, this will cause a lot of false alarms or false positives, since it invalidates the learning-based anomaly detection models trained based on the data before the change [17]. It is because the common assumption in deep learning that the data distribution must remain the same across training and test set [10] is violated. Therefore, these learning-based approaches have to be retrained. This retraining process, however, can consume tens to hundreds of days [28, 32] before reaching steady state.

To quantitatively measure how long it takes to "initialize"

*Shenglin Zhang is the corresponding author.

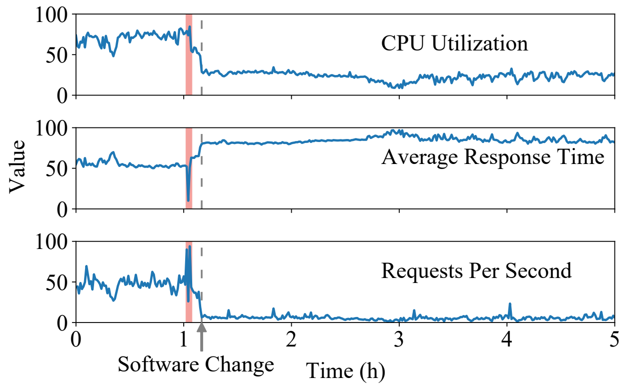


Figure 1: The multivariate time series (selected as examples) of an online service system before and after a software change. The red segment is labeled anomalous.

an anomaly detection model for multivariate time series, we first define *initialization time*, as the time lag between when the model is launched and when it becomes well trained. We then conduct an empirical study based on the datasets collected from real-world online service systems. Through this study, we summarize two key findings: (1) The average initialization time of existing learning-based approaches ranges from ten days to one hundred days. (2) The state-of-the-art approaches (*i.e.*, OmniAnomaly [28] and MSCRED [34]) do not achieve a satisfactory performance when they are improved with *incremental retraining* [15]. When the period of training data is short, the accuracy of these approaches are low. Therefore, we aim to design a robust anomaly detection approach for multivariate time series with small initialization time.

In this paper, we propose a novel multivariate time series anomaly detection approach, called ***JumpStarter***, that is based on *Compressed Sensing* (CS). CS is a signal processing technique where high-energy components in a matrix (multivariate time series) are sparse (*i.e.*, have few high-energy components) [4]. Hence, the difference between the original and the reconstructed multivariate time series, comprised only of low-energy components, should resemble white noise, when the original time series contains no anomaly. The intuition behind using CS for anomaly detection is that anomalies in multivariate time series, such as jitters, sudden drops or surges, usually manifest themselves as strong signals that contain high-energy components, which would differ significantly from white noise [16]. Hence we can tell whether a time series contains anomalies by checking whether the difference between the original and the reconstructed multivariate time series in a sliding window [32] looks very differently from white noise. Since CS only uses a fixed-length window to “train” an anomaly detection model, the initialization time of *JumpStarter* depends on the window size, which is typically twenty minutes. It is much shorter than the initialization time of learning-based approaches. We now provide an intuitive explanation of why CS-based *JumpStarter* requires much less training data than those learning-based approaches. A learning-based approach has to *explicitly* learn the “behavior" (probability distribution) of a normal multivariate time series in order to detect anomalies. In comparison, in *JumpStarter* the reconstructed multivariate time series of *implicitly* inherits this normal behavior without involving any explicit learning. Due to the complexity of real-world scenarios, however, it is challenging to apply CS to implement a jump-starting multivariate time series anomaly detection in the following situations:

Large number of time series. In large-scale online service systems, tens of time series are monitored for each system forming a multivariate time series [17]. Typically, it is time-consuming for CS to reconstruct a large number of time series, because it needs to solve a convex optimization problem whose time complexity depends on the number of time series. To tackle this challenge, we cluster these time series based on shape. Since the time series in the same group exhibit a similar shape, they can be reconstructed efficiently without losing temporal relationships [14].

Sampling from random anomalous segments in time series. CS typically uses a Gaussian distribution to sample from the original time series to guarantee Restricted Isometry Property (RIP). Nevertheless, it may inevitably generate reconstructed time series sampled from some long-lasting anomalous segments. The anomaly detection model based on these reconstructed time series can cause some false positives and/or false negatives. Therefore, we design an outlier-resistant sampling algorithm to sample from normal time series segments rather than anomalous ones.

We conducted a comprehensive study to evaluate the performance of *JumpStarter* based on three datasets from 28 and 30 large-scale industrial online service systems of two Internet companies, respectively. The first dataset, from company $\mathcal{A}$, is a 5-week-long time-series open dataset. The other two datasets are from a top-tier global content platform company $\mathcal{B}$, which provides service for more than 800 million users around the world. These datasets contain 7-week-long time series. Our experimental results illustrate that *JumpStarter* outperforms both the state-of-the-art learning-based multivariate time series anomaly detection approaches (*i.e.*, OmniAnomaly [28] and MSCRED [34]) clustering-based LESINN [20]. Also, *JumpStarter* is more suitable than RRCF [11] used in AWS CloudWatch. The average F1 score of *JumpStarter* is 94.12%, while those of the other four approaches are 86.51%, 59.64%, 82.5%, and 36.01% respectively. Also, our results demonstrate that the main components in *JumpStarter* (*i.e.*, shape-based clustering and outlier-resistant sampling) significantly contribute to the overall performance of *JumpStarter*. *JumpStarter* achieves good accuracy with an initialization time as short as twenty minutes, open sourced at `https://github.com/NetManAIOps/JumpStarter`. We applied *JumpStarter* to $\mathcal{B}$ and it indeed achieved a good anomaly detection performance in practice. We also present two cases

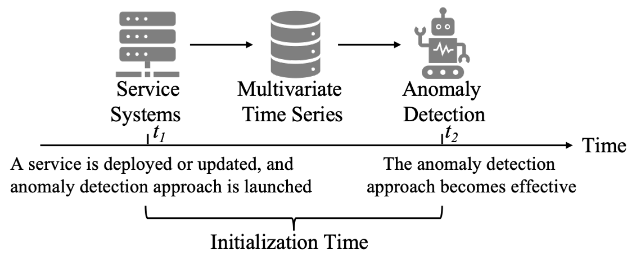

Service Systems
Multivariate Time Series
Anomaly Detection
t_1
t_2
Time
A service is deployed or updated, and anomaly detection approach is launched
The anomaly detection approach becomes effective
Initialization Time

Figure 2: The initialization time of an anomaly detector

and gain lessons for both academia and industry.

2 Background and Empirical Study

2.1 Background

Multivariate time series. In an online service system, operators continuously collect monitoring data of multiple metrics or extract numerical values from logs [37]. A service level metric (*e.g.*, average response time), or a machine level metric (*e.g.*, CPU utilization, memory utilization), is usually collected by equal interval, forming a univariate time series. Any univariate time series alone, however, cannot capture all types of system's performance issues [28]. Because a system typically has a collection of monitoring metrics, it can be denoted as a multivariate time series [17], which includes diverse types of univariate time series and thus track various aspects of performance issues. With the scale and complexity of the system increasing, it is becoming more difficult to manually inspect system anomalies. Therefore, multivariate time series anomaly detection is of great importance [28, 34]. We denote a multivariate time series at time t as $\mathbf{X}_t = [\mathbf{x}_t^1, \mathbf{x}_t^2 ..., \mathbf{x}_t^n]^{\mathrm{T}}$, where $\mathbf{x}_t^i = [x_{t-w+1}^i, x_{t-w+2}^i, ..., x_t^i]$ is the univariate time series of the i^{th} monitoring metric, n is the number of metrics, and w is the observation window size. We apply the sliding window, which is a common practice in time series anomaly detection [32, 35], to construct $\mathbf{X}_t$.

