Adaptive Performance Anomaly Detection for Online Service Systems via Pattern Sketching

Zhuangbin Chen
The Chinese University of Hong Kong
Hong Kong, China

Jinyang Liu
The Chinese University of Hong Kong
Hong Kong, China

Yuxin Su*
School of Software Engineering
Sun Yat-sen University
Zhuhai, China

Hongyu Zhang
The University of Newcastle
NSW, Australia

Xiao Ling
Yongqiang Yang
Huawei Cloud BU
Beijing, China

Michael R. Lyu
The Chinese University of Hong Kong
Hong Kong, China

ABSTRACT

To ensure the performance of online service systems, their status is closely monitored with various software and system metrics. Performance anomalies represent the performance degradation issues (e.g., slow response) of the service systems. When performing anomaly detection over the metrics, existing methods often lack the merit of interpretability, which is vital for engineers and analysts to take remediation actions. Moreover, they are unable to effectively accommodate the ever-changing services in an online fashion. To address these limitations, in this paper, we propose ADSketch, an interpretable and adaptive performance anomaly detection approach based on pattern sketching. ADSketch achieves interpretability by identifying groups of anomalous metric patterns, which represent particular types of performance issues. The underlying issues can then be immediately recognized if similar patterns emerge again. In addition, an adaptive learning algorithm is designed to embrace unprecedented patterns induced by service updates or user behavior changes. The proposed approach is evaluated with public data as well as industrial data collected from a representative online service system in Huawei Cloud. The experimental results show that ADSketch outperforms state-of-the-art approaches by a significant margin, and demonstrate the effectiveness of the online algorithm in new pattern discovery. Furthermore, our approach has been successfully deployed in industrial practice.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**; **Reliability**; **Maintainability and maintenance**.

KEYWORDS

Cloud computing, performance anomaly detection, online learning

*Corresponding author (suyx35@mail.sysu.edu.cn).

ICSE '22, May 21–29, 2022, Pittsburgh, PA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9221-1/22/05…$15.00
https://doi.org/10.1145/3510003.3510085

ACM Reference Format:
Zhuangbin Chen, Jinyang Liu, Yuxin Su, Hongyu Zhang, Xiao Ling, Yongqiang Yang, and Michael R. Lyu. 2022. Adaptive Performance Anomaly Detection for Online Service Systems via Pattern Sketching. In *44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3510003.3510085

1 INTRODUCTION

With the emergence of cloud computing, many traditional software systems have been migrated to cloud computing platforms as online services. Similar to conventional shrink-wrapped software, the performance of online service systems is an important quality attribute. As online services need to serve millions of customers worldwide, a short period of performance degradation could lead to economic loss and user dissatisfaction. Therefore, proactive and even adaptive system troubleshooting has become the core competence of online service providers. Enterprises that have promoted the automation of system troubleshooting have already received real gains in reliability, efficiency, and agility [6, 7, 21].

In industrial scenarios, online service systems are closely monitored with various metrics (e.g., the CPU usage of an application, service response delay) on a 24×7 basis. This is because the monitoring metrics often serve as the most direct and fine-grained signals that flag the occurrence of service performance issues. In addition, they provide informative clues for engineers to pinpoint the root causes. However, due to the large scale and complexity of online service systems, the number of metrics is overwhelming the existing troubleshooting systems [6]. Automated anomaly detection over the metrics, which aims to discover the unexpected or rare behaviors of the metric time series, is therefore an important means to ensure the reliability and availability of service systems.

Although many efforts, e.g., [13, 32, 37], have been devoted to performance anomaly detection, most of the existing work does not possess the merit of interpretability. Specifically, at each timestamp, they calculate a probability indicating the likelihood of performance anomalies. A threshold is then chosen to convert the probability into a binary label – normal vs. anomaly. However, in reality, a simple recommendation of the suspicious anomalies might not be of much interest to engineers. This is because they need to manually investigate the problematic metrics (recommended by the model) for fault localization. For large-scale online services, this process is like finding a needle in a haystack. The problem is compounded by the fact that false alerts are not rare. Moreover, many state-of-the-art methods train models with historical metric data in an offline setting. As online services continuously undergo feature upgrades and system renewal, the patterns of metrics may evolve accordingly, i.e., concept drift [11, 12]. Without adaptability, these models are unable to accommodate the ever-changing services and user behaviors.

In this paper, we propose ADSketch, a performance anomaly detection approach for online service systems based on *pattern sketching*, which is interpretable and adaptive. The main idea is to identify discriminative subsequences from metric time series that can represent classes of service performance issues. This is similar to the problem of shapelet discovery in time series data [28, 36]. Particularly, for multiple subsequences that describe the same type of performance issue, we take the average of them and regard the result as a *metric pattern* for the issue. For example, services may be experiencing performance degradation when we observe a level shift down on Service Throughput or a level shift up on CPU Utilization. The advantages of such metric patterns are twofold. First, the normality of the incoming metric subsequences can be quickly determined through a comparison with the metric patterns. Second, by associating the patterns with typical anomaly symptoms, we can immediately understand the ongoing performance issues when the metric subsequences exhibit known patterns. This is similar to failure/issue profiling [18, 22, 27]. In this way, ADSketch provides a novel mechanism to characterize service performance issues with metric time series. Previous work on failure/issue profiling often requires handcrafted features, which suffers from limited generalization. For example, Brandon et al. [4] manually defined a set of features collected from metrics, logs, and anomalies to characterize failures. Pattern sketching with metrics enjoys the advantages of automation and accuracy. Moreover, ADSketch is able to adaptively embrace new anomalous patterns when detecting anomalies on the fly. Experimental results demonstrate the superiority of our design over the existing state-of-the-art time series anomaly detectors on both public and industrial data. In particular, we have achieved an average F1 score of over 0.8 in production systems.