Anomaly detection. Anomaly detection using multivariate time series [28] is important in online service systems. In previous anomaly detection works [12, 22–24, 28, 34], operators have a rough consensus on the following points: 1) A multivariate time series anomaly is a data point or a data segment that significantly deviates from operators' expectations of normal behavior, and it can be visually observed (*e.g.*, in Figure 1). 2) An anomaly indicates something might have gone wrong, although further investigation may still be needed for verification. 3) Anomaly detection is often used as a failure discovery mechanism. Formally, we define multivariate time series anomaly detection: for time t, given its multivariate time series $\mathbf{X}_t$, we determine whether an anomaly occurs (*e.g.*, jitter, sudden drop or surge), which is denoted by $y_t = 1$ if yes and $y_t = 0$ otherwise.

Table 1: Comparison of the initialization time (days) on three datasets (S1∼S3) used in their works. * denotes univariate time series anomaly detector, which can be used for multivariate time series by combining it with majority vote [28].

Approach	S1	S2	S3	Avg.
MSCRED [34]	7	13	-	10
OmniAnomaly [28]	17	15	17	16.3
LSTM-NDT [12]	69	36	-	52.5
* Opprentice [15]	56	56	56	56
* Donut [32]	102	110	99	103.6

2.2 An Empirical Study on Initialization Time

Anomaly detection initialization time. With a new service being deployed or updated, operators usually launch an anomaly detection approach for it. The initialization time of the anomaly detection approach is the time lag between when it is launched (t_1) and when it becomes effective (t_2), as shown in Figure 2. Many prior approaches, *e.g.*, [12, 22, 23, 28, 34], use a learning-based workflow to detect anomalies. Typically, they are periodically trained based on historical data [15]. The initialization time of these approaches, *e.g.*, tens of days, is relatively long, because they usually need to offer a lot of historical data for training. In Table 1 we list the suggested initialization time of five learning-based anomaly detection approaches on different datasets. For example, OmniAnomaly [28] used two robot system datasets (denoted as S1, S2) and a server dataset (S3, which also used in our experiment as D1). From the last column of Table 1, we can see that the average initialization time of these approaches ranges from 10 days to more than one hundred days, indicating that it is unsuitable to use these approaches for newly deployed or updated systems.

Incremental retraining. Considering the long initialization time of learning-based anomaly detection approaches, one may suggest incremental retaining, *i.e.*, gradually (incrementally) adding a short-period (say one day) of data to train these approaches. In this way, we can improve the performance of these approaches step by step. Adding one day's data each time is because these learning-based approaches need at least thousands of data points to converge [32]. We then try to apply incremental retraining to the state-of-the-art multivariate time series anomaly detection approaches, *i.e.*, OmniAnomaly [28] and MECRED [34]. The dataset is the same as what is used in OmniAnomaly (see §4.1 for more details). We gradually enlarge the training set from one day's data to 13 days' data (*i.e.*, the largest training set of this dataset), and the testing set remains as the data collected after the 13th day.

This sounds ideal, but anomaly detection using incremental retraining cannot ensure satisfactory performance. Figure 3 shows the average F1 score and training time of OmniAnomaly and MECRED as the period of training data increases (day by day), respectively. From Figure 3(a), we can see that the average F1 scores of both OmniAnomaly and MECRED increase along with more training data being used, and

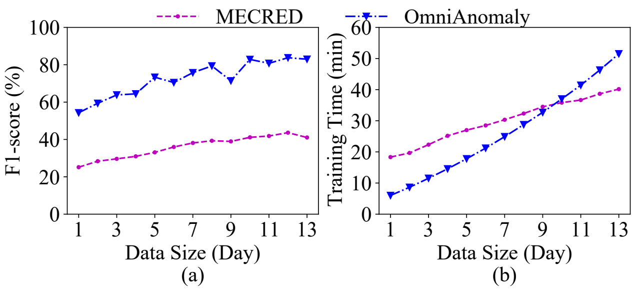


Figure 3: Performance of OmniAnomaly [28] and MECRED [34] by incremental retraining.

they do not converge until 10 days' data is used for training. One primary reason is that these learning-based approaches have to *explicitly* learn the probability distribution of a multivariate time series from a large amount of training data to capture its normal behavior. Figure 3(b) shows that the training time of both OmniAnomaly and MECRED increases linearly with the size of training data. When the training dataset contains 10 days of data, it takes about 35 minutes to train OmniAnomaly or MECRED. Therefore, these approaches are not suitable for newly deployed or updated systems due to their non-robustness and considerable training cost.

3 *JumpStarter* Approach

3.1 Key Idea and Challenges

To deal with the aforementioned limitations of learning-based approaches, we propose to use Compressed Sensing (CS) [9] for multivariate time series anomaly detection. CS is a signal processing technique for reconstructing a signal from a series of sampling measurements [8]. The signal reconstructed from these samples preserves the high-energy components of the original signal with high probability under some mild assumptions [8]. As explained earlier, we can detect anomalies in a multivariate time series by checking whether the reconstructed signal differs from the original signal (multivariate time series) by more than white noise. Since CS does not require any training, the initialization time of the CS-based anomaly detection for $\mathbf{X}_t$ is the window size w.

Two Strawman Solutions using CS. Intuitively, we can apply CS to reconstruct $\mathbf{X}_t$ in two ways: treating $\mathbf{X}_t$ as a whole $n \times w$ matrix, or as n separate univariate time series. In the former way, we randomly sample some data from $\mathbf{X}_t$ and then reconstruct it following [26] (more details to be provided in §3.5). Figure 4(a) shows the reconstructed multivariate time series. As shaded in Figure 4(a), there is a significant difference between the original (blue lines) and the reconstructed multivariate time series (dashed brown lines) when an anomaly occurs. This is a desired behavior (of CS). However, the first two reconstructed time series fluctuate frequently all the time, whereas both original time series are stable except

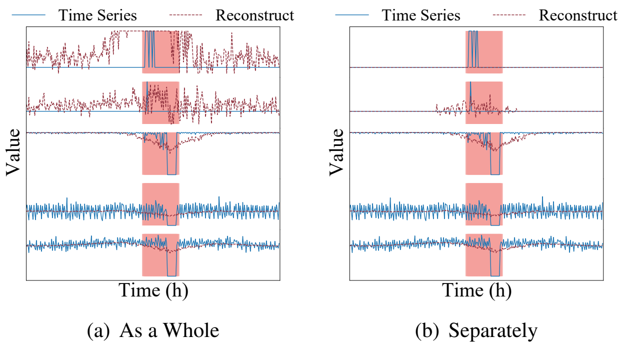


Figure 4: Examples of CS-based anomaly detection when the multivariate time series is reconstructed as a whole $n \times w$ matrix (a) or as n separate univariate time series (b). The red-shaded regions denote the anomalies labeled by operators.

during the anomaly window (shaded). This is an undesired behavior that indicates inaccurate reconstruction. In the latter way, as shown in Figure 4(b), the difference between the original and reconstructed univariate time series manifests as white noise in normal segments and as large fluctuations in anomalous ones, which accurately captures the anomalies for each *univariate time series*. However, it cannot capture the complex relationships among multivariate time series [28]. Moreover, due to the challenge of a large number of univariate time series, the separate reconstruction is more computationally expensive.

Problem of Random Gaussian Sampling. The first step of our CS approach is to sample from a multivariate time series. The sampled matrix needs to guarantee Restricted Isometry Property (RIP) [4] so that it can reconstruct the original multivariate time series properly. It has been shown that random Gaussian sampling satisfies RIP [9]. Random sampling, however, samples some data points also from anomalous segments. In this case, the reconstructed multivariate time series will not be significantly different from the original one when a system becomes anomalous. Thus it inevitably degrades the anomaly detection performance.