To sum up, this work makes the following major contributions:

- We propose ADSketch, an interpretable and adaptive approach for service performance anomaly detection. ADSketch offers a way to characterize service performance issues with monitoring metrics. Different from the existing work, ADSketch is able to provide explanations (e.g., the type of the underlying performance issues) for its prediction results and accept new patterns on the fly. The implementation of ADSketch and datasets are publicly available on GitHub[1].
- We conduct experiments with public data as well as industrial service metric data collected from Huawei Cloud. The experimental results demonstrate the effectiveness of ADSketch in terms of both anomaly detection and adaptive metric pattern learning. Furthermore, our framework has been successfully incorporated into the service performance monitoring system of Huawei Cloud. Our industrial practice confirms its practical usefulness.

[1]https://github.com/OpsPAI/ADSketch

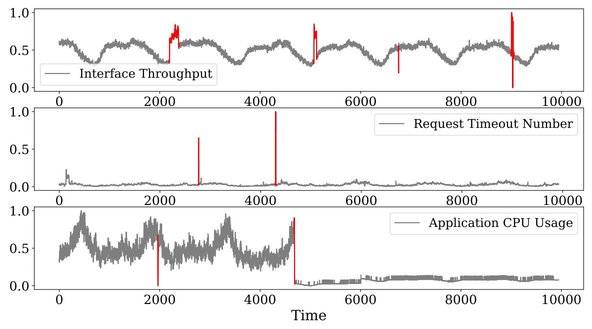

Figure 1: Examples of performance anomaly patterns

2 BACKGROUND & PROBLEM STATEMENT

2.1 Performance Anomaly Patterns in Online Service Systems

In online service systems, a large number of metrics are configured to monitor various aspects of both logical resources (e.g., a virtual machine) and physical resources (e.g., a computing server). Cloud systems often possess an abundance of redundant components, providing the ability of fault tolerance and self-healing (e.g., load balancing, availability zones). Consequently, the majority of service breakdowns tend to manifest themselves as performance anomalies first instead of fail-stop failures [14, 20]. We observe when performance anomalies of similar types happen, their impacts tend to trigger similar reactions/symptoms on the metric time series, which we refer to as metric patterns. For example, a level shift up on Interface Throughput may indicate slow service response, which could be caused by a load balancing failure; a level shift down on it may suggest service unavailability, and the culprit could be performance bugs (e.g., memory leak bugs). Similar observations have been made in [8, 22]. The rationale behind such a phenomenon is twofold. First, the design of the metrics is sophisticated and fine-grained, each of which is dedicated to monitoring a specific problem, e.g., request timeout, high API error rate. Second, cloud systems widely employ the microservices architecture, where cloud applications employ lightweight container deployment, e.g., cloud-native applications, serverless computing. With this architecture, each microservice is designated for well-defined and modularized jobs, e.g., user login, location service. Thus, they tend to develop individual and stable patterns, which can manifest through their monitoring metrics.

2.2 Metric Pattern Mining

Metric patterns (i.e., time-series subsequences describing the misbehaving moments of metrics) can be leveraged to sketch the performance issues for anomaly detection. This is essentially profiling the mode of recurrent anomalies. For example, hardware failures often come with a sudden drop in the corresponding metrics, and the value remains zero for some time. If anomalies come into existence, they can be immediately identified by matching the established patterns. Such metric patterns can also facilitate problem mitigation. For example, when low service throughput and high CPU usage are detected, engineers can scale up the microservice (by adding local cores) to increase its capacity. The key challenge is how to automatically discover what anomalous patterns a metric

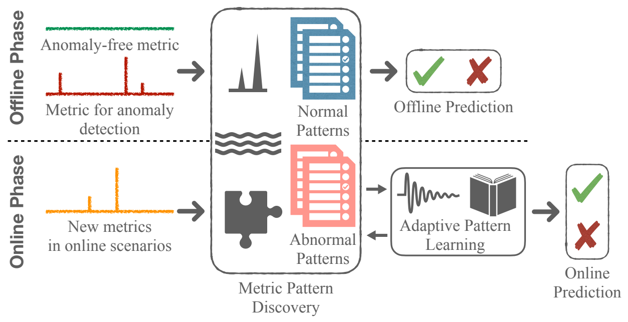

Figure 2: The Overall Framework of ADSketch

Table 1: Summary of Variables

Variable	Meaning
$\mathcal{T}_n$	An anomaly-free metric time series
$\mathcal{T}_a$	An input metric time series for anomaly detection
t	A subsequence of metric time series
m	The length of the metric subsequence t
p	The percentile threshold to find deviated subseqs
$\mathcal{P}_n$	The index set of normal metric patterns
$\mathcal{P}_a$	The index set of anomalous metric patterns
μ_C	The vector of cluster mean vectors
$\mathcal{S}_C$	The vector of cluster sizes
$\mathcal{R}_C$	The vector of cluster radii

time series has experienced. For each identified pattern, engineers can label the typical performance issues it often associates with. In online scenarios, if a metric encounters any known anomalous patterns, the underlying performance issues can be recommended. Pattern sketching therefore provides a means to accumulate and utilize engineers' knowledge.