3.2 Overview of *JumpStarter*

The *JumpStarter* approach, which consists of both offline and online processing procedures, is shown in Figure 5. To tackle the challenge of a large number of time series, we adopt a *shape-based clustering* method to group the univariate time series of a multivariate time series into several groups in the offline processing. The sliding window technique is applied to multivariate time series in the online anomaly detection. For each group of univariate time series, we propose a novel *outlier-resistant sampling* algorithm to solve the challenge introduced by sampling from anomalous segments, and apply *compressed sensing* to reconstruct them. After that, we concatenate these reconstructed time series, measure the differ-

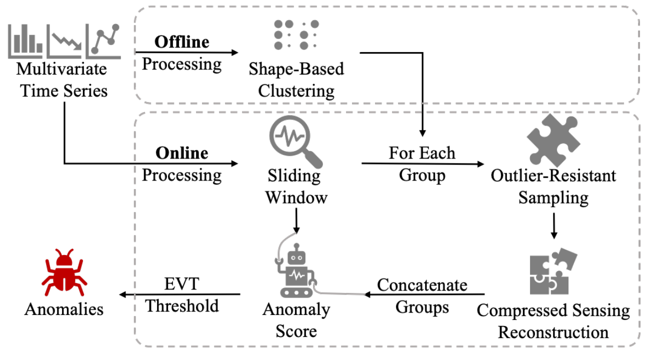


Figure 5: *JumpStarter* approach consists of offline processing and online processing, of which output is whether anomaly or not.

ence between the original and the reconstructed multivariate time series as an anomaly score, and detect anomalies using *EVT threshold* against the anomaly score.

3.3 Shape-Based Clustering

While each strawman solutions has its advantages and drawbacks, we find a way to combine them to get the better of both. Specifically, we split $\mathbf{X}_t$ into several clusters of time series and reconstruct each cluster. The question is, how this splitting should be done. Recall the first strawman solution cannot deal with time series of different shapes. Hence, we choose to split $\mathbf{X}_t$ based on their shape. This solution can achieve both high accuracy and high efficiency.

We adopt shape-based distance [21], a cross correlation-based method, to measure the distance between two univariate time series. It can achieve high computational efficiency when dealing with high-dimensional time series. Different from ROCKA [14], we use hierarchical clustering, which is efficient and need not manually configure the number of clusters, as the base clustering algorithm. Table 2 illustrates an example of clustering results. The nine univariate time series of a multivariate time series are grouped into three clusters. In each cluster, the time series are correlated to the physical meaning of their corresponding monitoring metrics, demonstrating that our method is intuitive.

We cluster univariate time series for every multivariate time series based on one-day worth of data, because most univariate time series are roughly periodical with a 24-hour cycle that coincides with customers' diurnal usage pattern [17]. Moreover, we observe that the shape of a univariate time series usually remains unchanged after a software change [38]. As a result, it has no need to re-cluster after a software change.

3.4 Outlier-Resistant Sampling

Insight. To solve the problem of random Gaussian sampling, we gain insight from the investigation of a large number

Table 2: An example of clustering the multivariate time series into three clusters. Each univariate time series is named based on its corresponding monitoring metric.

#	Cluster of Univariate Time Series	Explanation
1	`rx-pkts-eth0`, `rx-bytes-eth0`	# received packets/bytes
2	`tcp-insegs`, `tcp-outsegs`, `tx-pkts-eth0`	TCP network metrics
3	`cpu-ctxt`, `cpu-user`, `cpu-system`, `cpu-nice`	CPU utilization metrics

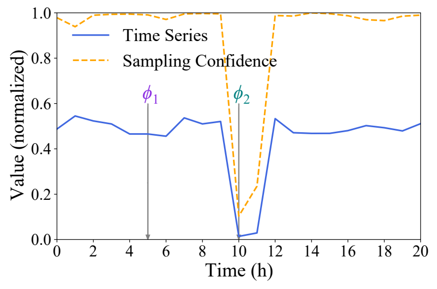
of online service systems and the discussion with operators. Anomalies rarely occur in real-world scenarios [15, 32]. That is, anomalies are usually outliers in an observation (sliding) window [16]. If an anomaly lasts longer than the window size, it can be captured from the beginning since it is significantly different from the normal pattern. Therefore, we can adopt a simple outlier detection algorithm to obtain the *sampling confidence* of each data point. The higher a data point is likely to be an outlier, the lower its sampling confidence is, and the less likely it will be selected. Based on this insight, we design an outlier-resistant sampling algorithm, *i.e.*, one-dimensional random Gaussian, which not only guarantees RIP but also resists outliers.

Algorithm. We describe the design of outlier-resistant sampling in Algorithm 1 and show the main steps in Figure 6. After the shape-based clustering, for each cluster, we can obtain an $w*k$ matrix, which is constituted of k univariate time series. We also configure a sampling ratio, θ, as the input of the algorithm. To begin with, from Line 1 to 4 of Algorithm 1, we initialize a zero $m*w$ matrix $\mathbf{T}$, where $m = \lceil w*\theta \rceil$. ϕ is initialized as a vector containing m random Gaussian sample timestamps ranging from 0 to w. Motivated by [20], we adopt a light-weight algorithm LESINN to calculate the sampling confidence vector $\mathbf{sc}$, which determines the sampling confidence of each timestamp. Nevertheless, as demonstrated in §4.2, LESINN itself cannot effectively handle multivariate time series anomaly detection because it cannot capture the complex temporal relationships among these time series. Figure 6(a) shows one example of the original time series (blue line) with a window size of 20 and its sampling confidence (orange dotted line) for each timestamp.

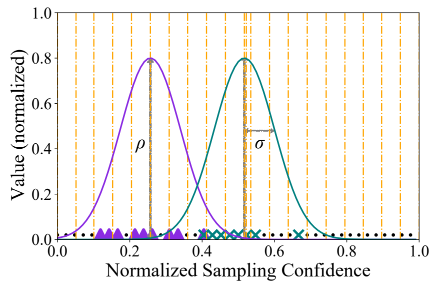
Then, we perform value sampling (Line 5 to 14). We first normalize $\mathbf{sc}$ to make its values sum up to one, and create R equal-width *steps*. Each *step* is mapped to a timestamp t based on the normalized $\mathbf{sc}$ (Line 8). For each *step*, we randomly sample a value from $(0,1)$, and compare it with the probability of a similar version of Gaussian distribution:

$$P_i(step) = \rho \cdot exp(-\frac{(\phi_i - step)^2}{2\sigma^2}) \tag{1}$$

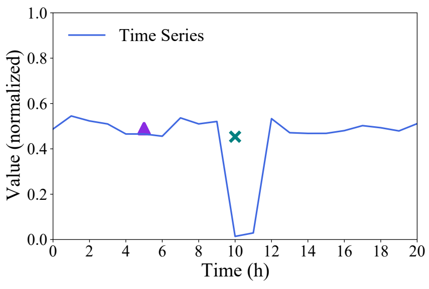
Note that ρ is a parameter representing the max sampling points (height in Figure 6(b)). For more theoretical details, please refer to the work of Barranca, *et al.* [2]. σ is the standard deviation of the distribution. If the random value is smaller than $P(step)$, we add one to $\mathbf{T}[i][t]$. Note that after



(a) Initialize



(b) Sampling



(c) Results

Figure 6: A toy example of outlier-resistant sampling. In (a), to make it simple, we just plot one time series whose values are normalized, and it has an anomalous segment ranging from (timestamp) 9 to 12. We set $m = 2$ in this example. In (b), the small black dot is steps, and the purple and green marks are "selected" steps. (c) shows two sampled data points.

all the iterations, the randomly sampled timestamps in ϕ may not be "shot" by t. Therefore, we add one to $\mathbf{T}[i][\phi_i]$ for each sampled timestamp i. Figure 6(b) shows a concrete example of sampling values. In this figure, there are $R = 42$ steps (the small points in the bottom), and the normalized **sc** (the orange box) is shown in its original temporal order, which represents a timestamp.

Finally, we normalize each column of $\mathbf{T}$ (Line 15 to 18). The sampling matrix $\mathbf{B}$ is the dot product of $\mathbf{T}$ and $\mathbf{X}_t^c$. As shown in Figure 6(c), the purple dot and green cross are the sampled data points in $\mathbf{B}$. We can see that our algorithm is resistant to anomalous data points because the sampled green cross point represents "normal pattern" even though its corresponding point in the original time series is anomalous. Note that although we apply LESINN to calculate the sampling confidence vector, which is of great importance to the resistant outlier, our algorithm is also robust to other sampling confidence measurements beyond LESINN.