In real-world scenarios, the patterns exhibited in metrics are extremely complicated and can have numerous variants in terms of scale, length, and combination. Particularly, we have identified the following challenges for metric pattern discovery, which are illustrated in Fig. 1. Each metric time series records around one week of monitoring data, whose anomalies are shown in red.

Background noise. Although a large amount of metric time series is generated, a significant portion of them is trivial, which only records plain system runtime behaviors. Moreover, due to the dynamics of online services, some metrics may experience concept drift [11]. For example, the Application CPU Usage in Fig. 1 drops abruptly, which could be caused by a role switch (e.g., from a primary node to a backup node) or user behavior change. How to distinguish anomalous patterns from normal ones is non-trivial.

Pattern variety. A metric curve can possess multiple distinct patterns simultaneously. For example, in Fig. 1, the Interface Throughput has two anomaly patterns, i.e., spike up and spike down. Also, the patterns can have different scales, as indicated by the two spikes in the Request Timeout Number. We need to consider the context of each metric for pattern extraction.

Varying anomaly duration. Different performance issues may vary in duration. The first two anomalies in the Interface Throughput constitute such an example. Particularly, how long an anomaly lasts is also an important factor that engineers rely on to understand a service's health state. When characterizing the issues, such a fact should be properly considered.

2.3 Problem Statement

The goal of this work is to detect performance anomalies for modern software systems, especially online service systems, based on monitoring metrics. To facilitate issue understanding and problem mitigation, we intend to improve the interpretability of the detection results. To this end, we propose to sketch performance issues with metrics based on our observation that similar issues often exhibit alike patterns. By extracting such anomalous metric patterns, we can conduct performance anomaly detection by examining whether the incoming metric subsequences match the known patterns. Moreover, by associating the extracted metric patterns to specific performance issues, we can obtain a quick understanding of the ongoing issues in online scenarios. Additionally, as online services are continuously evolving, unprecedented metric patterns may emerge. Thus, our algorithm should be adaptive to the new patterns. The problem can be formally defined as follows.

The input of a metric time series can be represented as $\mathcal{T} \in \mathbb{R}^l = [t_1, t_2, ..., t_l]$, where l is the number of observations. $t_i^m = [t_i, ..., t_{i+m-1}]$ is a consecutive subsequence of $\mathcal{T}$ starting from t_i with length m, where $i \in [0, l-m]$. The objective of performance anomaly detection is to determine whether or not a given t_i^m is anomalous, i.e., whether there are performance issues happening from timestamp i to $i+m-1$. Particularly, we also try to explain the type of performance issues associated with t_i^m. The anomalous subsequences will be used to construct abnormal metric patterns, while the benign ones will be regarded as normal patterns. Both the normal and abnormal metric patterns will be updated as the anomaly detection proceeds.

3 METHODOLOGY

3.1 Overview

In online service systems, performance anomalies often serve as the (early) signals for critical failures, which should be detected effectively. However, accuracy alone is far from satisfactory, as it will be labor-intensive to manually investigate the problematic metrics for issue understanding. ADSketch facilitates this process by providing prompt anomaly alerts with explanations.

The overall framework of ADSketch is shown in Fig. 2, which consists of two phases, namely, *offline anomaly detection* and *online anomaly detection*. In the offline phase, ADSketch takes as input a pair of metric time series. One metric time series is anomaly-free, which serves as the basis to detect anomalies in the other metric (if any). In this process, a set of metric patterns will be automatically learned. A metric pattern is essentially the mean of a set of similar metric subsequences representing similar service behaviors. The identified metric patterns are divided into two types, i.e., normal and abnormal. The abnormal patterns often characterize some particular types of performance issues, as discussed in Sec. 2.1. Thus,

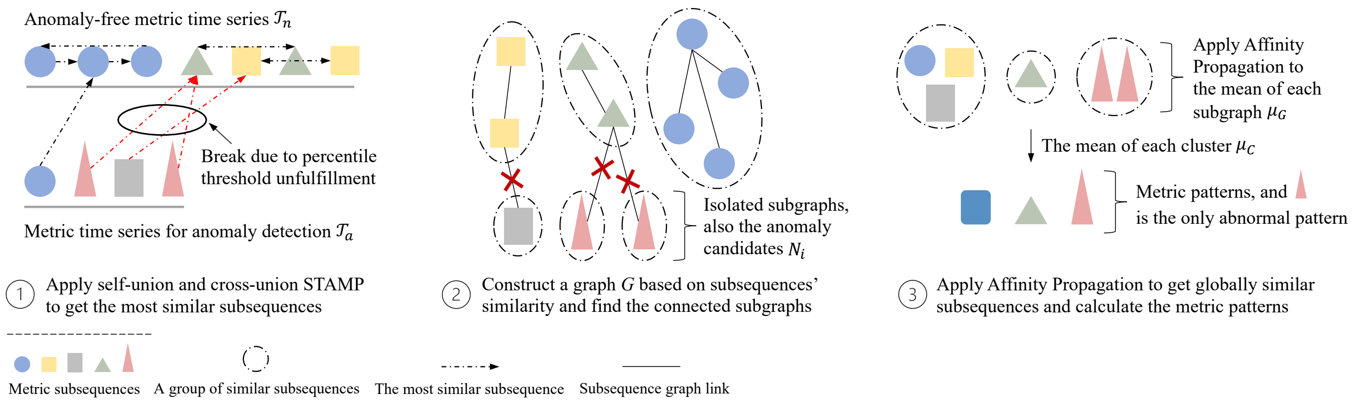
Anomaly-free metric time series $\mathcal{T}_n$
Break due to percentile threshold unfulfillment
Metric time series for anomaly detection $\mathcal{T}_a$
Isolated subgraphs, also the anomaly candidates N_i
Apply Affinity Propagation to the mean of each subgraph μ_G
The mean of each cluster μ_C
Metric patterns, and is the only abnormal pattern
① Apply self-union and cross-union STAMP to get the most similar subsequences
② Construct a graph G based on subsequences' similarity and find the connected subgraphs
③ Apply Affinity Propagation to get globally similar subsequences and calculate the metric patterns
Metric subsequences
A group of similar subsequences
The most similar subsequence
Subsequence graph link

Figure 3: The Algorithm of Performance Anomaly Pattern Discovery

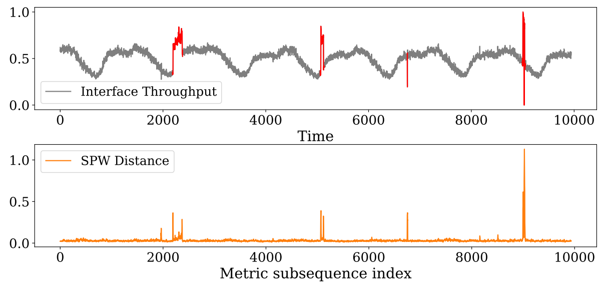

Figure 4: The SPW distance of different metric subsequences

by investing manual efforts to link them to the corresponding issues, a clearer picture of the underlying problems can be easily obtained if similar patterns are encountered again. In the online phase, we leverage the metric patterns built in the offline phase to conduct anomaly detection in online scenarios, where metrics arrive in streams. Particularly, in production environments, unprecedented patterns could appear. Thus, we design an adaptive learning algorithm to capture the new patterns continuously.

Before formally introducing our algorithms, we have summarized the variables involved in Table 1.

3.2 Offline Anomaly Detection

3.2.1 Metric Pattern Discovery. The idea for discovering the abnormal patterns follows the basic definition of an anomaly: if a metric subsequence deviates significantly from those collected during a service's normal executions, it is likely that the subsequence captures some misbehaving moments of the service. To measure how deviated a metric subsequence is, we calculate its distance to other subsequences and search for the smallest distance score. Intuitively, metric subsequences which have large scores to others tend to be anomalous. The function for distance measure is customizable, and we adopt Euclidean distance in this paper.

Given a metric time series with l observations, the number of all possible subsequences is $l - m + 1$, where m is the length of its subsequences. A naïve solution for calculating the smallest pair-wise distance (which we refer to as *SPW distance* hereafter) would be brute force searching. However, this algorithm owns a quadratic time complexity, which is practically infeasible for large time series. Fortunately, some novel scalable algorithms [35, 36, 38] have been proposed in the literature to attack such all-pairs-similarity-search problems for time series subsequences. Particularly, Yeh et al. [36] proposed STAMP, which has achieved orders of magnitude faster compared to state-of-the-art methods. For exceptionally large datasets, an ultra-fast approximate solution is also provided. An illustrating example is provided in Fig. 4, where we can see the misbehaving metric subsequences have larger SPW distances. In particular, the original STAMP algorithm adopts z-normalization for data preprocessing. However, we found min-max normalization yields more meaningful results in our scenario. For a subsequence t_i^m in a metric time series $\mathcal{T}$, we record the index and distance score of another subsequence having the SPW distance to it. Such index and score of all subsequences, i.e., t_i^m $(i \in [0, l-m])$, constitute two vectors $\mathcal{I}$ and $\mathcal{S}$. In particular, for t_i^m, its closest subsequence can either come from the same time series (i.e., self-union) or another time series (i.e., cross-union). In the first case, a trivial match region around t_i^m will be excluded to avoid self matches [36].