Algorithm 1: Outlier-Resistant Sampling

```
Input: $\mathbf{X}_t^c(w*k)$: $k$ univariate time series of $\mathbf{X}_t$ in cluster $c$,
       $\theta$: Initial sampling ratio
Output: Sampling matrix $\mathbf{B}(m*k)$
1  $m \leftarrow \lceil w*\theta \rceil$
2  $\mathbf{T}(m*w) \leftarrow 0$
3  $\phi(m*1) \leftarrow$ randomly sampling $m$ timestamps from $[0, w]$
4  $\mathbf{sc}(w*1) \leftarrow$ SamplingConfidence $(\mathbf{X}_w)$
5  $\mathbf{sc} \leftarrow \frac{\mathbf{sc}}{\sum_{j=0}^{w-1} \mathbf{sc}_j}$
6  $step = 0$
7  while $step \le 1$ do
8      $t \leftarrow \arg\min_{0 \le a < w} step < \sum_{j=0}^{a} \mathbf{sc}(j)$
9      foreach $i \in [0, m)$ do
10         if random(0, 1) $< P_i(step)$ then
11             $\mathbf{T}[i][t] \leftarrow \mathbf{T}[i][t] + 1$
12     $step \leftarrow step + 1/R$
13 foreach $i \in [0, m)$ do
14     $\mathbf{T}[i][\phi_i] \leftarrow \mathbf{T}[i][\phi_i] + 1$
15 foreach $i \in [0, m)$ do
16     $\mathbf{T}[i] \leftarrow \mathbf{T}[i] / \sum_{j=0}^{w} \mathbf{T}[i][j])$
17 $\mathbf{B} \leftarrow \mathbf{T} \cdot \mathbf{X}_t^c$
18 return $\mathbf{B}$
```

3.5 Compressed Sensing Reconstruction

Here we provide a quick introduction to the signal reconstruction step in compressed sensing, using this anomaly detection problem as the context. The objective of compressed sensing reconstruction is to "solve" [4]:

$$\mathbf{A}\mathbf{X'}_t = \mathbf{B} \tag{2}$$

where $\mathbf{X'}_t$ is the multivariate time series to be reconstructed from $\mathbf{B}$ that is sampled from the original $\mathbf{X}_t$. We put quotations around the word "solve", since this equation is not solvable in the usual sense, as it is under-determined. Rather, ideally we would like to compute the sparest (*i.e.*, containing the smallest number of nonzero components) such $\mathbf{X'}_t$. However, this computation, known as L^0 minimization, is NP-complete. A classic CS result is that, when $\mathbf{X}_t$ is sparse, minimizing the L^1 of $\mathbf{X'}_t$ (while satisfying Equation 2) results in the same solution as the L^0 minimization with high probability [4].

The sampling matrix $\mathbf{A}$ is calculated as:

$$\mathbf{A} = \phi(\mathbf{D} \otimes \mathbf{D}^{\mathbf{T}}) \tag{3}$$

where $\mathbf{D}$ is the inverse discrete cosine transform [13] of the original time series $\mathbf{X}_t^c$, and $\otimes$ is the Kronecker product [29].

To solve Equation 2 (by L^1 minimization), we adopt CVXPY [7], an efficient convex optimization tool set, to calculate the L^1 minimum [8]. CVXPY may return no result in some edge cases because the equation is non-homogeneous. Therefore, we gradually increase θ by 0.1 to avoid such a scenario. We set $\theta = 0.2$ based on the evaluation experiments as shown in §4.4. Reconstructing each cluster of univariate time series in a multivariate time series is much more efficient than reconstructing the whole multivariate time series (§4.4).

Anomaly score. We first obtain the reconstructed time series for each cluster of univariate time series to form an original multivariate time series. We then concatenate the reconstructed univariate time series to form a reconstructed multivariate time series $\mathbf{X'}_t$. Note that the original and reconstructed multivariate time series have the same order of univariate time series. Intuitively, an anomaly score is needed to measure the similarity between the original and the reconstructed multivariate time series. We measure the differences of the n time series between $\mathbf{X}_t$ and $\mathbf{X'}_t$ using euclidean distance [15]: $\mathbf{d}_t^i = |\mathbf{x}_t^i - \mathbf{x'}_t^i|$, where $\mathbf{x'}_t^i$ is the reconstructed univariate time series of $\mathbf{x}_t^i$. To avoid an anomaly score being dominated by a single significant spike in a univariate time series, we calculate s_t using the harmonic mean of $\mathbf{d}_i$, *i.e.*, $s_t = n/(\sum_{i=1}^{n} \mathbf{d}_t^{i^{-1}})$.

Choosing threshold. To properly generate anomaly alerts, we need to accurately choose a threshold to determine whether an anomaly score is high enough to trigger an alert. A static threshold does not work well since the data distribution changes over time. Because an extreme value of the anomaly score generated by *JumpStarter* usually represents an anomaly, we adopt the widely used Extreme Value Theory (EVT) [27] to tailor the anomaly threshold automatically. EVT is a statistical theory aiming to find the law of extreme values, and it does not assume data distribution. It has been demonstrated to accurately choose the threshold for anomaly detection methods [17,28]. Note that EVT for choosing threshold is not the main contribution of our work.

4 Experiments

In the study, we address the following research questions:
RQ1: How well does *JumpStarter* perform in multivariate time series anomaly detection?
RQ2: Does each component contribute to *JumpStarter*?
RQ3: How do the major parameters of *JumpStarter* influence its performance?

4.1 Experimental Design

4.1.1 Datasets

We conduct experiments on three datasets, including one open dataset[1] – D1 from a large Internet company $\mathcal{A}$, and two datasets (D2, D3) collected from a top-tier global content platform $\mathcal{B}$ providing services for over 800 million daily active (over 1 billion cumulative) users across all of its content platforms. Specifically, D1 is a five-week-long dataset collected from 28 online service systems, and it is sampled once per minute. These 58 online service systems are located in different servers, which provide services such as searching, ranking, and data processing, *etc*. D2 and D3 are two datasets collected from 30 online service systems over two different seven-week-long periods, respectively. They are both sampled once every five minutes.

Table 3: The detailed information of the datasets (# Training/Test Points = # Online Services * n * # Days * collected data points per day)

Dataset	# Online Services	n	# Training Points (# Days)	# Test Points (# Days)	Anomaly ratio
D1	28	38	19,835,340 (13)	19,835,760 (13)	4.16
D2	30	19	3,283,200 (20)	4,104,000 (25)	5.25
D3	30	19	3,283,200 (20)	4,104,000 (25)	20.26

This work studies metrics for a single service hosted on one or multiple machines. These metrics are equally important and have no hierarchy among them. The ground truth of anomalies in all the three datasets are manually labeled by operators based on performance issues and failure tickets. The point-wise anomaly rates ($\frac{\textit{\# anomaly data points}}{\textit{\# total data points}}$) are diverse in these datasets. For example, the anomaly rate of D3 (20.26%) is much higher than those of D1 (4.16%) and D2 (5.25%), mainly because D3 contains a severe outage that lasted a long time. Table 3 lists the detailed information of each dataset, including the number of metrics (n), the scale of the training and test sets, and the anomalies ratio. For each service, the monitoring metrics constitute its multivariate time series. The numbers of metrics monitored in D1, D2 and D3 are 38, 19 and 19, respectively. Monitoring tens of metrics is a typical setting for online service systems.

4.1.2 Compared Approaches

We compare *JumpStarter* with two learning-based unsupervised approaches for multivariate time series anomaly detection, namely, MSCRED and OmniAnomaly. We also compare it with two other anomaly detection algorithms: robust random cut forest (RRCF) and least similar nearest neighbors (LESINN). Since it has been demonstrated [28] that univariate time series anomaly detection approaches are not suitable for multivariate time series, we do not compare *JumpStarter* with the baseline methods designed for univariate time series.

RRCF [11]. RRCF is the base anomaly detection algorithm used in AWS CloudWatch, which improves the robustness of original random cut forest probabilistic data structure when detecting anomalies in streaming data. It is open-sourced[2].

LESINN [20]. LESINN is a time series outlier detection algorithm. For each data point, it calculates the least similar nearest neighbors of it in a time window. If the data point does not have any similar nearest neighbor, it is an outlier.