The proposed algorithm for metric pattern discovery is presented in Algorithm 1, which is illustrated in Fig. 3. Algorithm 1 takes as input two metric time series, i.e., $\mathcal{T}_n$ and $\mathcal{T}_a$ ($\mathcal{T}_n$ is anomaly-free and $\mathcal{T}_a$ may contain anomalies to be detected), and two hyper-parameters, i.e., m and p (m is the length of subsequences and p is the percentile threshold to find the deviated subsequences). As production service systems are mostly running in normal status [6], the anomaly-free input is easily obtainable (we discuss how we address the violating cases in Sec. 5.3). In line 1 of Algorithm 1, we apply STAMP to $\mathcal{T}_n$ with *self-union* (i.e., similar subsequences come from $\mathcal{T}_n$), and obtain the index and score vectors $\mathcal{I}_{nn}$ and $\mathcal{S}_{nn}$. In line 2, we search similar subsequences for $\mathcal{T}_a$ from $\mathcal{T}_n$, i.e., *cross-union*, and get $\mathcal{I}_{na}$ and $\mathcal{S}_{na}$. Intuitively, given the fact that $\mathcal{T}_n$ is anomaly-free, subsequences in $\mathcal{T}_a$ having large SPW distances to their closest peers in $\mathcal{T}_n$ are suspected to be anomalous. Interestingly, we later learn that Mercer et al. [23] proposed a similar idea concurrently. We introduce a percentile threshold (i.e., p) on $\mathcal{S}_{na}$ to find such deviated subsequences. In particular, p is loosely set to avoid missing anomalies, i.e., false negatives. Such a setting will inevitably produce false positives. We next discuss how we alleviate this issue.

Algorithm 1: Performance Anomaly Pattern Discovery

```
Input: $\mathcal{T}_n$, $\mathcal{T}_a$, $m$, and $p$
Output: Two disjoint sets of $\mathcal{P}_n$ and $\mathcal{P}_a$
1  $\mathcal{I}_{nn}, \mathcal{S}_{nn} \leftarrow$ STAMP($\mathcal{T}_n, \mathcal{T}_n, m$)
2  $\mathcal{I}_{na}, \mathcal{S}_{na} \leftarrow$ STAMP($\mathcal{T}_n, \mathcal{T}_a, m$)
3  $G \leftarrow$ ConnectedSubgraphs($\mathcal{I}_{nn} + \mathcal{I}_{na}, \mathcal{S}_{na}, p$)
4  $N_i \leftarrow$ IsolatedNodes($G$)
5  $\mu_G \leftarrow$ GraphWiseMean($G$)
6  $C \leftarrow$ AffinityPropagation($\mu_G$)
7  $\mu_C \leftarrow$ ClusterWiseMean($C$)
8  $\mathcal{P}_n \leftarrow$ EmptyArray, $\mathcal{P}_a \leftarrow$ EmptyArray
9  for each idx in 1 : Size($C$) do
       // $C[idx]$: all subsequences in the cluster
10     if $C[idx] \subset N_i$ then
11         $\mathcal{P}_a \leftarrow$ Append $\mathcal{P}_a$ with idx
12     else
13         $\mathcal{P}_n \leftarrow$ Append $\mathcal{P}_n$ with idx
14     end
15 end
```

Algorithm 2: Performance Anomaly Detection

```
Input: $t$, $\mathcal{P}_a$, and $\mu_C$
Output: Anomaly detection result for $t$
1  $\mathcal{D}_t \leftarrow$ PairWiseDistance($t, \mu_C$)
2  $idx \leftarrow$ MinIndex($\mathcal{D}_t$)
3  if $idx \in \mathcal{P}_a$ then
4      return True
5  else
6      return False
7  end
```

A metric pattern is defined as the mean of a group of similar subsequences, which represents some typical behaviors of the metric time series. To mine similar subsequences, we propose to leverage their similarity connections. Specifically, in line 3, we construct a graph G whose nodes correspond to the subsequences. Two nodes will be linked if any one of them is deemed as the most similar subsequence to the other, as indicated by $\mathcal{I}_{nn}$ and $\mathcal{I}_{na}$. Note such a relationship is not mutual, i.e., t_i^m is the most similar to t_j^m does not necessarily imply the opposite case. We break the edges whose distance score fails to meet the threshold requirement p. The above operations are depicted in the first part of Fig. 3. Next, we find the connected subgraphs of G, each of which is composed of subsequences resembling each other. Particularly, there will be some isolated nodes, i.e., subgraphs with a single node, which are collected at line 4. Such deviated subsequences constitute a set of anomaly candidates, i.e., N_i. The second part of Fig. 3 illustrates this process.

Up to this point, we have divided the subsequences of $\mathcal{T}_n$ and $\mathcal{T}_a$ into different parts, each of which is represented as a subgraph. However, each subgraph cannot be directly regarded as a metric pattern because: 1) the graph construction criteria can be too strict (i.e., only the most similar pairs are connected), so some subgraphs might still be similar; 2) the loosely set percentile threshold p may flag some normal subsequences as abnormal (i.e., false positives). To further combine the similar subsequences, we apply the Affinity Propagation algorithm [10] to cluster the mean vector of each subgroup (line 5-6). We choose this algorithm because of its superior performance and efficiency, and it requires no pre-defined cluster number. As a result, similar normal subgraphs can be merged together, and abnormal subgraphs have a chance to embrace their normal communities. Thus, each cluster will contain all similar subsequences across the two time-series inputs and different clusters represent distinct patterns. The mean of clusters (i.e., μ_C) will form the set of metric patterns (line 7). For each cluster, we check whether or not all its members come from the set of anomaly candidates N_i (line 9-15). If yes, the mean of the cluster will be regarded as an abnormal metric pattern and otherwise normal, indexed by $\mathcal{P}_a$ and $\mathcal{P}_n$, respectively. The third part of Fig. 3 presents the above operations. Finally, all subsequences in the anomalous clusters will be predicted as an anomaly to be the output of this phase.

3.2.2 Metric Pattern Interpretability. In this section, we expound on how to label the performance issues that each metric pattern represents. By allowing metric patterns to have semantics, the understanding and mitigation of service problems can be greatly accelerated. Given the fact that the duration of different performance issues may vary, our fixed-length metric patterns may over-represent (i.e., the metric pattern is much larger than the issue's duration) or under-represent (i.e., the metric pattern is only an excerpt of the issue) the corresponding issues. To alleviate the first problem, we select a relatively small m, which turns out to be aligned with the goal of better performance. For the second problem, we adopt the following strategy to group clusters which are actually describing a common issue. For each pair of clusters, we check whether they have some subsequences that share some parts in common. All clusters sharing such overlaps together can recover the complete picture of the issue. Thus, we regard them as describing an identical issue. Finally, for each metric pattern, domain engineers will label the type of performance issue that triggers it. Particularly, one pattern can have multiple labels simultaneously. The metric patterns with overlaps will share the same set of performance issue labels.

3.3 Online Anomaly Detection

3.3.1 Anomaly Detection on the Fly. Based on the metric patterns identified in Algorithm 1, we now describe our algorithm (Algorithm 2) for anomaly detection in online scenarios. The idea is straightforward: given a new metric subsequence t with length m, we search for its most similar metric pattern (line 1-2) and check which pattern pool it comes from. If t is more similar to an abnormal pattern, it will be predicted as anomalous; otherwise, normal (line 3-7). In real-world systems where monitoring metrics are generated in a stream manner, this process is continuously running for all coming subsequences. When an anomaly is identified, we would like to provide more interpretation about it, e.g., what kinds of performance issues have happened. This is done by simply recommending the issues associated with the most similar metric pattern for all involved metrics. Particularly, in Algorithm 1, each cluster (i.e., C at line 6) contains all subsequences that are deemed as similar. The design of our online anomaly detection only requires the

Algorithm 3: Adaptive Pattern Learning

```
Input: $t$, $\mathcal{P}_n$, $\mathcal{P}_a$, $\mu_C$, $\mathcal{S}_C$, and $\mathcal{R}_C$
Output: Updated variables: $\mathcal{P}_n$, $\mathcal{P}_a$, $\mu_C$, $\mathcal{S}_C$, and $\mathcal{R}_C$
1  $\mathcal{D}_t \leftarrow$ PairWiseDistance$(t, \mu_C)$
2  $idx \leftarrow$ MinIndex$(\mathcal{D}_t)$
3  $\mu' \leftarrow (\mu_G[idx] \times \mathcal{S}_C[idx] + t)/(\mathcal{S}_C[idx] + 1)$
4  $d_w \leftarrow$ Distance$(\mu_C[idx], \mu') + \mathcal{R}_C[idx]$
5  $d_t \leftarrow$ Distance$(t, \mu')$
6  $d' \leftarrow$ Max$(d_t, d_w)$
7  $d_n, d_a \leftarrow$ Max$(\mathcal{R}_C[\mathcal{P}_n])$, Max$(\mathcal{R}_C[\mathcal{P}_a])$
8  if $idx \in \mathcal{P}_a$ then $d \leftarrow d_a$ else $d \leftarrow d_n$ end
9  if $\mathcal{D}_t[idx] < d$ then
       // add $t$ to the most similar cluster
10     $\mu_C[idx], \mathcal{S}_C[idx], \mathcal{R}_C[idx] \leftarrow \mu', \mathcal{S}_C[idx] + 1, d'$
11     if $\mathcal{S}_C[idx] >$ Max$(\mathcal{S}_C[\mathcal{P}_a])$ and idx is a new cluster
        then
12         $\mathcal{P}_n \leftarrow$ Append $\mathcal{P}_n$ with $idx$
13         $\mathcal{P}_a \leftarrow$ Remove $idx$ from $\mathcal{P}_a$
14     else
15         $d \leftarrow$ Max$(d, d')$ // $d$ will be assigned to $d_n$
               or $d_a$ accordingly
16     end
17 else
       // create a new anomalous cluster for $t$
18     $\mathcal{P}_a \leftarrow$ Append $\mathcal{P}_a$ with Length$(\mu_C) + 1$
19     $\mu_G \leftarrow$ Append $\mu_G$ with $t$
20     $\mathcal{R}_C \leftarrow$ Append $\mathcal{R}_C$ with 0
21     $\mathcal{S}_C \leftarrow$ Append $\mathcal{S}_C$ with 1
22 end
```

mean vector of each cluster, i.e., μ_C. Thus, instead of keeping all its members (which is storage-intensive), the clusters can be simply represented by their mean vectors.