MSCRED [34]. MSCRED first encodes the temporal correlations among the time series using an attention-based ConvLSTM network, and then reconstructs time series to detect anomalies using a convolutional decoder.

[1]https://github.com/NetManAIOps/OmniAnomaly

[2]https://github.com/aws/random-cut-forest-by-aws

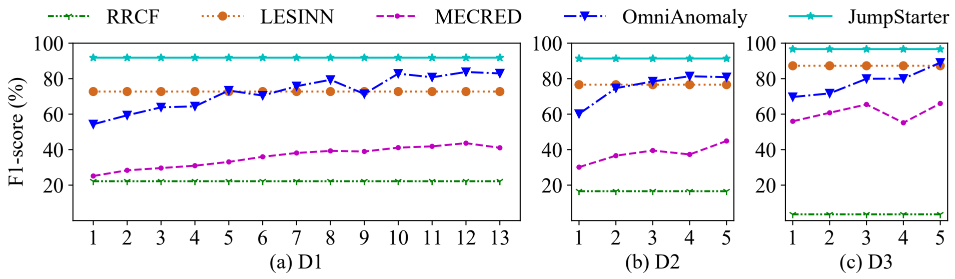


Figure 7: The average F1 score for the three datasets D1, D2, and D3 as a function of the training dataset size in segments.

OmniAnomaly [28]. OmniAnomaly is an unsupervised deep learning based approach. It glues GRU and VAE to model the temporal dependence and stochasticity of time series.

4.1.3 Implementation

JumpStarter is implemented using Python 3.7. For the hyper-parameters, $\rho = 0.1$ and $\sigma = 0.5$ are used in all settings. The detection window size $w = 20$ and the sampling rate $\sigma = 0.2$ are used in §4.4. Our study is conducted on a Dell R420 server with 16 * Intel Xeon E5-2420 CPUs and a 64GB memory.

4.1.4 Evaluation Metrics

The output of a multivariate time series anomaly detection approach for a specific timestamp is either anomalous or not. We use True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) to label an anomaly detection result according to the ground truth. A TP is an anomaly both confirmed by operators and detected by the approach. If an anomaly is labeled by operators but not detected by the approach, we label the item as an FN. An FP is an “anomaly" that is detected by the approach but is actually normal. An item is TN, if neither the operators nor the approach considers it an anomaly. We use three metrics for evaluating the performance of *JumpStarter* and related approaches: Precision = TP / (TP + FP), Recall = TP / (TP + FN), F1 score = 2 * Precision * Recall / (Precision + Recall). The accounting of the three metrics is point-adjusted. That is, if any point in an anomalous segment in the ground truth is detected, we consider the entire segment, or all anomalous points therein, as detected correctly. Point-adjusted metrics are widely adopted in anomaly detection [28, 32], since operators care more about anomalies in a contiguous segment than point-wise anomalies.

4.2 RQ1: Performance of *JumpStarter*

We evaluate two aspects of the performance of anomaly detection each using a different partitioning of training and test sets. First, we conduct service anomaly detection in the online mode and evaluate it as an online experiment. In addition, we collect ten software changes from $\mathcal{B}$ for evaluating the performance of these approaches in reacting to a software change. Second, in the offline experiment, we adopt the same experiment settings as used in previous work.

Online experiment. We evenly split the training set of D1 into 13 segments (1-day-long data per segment and each has a similar number of anomalies), and D2 and D3 each into 5 segments (4-day-long data per segment and each has a similar number of anomalies). D2 and D3 have a longer segment because they have fewer anomalies per day. For each dataset, the test set remains the same for a fair comparison (see Table 3). Figure 7 shows the average F1 score of *JumpStarter* and four baseline methods as the amount (scale) of training data increases from 1 segment to 13 consecutive segments for D1, and to 5 consecutive segments for D2 and D3. As for *JumpStarter*, RRCF and LESINN, they conduct anomaly detection without any training data. Therefore, their performance stays the same when the scale of training data varies.

We can see that *JumpStarter* performs significantly better than the four baseline approaches across all segments on all the three datasets. RRCF is less accurate than the other approaches because it aims to detect the anomalous behavior of *a single data point*, which is not suitable in our scenario where the anomalous behavior of *a time series segment* is studied. The F1 scores of learning-based approaches, namely OmniAnomaly and MSCRED, increase as the scale of training set increases, and approach 90% and 60% respectively toward the end. OmniAnomaly achieves higher accuracy than LESINN when the amount of training data is sufficient. MSCRED does not perform well because it mainly focuses on the inter-correlations rather than the overall performance of multivariate time series. For all approaches except RRCF, they achieve the best performance on D3 because the anomalous patterns in D3 are easier to capture than those in the other two datasets. The outperformance of *JumpStarter* will be explained next, when we will perform experiments concerning software changes.

Anomaly detection after software changes. Recall that software changes occur frequently in online service systems. We evaluate the performance of the five approaches during ten software changes deployed on two service systems. We do so using the average false positive rate (FPR), defined as FP / (TN + FP), as the metric. FPR measures the burden of false alarms, which is an appropriate metric here since anomalies

Table 4: Average Precision (P), Recall (R), and F1 Score (F) of *JumpStarter* and related approaches

Method	D1			D2			D3			Avg.		
	P	R	F	P	R	F	P	R	F	P	R	F
RRCF [11]	40.26	54.45	39.54	44.06	39.02	30.10	28.33	75.33	38.38	37.55	56.27	36.01
LESINN [20]	75.49	77.40	76.43	77.50	87.15	82.04	87.02	90.95	88.94	80.00	85.17	82.50
MSCRED [34]	46.19	56.16	50.69	46.91	58.26	51.97	68.21	86.47	76.26	53.77	66.96	59.64
OmniAnomaly [28]	78.19	**95.03**	85.79	77.24	85.84	81.31	89.59	95.02	92.23	81.67	91.96	86.51
JumpStarter	**90.35**	94.31	**92.29**	**92.05**	**94.51**	**93.26**	**94.14**	**99.60**	**96.79**	**92.18**	**96.14**	**94.12**

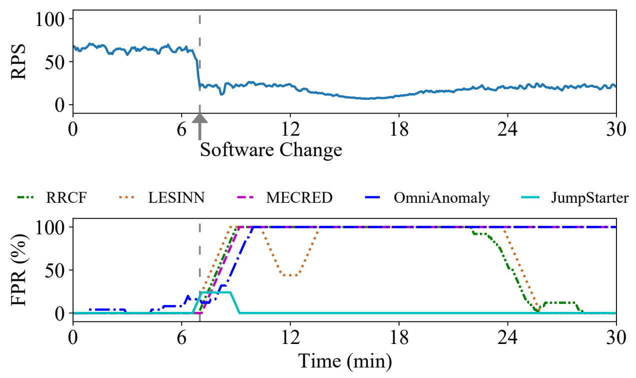


Figure 8: The average False Positive Rate (FPR) during ten software changes. We plot the software changes starting at the seventh minute. The top figure is a constituent univariate time series representing Requests Per Second (RPS).

rarely occur in a short period.

Figure 8 shows the average FPR of the five approaches during the 10 software changes, all of which occur at the seventh minute in the figure. Some constituent univariate times series of the multivariate time series witness level shifts [17]. We observe that all five approaches produce false positives after these software changes. However, *JumpStarter* suffers from high FPR only for about five minutes, after which its FPR becomes quite low. RRCF and LESINN detect anomalies by calculating outliers in a relatively long period, and hence their FPRs do not start to decrease until 17 minutes after software changes. Both OmniAnomaly and MECRED perform well only when the joint distribution of multivariate time series remains unchanged. Consequently, they produce false alarms for a long period after software changes.