Note that the offline and online anomaly detection can work collaboratively as a performance anomaly detector without the interpretability component, which requires human intervention.

So far the metric patterns for anomaly detection are discovered based on historical data. However, due to the dynamics of online service systems (e.g., software upgrade, customer behavior change), the metrics may experience concept drift [11, 12], which produces brand-new patterns. Thus, an adaptive learning mechanism is desirable to help adapt to such unprecedented patterns and update the metric patterns accordingly. In the next section, we will introduce the algorithm to this end called adaptive pattern learning.

3.3.2 *Adaptive Pattern Learning.* The algorithm of adaptive pattern learning is presented in Algorithm 3, which automatically updates metric patterns during streaming anomaly detection. To start with, for each cluster, we calculate its size and the maximum distance between its mean vector and all members (which we refer to as *radius*), denoted as $\mathcal{S}_C$ and $\mathcal{R}_C$, respectively. In particular, the size and radius of clusters with only a single member are one and zero. For adaptive pattern learning, all clusters can be sufficiently represented with the following properties: μ_C, $\mathcal{S}_C$, and $\mathcal{R}_C$. All subsequences can be discarded.

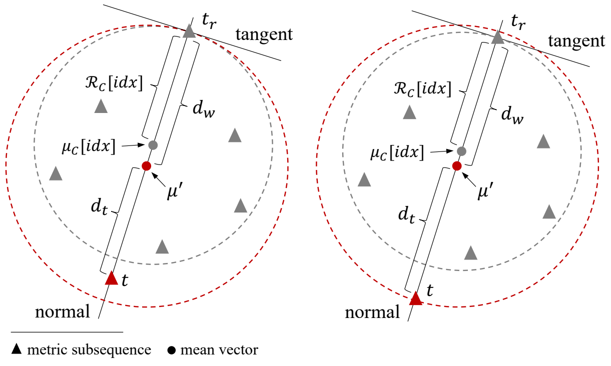

Figure 5: The update of the radius of a cluster

The main idea is that given a new subsequence t, we determine whether it possesses a known metric pattern carried by an existing cluster. If yes, the cluster will absorb t as a new member and update its properties; otherwise, a brand-new anomalous cluster with only t itself will be created, representing an unseen metric pattern. Specifically, we first search for the closest pattern of t (line 1-2). Then, we determine whether t should become a new member to the corresponding cluster by checking if the distance $\mathcal{D}_t[idx]$ is smaller than the largest radius recorded in all clusters, i.e., $\mathcal{D}_t[idx] \leq$ Max$(\mathcal{R}_C)$. If it is the case, t should be considered as an old pattern; otherwise, it should be expressing a new pattern.

When a cluster accepts a new member (line 9-16), we need to update its mean vector $\mu_C[idx]$ (i.e., the metric pattern), size $\mathcal{S}_C[idx]$, and radius $\mathcal{R}_C[idx]$. For $\mu_C[idx]$, it can be precisely updated by the equation at line 3 (i.e., μ'). $\mathcal{S}_C[idx]$ can be trivially updated by increasing itself by one. The update of the radius $\mathcal{R}_C[idx]$ is a bit problematic. We cannot directly calculate the new radius as the original subsequences are not available. To address this problem, we employ the worst-case distance for approximation. As shown in Fig. 5, the new radius reaches its maximum value when t lies in the (inward-pointing) normal of the tangent space at the member yielding the radius (denoted as t_r) [3], which can be calculated by the equation at line 4. We omit the proof, which is standard. Two cases are possible. The first (the left subfigure) is that t_r continues to be the farthest member from the new mean μ'. The second (the right subfigure) is that t takes the place of t_r and becomes the farthest one. Therefore, besides d_w, we also compute the distance between t and μ', i.e., d_t, and compare them (line 4-6). The bigger one will be the new radius (line 10). Recall we need to check if $\mathcal{D}_t[idx] \leq$ Max$(\mathcal{R}_C)$ to decide whether or not t should be taken as a new member. Considering the high imbalance between normal and abnormal clusters, we maintain two maximum radii for them, denoted as d_n and d_a, respectively (line 7). Once a cluster alters its radius, we reset the maximum radius of its kind (d_n or d_a as determined by line 8) if it is exceeded by d' (line 15). On the other hand, if the cluster rejects t, we form a new anomalous cluster containing only t by properly setting its properties (line 18-21).

An issue with this strategy is that false positives will accumulate in $\mathcal{P}_a$ as the unseen patterns can also be normal. We alleviate it by setting a threshold to the size of the newly-formed anomalous clusters (line 11). The role of the cluster will be switched from abnormal to normal if its size exceeds the threshold (line 12-13). The rationale is that performance anomalies are generally rare events. A large anomalous cluster would mean the particular type of issue it represents occurs too often. However, a pattern with a large frequency tends to be the metric's normal behavior. In this paper, we simply set the default threshold as the largest size of the anomalous clusters identified in the offline stage, i.e., $\text{Max}(\mathcal{S}_C[\mathcal{P}_a])$. Nevertheless, more sophisticated strategies can be applied by, for example, considering the distribution of clusters' sizes.