Offline experiment. Now we study the potential performance in the offline setting using best F1 score. Since we need a long time to train models of learning-based anomaly detection approaches, we split the dataset into training and test set following the settings in [28]. Then, we calculate the best F1 score of each approach by grid searching their parameters and anomaly thresholds. Table 4 lists the average best F1 scores of *JumpStarter* and baseline approaches on each dataset, as well as their corresponding Precision and Recall. The average best F1 score of *JumpStarter* across the three datasets is 94.12%, significantly higher than those of the other four approaches, which are 86.51%, 59.64%, 82.50%, and 36.01%, respectively. This is because D2 contains a large quantity of noises, and none of the other four approaches is robust to such noises. In contrast, *JumpStarter* reconstructs an anomaly-free time series with outlier-resistant sampling, making it robust to such noises in each dataset.

Table 5: The average initialization time (IT) and detection time (DT) of *JumpStarter* and baseline approaches

Approach	RRCF	LESINN	MSCRED	Omni-Anomaly	*JumpStarter*
IT (min)	20	20	>86400	>86400	20
DT (ms)	41.24	118.63	122.82	191.86	127.13

Efficiency. Table 5 lists the average initialization time and detection time of the five approaches. The initialization time of *JumpStarter*, as demonstrated in §4.4, is only twenty minutes, much shorter than those of other approaches. Although RRCF and LESINN achieves the same initialization time as *JumpStarter*, they suffer from low accuracy as shown in Table 4. The detection time of *JumpStarter* for each multivariate time series is 127.13 ms, similar to that of the other approaches. In *JumpStarter*, the shape-based clustering step takes at most 500ms on all three datasets.

4.3 RQ2: Contributions of Components

In this section, we evaluate the relative contributions of two component techniques in *JumpStarter*, namely shape-based clustering and outlier-resistant sampling, to its outperformance. To this end, we reconfigure *JumpStarter* to create three variants. The first two variants concern *JumpStarter* without shape-based clustering. The first variant, called *w/o Clustering: As a Whole*, is to treat all time series as a whole, whereas the second variant, called *w/o Clustering: Separately*, is to sample and reconstruct time series separately. The third variant, called *w/o Sampling*, is *JumpStarter* without outlier-resistant sampling.

Figure 9 shows the comparison results of *JumpStarter* and its three variants on three datasets in terms of average best F1 scores. *JumpStarter* performs better than all its three variants. Specifically, *JumpStarter* improves the average best F1 score of *w/o Clustering: As a Whole*, *w/o Clustering: Separately*, and *w/o Sampling* by 5.81% $\sim$ 14.90%, 2.58% $\sim$ 9.96%, and

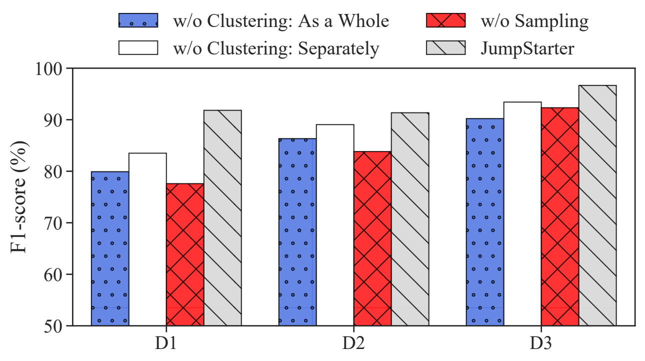


Figure 9: The average F1 score of *JumpStarter* and its three variants on three datasets

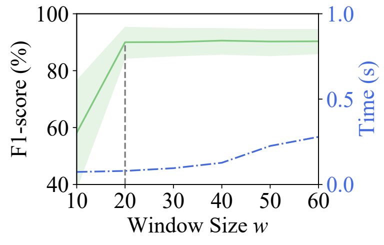
4.69% $\sim$ 18.34%, respectively. The results demonstrate that both shape-based clustering and outlier-resistant sampling contribute significantly to a more accurate *JumpStarter*. In addition, *w/o Clustering: Separately* outperforms *w/o Clustering: As a Whole*, because separate reconstruction of time series more accurately captures anomalous patterns than reconstruction as a whole, as shown in Figure 4.

To compare the computational efficiency of different methods, we compare their detection times. For each multivariate time series, the detection time of *w/o Clustering: As a Whole*, *w/o Clustering: Separately*, *w/o Sampling*, and *JumpStarter* is 7891.45 ms, 2056.56 ms, 121.75 ms, 127.13 ms, respectively. This demonstrates that the shape-based clustering technique significantly improves the computational efficiency of *JumpStarter*. In conclusion, the combination of the shape-based clustering and outlier-resistant sampling facilitates an accurate and efficient *JumpStarter*.

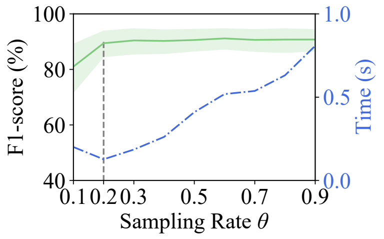
4.4 RQ3: Parameter Sensitivity

Recall that our goal is to shorten the initialization time of anomaly detection. The initialization time of *JumpStarter* depends on the detection window size w. We empirically increase the window size from ten minutes to sixty minutes. Figure 10(a) shows how the average best F1 score and point-wise detection time of *JumpStarter* changes as the window size increases. The accuracy of *JumpStarter* increases before the window size reaches 20 minutes, after which it becomes stable, whereas the detection time increases gradually. Therefore, the window size is preset as twenty minutes, which makes *JumpStarter* both accurate and efficient. Note that for those anomalies lasting longer than 20 minutes, *JumpStarter* is still able to detect them because it can easily capture these anomalies when they start.

Another important parameter of *JumpStarter* is σ, the initial sampling rate. Figure 10(b) shows how the average best F1 score and point-wise detection time of *JumpStarter* changes as σ increases. Similarly, when we increase the sampling rate



(a)



(b)

Figure 10: The average F1 score (green line with error bounds) and average point-wise detection time (dotted blue line) of *JumpStarter* under different parameters.

from 0.1 to 0.2, the F1 score of *JumpStarter* increases, and it becomes stable after that. The point-wise detection time of *JumpStarter* decreases with an increase in the sampling rate from 0.1 to 0.2, and it then increases after that. That is because, if the sampling rate is too low (e.g., 0.1), reconstruction is likely to fail, which in turn increases the number of retries. Therefore, we set the sampling rate to 0.2.

5 Deployment and Discussion

5.1 Success Story

Case study. We applied *JumpStarter* into 30 online service systems in a top-tier global content platform $\mathcal{B}$, which has more than 800 million users around the world. Operators can register a multivariate time series monitor task by ingesting these time series from influxDB and Kafka. From the monitoring dashboard of these services, we can see the multivariate time series of service monitoring. Figure 11 shows some time series of two technical outages.

Case I: Long service response time caused by network issue. As illustrated in Figure 11(a), we observed jitters in time series `tcpext_listendrops` and `tcp_attemptfails`. In the meantime, time series such as `cpu_user` and `load_one` drastically dropped. *JumpStarter* successfully pinpointed this anomaly and generated an alert. After diagnosing the anomaly, operators found that it was a network issue of a database node.

Case II: Service hang-up due to software change. As illustrated in Figure 11(b), time series `tcp_retrans_percentage` witnessed significant jitters and `cpu_idle` plunged to zero. After that, `cpu_sintr`, `cpu_ctxt`, `rx_byptes_eth0` and tx_pkts_eth0 increased significantly. *JumpStarter* detected and reported this anomaly to operators. Operators conducted software changes and a configuration error occurred in the new version. Thanks to *JumpStarter*, operators found out this error in time and quickly rolled out the software change.

Help with root cause diagnosis. *JumpStarter* can help with root cause diagnosis in two aspects. First, hundreds of multivariate time series need to be monitored. After shape-based clustering in *JumpStarter*, operators can focus on limited time series groups with a similar shape of variations. Second, *JumpStarter* respectively calculates the distance between the original univariate time series and the reconstructed ones. Therefore, it can output a rank list of time series' contributions to the overall anomaly. For example, tcpext_listendrops in Figure 11(a) is detected as the most anomalous time series in this figure. It can explicitly indicate the issue caused by the network component.

5.2 Lessons Learned

Different services may prefer precision and recall differently. With collaboration with different services teams, we found that their preferences on precision and recall are diverse. For example, recall weighs more than precision does in a user interactive core service since operators do not want to miss any potential anomaly that can negatively impact the user experience. In addition, precision is more valuable in a data analysis job because operators would better detect anomalies precisely than to obtain a lot of false alerts. Therefore, the F1 score alone is not a suitable metric for all services. Going forward, we can provide operators with an interface to choose their precision and recall preference level. Specifically, *JumpStarter* can accordingly set the anomaly score using different parameters of the EVT algorithm to guide the sensitivity of detection output. We also observe that severe faults (examples in Figure 11) rarely happen in online services but performance issues do happen a lot. We aim to reduce mean time to restore for severe faults in future work.

Alert system is not just anomaly detection. Our *JumpStarter* for robust and quickly initialized anomaly detection is not the end of the story. Building an intelligent alert system based on anomaly detection results is also a complex task in both engineering and academic aspect. Some anomalies may have no or little signal in the monitored time series. Therefore, *JumpStarter* may miss these anomalies. For this scenario, we aim to collect more types of monitoring data, *e.g.*, logs, traces, to build a more comprehensive anomaly detection model. We believe *JumpStarter* can be easily extended for localizing the anomalous metrics, however, there is a significant gap between anomalous metrics and the root causes of anomalies [16,25]. An intelligent alert system needs to merge similar anomalous cases, pinpoint more sharply to the root causes of anomalies. It also had better learn the priority of different anomalous cases [5]. Besides, adeptly integrating domain knowledge into the alert system is also of great importance since the system needs feedback from operators. Therefore, apart from the anomaly detection approach *JumpStarter*, we will improve the alert system behind it.

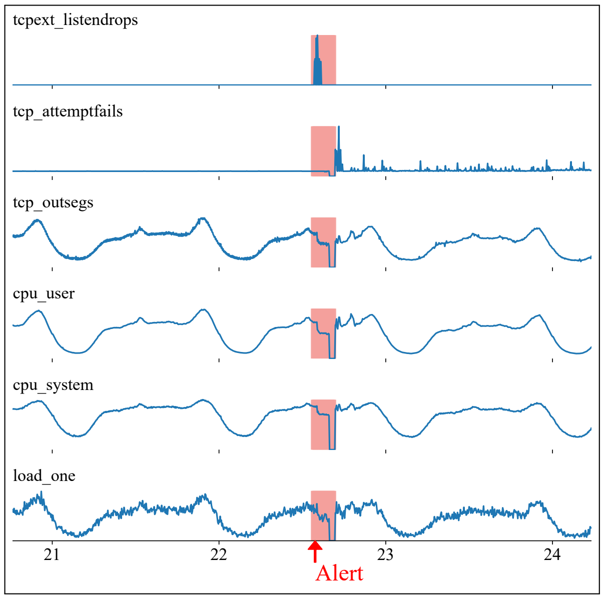
5.3 Threats to Validity

Anomaly labeling. In this work, we use one open dataset from OmniAnomaly and two datasets from real-world services. All the labels in these datasets are provided by operators based on performance issues and incident reports. Manually labeling anomaly points in the timeline may introduce noise (false positives or negatives) because no clear boundaries lie in anomalies and normal patterns. However, domain operators with profound experience suggest the noise in those labels accounts for a very small portion. Besides, operators design evaluation metrics that utilize contiguous anomaly segments instead of point-wise anomalies. Adopting these widely used metrics [17,28,32], we can also eliminate labeling noises.

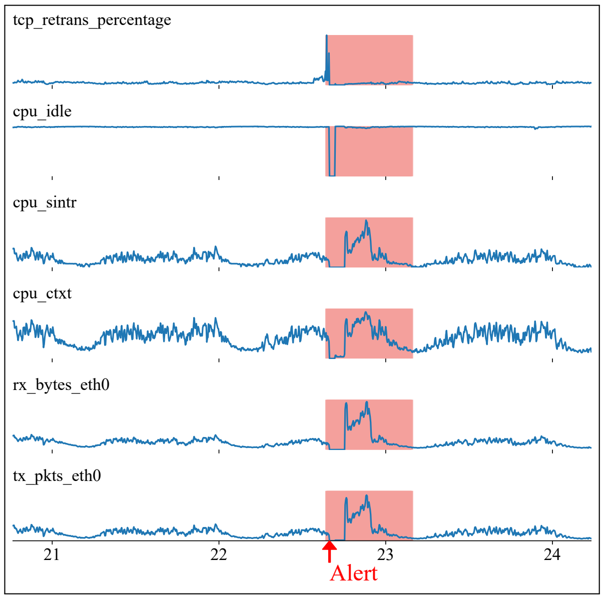
Subject systems. In our experiments, we use an open dataset D1 from a large Internet company. We also collect datasets D2 and D3 from large-scale real-world online service systems. The granularity of the time series of these datasets is one and five minutes, respectively. The efficacy of our algorithm is not influenced by the granularity. With fine-grained granularity, say one second, we believe our algorithm can still work without additional efforts. Since *JumpStarter*'s versatility has been demonstrated using three datasets collected from 58 different services, it should be easy for *JumpStarter* to work with a new dataset. Admittedly, the number of subject services is still limited. We will experiment with *JumpStarter* on a variety of online service systems in the future.

6 Related work

Multivariate time series anomaly detection. Existing approaches include: (1) Traditional statistic method: time series analysis [6], RRCF [11] and clustering-based LESINN [20] do not need training data thus the initialization time is short. However, they all suffer from parameter tuning [15] and being not robust in real practice [17]. (2) Supervised learning based method: Opprentice [15] is an ensemble supervised approach. It needs operators to label anomalies for a long period to train a model. (3) Unsupervised learning based method: LSTM-NDT [12], LSTM-VAE [22], MSCRED [34] and OmniAnomaly [28] build anomaly detection models by learning the anomaly patterns using a large span of historical data. These methods suffer long initialization time for training the model and are less accurate when facing software change [17]. Different from them, we take advantage of compressed sensing, which is a quick initiated signal processing based approach without any training data.



(a) Network Issue



(b) Software Change

Figure 11: Two anomaly cases with selected time series (service performance metrics, *e.g.*, average response time, error rate, are hidden for the confidential reason). Time (X-axis) is shown in days. The alert is generated by *JumpStarter*.

Signal processing based anomaly detection. Since time series of online service systems fluctuate and not always periodic [35]. Other signal processing methods are not suitable in this scenario. For example, Fourier transform can only capture period, the global information of time series [38]. For local information, wavelet analysis can capture local patterns but it is very time consuming [1]. Kalman filtering [18] and PCA [19] are not suitable for variation time series of online service systems [15]. Different from them, our work adopts compressed sensing as the base technique and our experimental results have demonstrated the effectiveness of *JumpStarter* in multivariate time series anomaly detection scenario.

Compressed sensing. As body of theory regarding signal recovery, CS has been widely used in image reconstruction [2], genome-wide association study [30], and many other applications [9]. *JumpStarter* is parallel to them, as it detects anomaly adopting the idea of CS reconstruction.

7 Conclusion

In recent years, online service systems are increasingly deployed on cloud computing platforms. To adapt to frequent changes in online service systems, multivariate time series anomaly detection approaches should be robust and quickly initialized. In this paper, we propose a jump-starting multivariate time series anomaly detection approach with a relatively short initialization time of only twenty minutes. Based on the compressed sensing technique, *JumpStarter* adopts a shape-based clustering strategy to deal with the large number of univariate time series in a multivariate time series, and outlier-resistant sampling to avoid sampling anomalous values. Experiments conducted on 58 real-world online service systems of two Internet companies demonstrate the effectiveness of *JumpStarter*, achieving an average F1 score of 94.12% and outperforming four state-of-the-art approaches. Besides, our results also endorse the contributions of the main components. In particular, we have applied *JumpStarter* in a real-world online service system and gained some useful lessons.

8 Acknowledgements

We thank our shepherd and the anonymous reviewers for their thorough comments and helpful suggestions. We thank Shiyu Ma, Tiankai Yang, and Hanwen Hu for helping with experiments. We appreciate Christopher Zheng for proofreading our paper. This work has been partially supported by the National Key Research and Development Program of China under Grant No.2019YFE0105500, the National Natural Science of China under Grant 62072264, and the Beijing National Research Center for Information Science and Technology (BNRist) key projects. Shenglin Zhang has been partially supported by the National Natural Science Foundation of China under Grant No. 61902200 and the China Postdoctoral Science Foundation under Grant No.2019M651015. Junjie chen has been partially supported by the National Natural Science Foundation of China under Grand No. 62002256. The work of Jun Xu has been partially supported by US NSF through awards CNS-1909048 and CNS-2007006.

References

[1] Vicente Alarcon-Aquino and Javier A Barria. Anomaly detection in communication networks using wavelets. *IEE Proceedings-Communications*, 148(6):355–362, 2001.

[2] Victor J Barranca, Gregor Kovačič, Douglas Zhou, and David Cai. Improved compressive sensing of natural scenes using localized random sampling. *Scientific reports*, 6:31976, 2016.

[3] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, Inc., 2016.

[4] Emmanuel J Candes and Terence Tao. Decoding by linear programming. *Transactions on Information Theory*, 51(12):4203–4215, 2005.

[5] Yujun Chen, Xian Yang, Hang Dong, Xiaoting He, Hongyu Zhang, Qingwei Lin, Junjie Chen, Pu Zhao, Yu Kang, Feng Gao, et al. Identifying linked incidents in large-scale online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 304–314, 2020.

[6] David R Choffnes, Fabián E Bustamante, and Zihui Ge. Crowdsourcing service-level network event monitoring. In *Proceedings of the SIGCOMM Conference*, pages 387–398. ACM, 2010.

[7] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[8] David L Donoho. For most large underdetermined systems of linear equations the minimal l1-norm solution is also the sparsest solution. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 59(6):797–829, 2006.

[9] Yonina C Eldar and Gitta Kutyniok. *Compressed sensing: theory and applications*. Cambridge University Press, 2012.

[10] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *Computing Surveys (CSUR)*, 46(4):1–37, 2014.

[11] Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. Robust random cut forest based anomaly detection on streams. In *International Conference on Machine Learning*, pages 2712–2721. PMLR, 2016.

[12] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 387–395. ACM, 2018.

[13] Andy C Hung and Teresa H-Y Meng. A comparison of fast inverse discrete cosine transform algorithms. *Multimedia Systems*, 2(5):204–217, 1994.

[14] Zhihan Li, Youjian Zhao, Rong Liu, and Dan Pei. Robust and rapid clustering of kpis for large-scale anomaly detection. In *Proceedings of the 26th IWQoS International Symposium on Quality of Service*, pages 1–10. IEEE, 2018.

[15] Dapeng Liu, Youjian Zhao, Haowen Xu, Yongqian Sun, Dan Pei, Jiao Luo, Xiaowei Jing, and Mei Feng. Opprentice: Towards practical and automatic anomaly detection through machine learning. In *Proceedings of the IMC Internet Measurement Conference*, pages 211–224. ACM, 2015.

[16] Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, et al. Diagnosing root causes of intermittent slow queries in cloud databases. *Proceedings of the VLDB Endowment*, 13(8):1176–1189, 2020.

[17] Minghua Ma, Shenglin Zhang, Dan Pei, Xin Huang, and Hongwei Dai. Robust and rapid adaption for concept drift in software system anomaly detection. In *Proceedings of the 29th ISSRE International Symposium on Software Reliability Engineering*, pages 13–24. IEEE, 2018.

[18] Joseph Ndong and Kavé Salamatian. Signal processing-based anomaly detection techniques: a comparative analysis. In *Proc. 2011 3rd International Conference on Evolving Internet*, pages 32–39, 2011.

[19] Netflix. Rad — outlier detection on big data. https://netflixtechblog.com/rad-outlier-detection-on-big-data-d6b0494371cc.

[20] Guansong Pang, Kai Ming Ting, and David Albrecht. Lesinn: Detecting anomalies by identifying least similar nearest neighbours. In *Proceedings of the ICDMW International Conference on Data Mining Workshop*, pages 623–630. IEEE, 2015.

[21] John Paparrizos and Luis Gravano. k-shape: Efficient and accurate clustering of time series. In *Proceedings of the SIGMOD International Conference on Management of Data*, pages 1855–1870. ACM, 2015.

[22] Daehyung Park, Yuuna Hoshi, and Charles C Kemp. A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder. *Robotics and Automation Letters*, 3(3):1544–1551, 2018.

[23] Daehyung Park, Hokeun Kim, Yuuna Hoshi, Zackory Erickson, Ariel Kapusta, and Charles C Kemp. A multimodal execution monitor with anomaly classification for robot-assisted feeding. In *Proceedings of the IROS International Conference on Intelligent Robots and Systems*, pages 5406–5413. IEEE, 2017.

[24] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. Time-series anomaly detection service at microsoft. In *Proceedings of the 25th SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 3009–3017. ACM, 2019.

[25] Zhilei Ren, Changlin Liu, Xusheng Xiao, He Jiang, and Tao Xie. Root cause localization for unreproducible builds via causality analysis over system call tracing. In *Proceedings of the 34th ASE International Conference on Automated Software Engineering*, pages 527–538. IEEE/ACM, 2019.

[26] Sikandar Samar, Dimitry Gorinevsky, and Stephen Boyd. Likelihood bounds for constrained estimation with uncertainty. In *Proceedings of the 44th Conference on Decision and Control*, pages 5704–5709. IEEE, 2005.

[27] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouet. Anomaly detection in streams with extreme value theory. In *Proceedings of the 23rd SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1067–1075. ACM, 2017.

[28] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2828–2837. ACM, 2019.

[29] Charles F Van Loan. The ubiquitous kronecker product. *Journal of computational and applied mathematics*, 123(1-2):85–100, 2000.

[30] Shashaank Vattikuti, James J Lee, Christopher C Chang, Stephen DH Hsu, and Carson C Chow. Applying compressed sensing to genome-wide association studies. *GigaScience*, 3(1):2047–217X, 2014.

[31] Tom Warren. Zoom grows to 300 million meeting participants despite security backlash. https://www.theverge.com/2020/4/23/21232401/zoom-300-million-users-growth-coronavirus-pandemic-security-privacy-concerns-response.

[32] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 26th WWW World Wide Web Conference*, pages 187–196, 2018.

[33] Yong Xu, Kaixin Sui, Randolph Yao, Hongyu Zhang, Qingwei Lin, Yingnong Dang, Peng Li, Keceng Jiang, Wenchi Zhang, Jian-Guang Lou, et al. Improving service availability of cloud systems by predicting disk error. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 481–494, 2018.

[34] Chuxu Zhang, Dongjin Song, Yuncong Chen, Xinyang Feng, Cristian Lumezanu, Wei Cheng, Jingchao Ni, Bo Zong, Haifeng Chen, and Nitesh V Chawla. A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1409–1416, 2019.

[35] Shenglin Zhang, Ying Liu, Dan Pei, Yu Chen, Xianping Qu, Shimin Tao, and Zhi Zang. Rapid and robust impact assessment of software changes in large internet-based services. In *Proceedings of the 11th CoNext Conference on Emerging Networking Experiments and Technologies*, pages 1–13. ACM, 2015.

[36] Xu Zhang, Junghyun Kim, Qingwei Lin, Keunhak Lim, Shobhit O Kanaujia, Yong Xu, Kyle Jamieson, Aws Albarghouthi, Si Qin, Michael J Freedman, et al. Cross-dataset time series anomaly detection for cloud systems. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 1063–1076, 2019.

[37] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 27th ESEC/FSE Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 807–817. ACM, 2019.

[38] Nengwen Zhao, Jing Zhu, Yao Wang, Minghua Ma, Wenchi Zhang, Dapeng Liu, Ming Zhang, and Dan Pei. Automatic and generic periodicity adaptation for kpi anomaly detection. *Transactions on Network and Service Management*, 16(3):1170–1183, 2019.