3.4 Time and Space Complexity

3.4.1 Time Complexity. For Algorithm 1, the theoretical time complexity of operation STAMP is $O(n^2)$. Thus, line 1-2 require $O(l_n^2)$ and $O(l_a^2)$, respectively, where l_n and l_a are the length of $\mathcal{T}_n$ and $\mathcal{T}_a$. Another operation with an interesting time complexity is the affinity propagation algorithm (line 7), whose complexity is quadratic in the number of clusters (which is often small), i.e., $O(|C|^2)$. Other operations are of trivial linear time complexity, which is also the case for Algorithm 2 and Algorithm 3. Overall, ADSketch owns a time complexity of $O(n^2)$ ($O(l_n^2 + l_a^2 + |C|^2)$). Fortunately, unlike other models such as deep neural networks, STAMP can be embarrassingly parallelized by distributing its unit operation (SPW distance calculation) to multi-core processors [36]. Moreover, STAMP has an ultra-fast approximation to generate results in an anytime fashion.

3.4.2 Space Complexity. As described in Sec. 3.3.2, pattern clusters have a lightweight representation, i.e., μ_C, $\mathcal{S}_C$, and $\mathcal{R}_C$. We also need $\mathcal{P}_n$ and $\mathcal{P}_a$ to distinguish anomalous patterns from the normal ones. Besides μ_C whose space complexity is $O(m \times |C|)$, other vectors are of $O(|C|)$. Therefore, the dominant term of space complexity is $O(m \times |C|)$. Since both m and $|C|$ are usually small, the space overhead of ADSketch can be considered trivial.

4 EXPERIMENTS

In this section, we evaluate ADSketch using both public data and real-world metric data collected from the industry. Particularly, we aim at answering the following research questions.

RQ1: How effective is ADSketch's offline anomaly detection?

RQ2: How effective is ADSketch's online anomaly detection?

RQ3: How effective is ADSketch's adaptive pattern learning?

The evaluation process of much existing work, e.g., [29, 32], essentially corresponds to the process adopted in RQ1 (i.e., the offline anomaly detection phase), because the threshold they select for anomaly alerting is determined by iterating the full range of its possible values. The best results achieved during the iteration process are reported. To fully examine the performance of different methods in online scenarios, we fix models' data and parameters (including the threshold learned in offline mode) as if they are deployed in production systems, i.e., RQ2. The online adaptability of ADSketch will be evaluated in RQ3.

4.1 Experiment Setting

4.1.1 Dataset. To evaluate the effectiveness of ADSketch in performance anomaly detection, we conduct experiments on two publicly available datasets. Moreover, to confirm its practical significance, we collect a production dataset from a large-scale online service of Huawei Cloud. Table 2 summarizes the statistics of the datasets.

Table 2: Dataset Statistics

Dataset	#Curves	#Points	Anomaly Ratio
Yahoo	67	94,866	1.8%
AIOps18	58	5,922,913	2.26%
Industry	436	4,394,880	1.07%

Public dataset. The public datasets for experiments are Yahoo [30] and AIOps18 [2, 29]. Particularly, we do not conduct online anomaly detection on Yahoo due to its limited number of anomalies.

- *Yahoo.* Yahoo released by Yahoo! Research [30] is a benchmark dataset for time series anomaly detection. Part of the dataset is synthetic (which is simulated by algorithmically injecting anomalies), and part of the dataset is collected from the real traffic of Yahoo services. The anomalies in the real dataset are manually labeled. All time series are sampled every hour. In particular, as our goal is detecting performance anomalies for online services, we only use the real dataset, which reflects the real-world service performance issues. For each time series, we select the first 300 data points as the anomaly-free input (any anomalies are ignored), while the remaining part as the input for offline anomaly detection.
- *AIOps18.* AIOps18 dataset was released by an international AIOps competition held in 2018 [1]. The dataset is composed of multiple metric time series collected from the web services of large-scale IT companies. Particularly, the dataset contains two types of metrics, i.e., service metrics and machine metrics. The service metrics record the scale and performance of the web services, including response time, traffic, connection errors; while the machine metrics reflect the health states of physical machines, including CPU usage, network throughput. Some metric time series has a sampling interval of one minute, while that of others is five minutes. Each metric has a training and a testing time series. Thanks to its large quantity, we follow the following procedure to separate the data for ADSketch offline and online anomaly detection. First, we extract a small part of the training time series that is anomaly-free, which often contains thousands of data points. Then, we use the remainder of the training time series for offline anomaly detection. Finally, the whole testing time series will be employed for online anomaly detection. We also compare the performance of online anomaly detection with and without the adaptive learning component.

Industrial dataset. To evaluate ADSketch in production scenarios, we collect various metrics (e.g., Application CPU Usage, Interface Throughput, Request Timeout Number, Round-trip Delay) from a large-scale online service (we conceal the name for privacy concern) of Huawei Cloud. The system under study produces millions of metric time series, which contain an abundance of different metric patterns. The number of metric curves collected is 436, which come from multiple instances of virtual machines, containers, and applications of the selected service system. For each metric, we collect one week of data with a sampling interval of one minute, resulting in more than four million data points in total. The anomalies representing the performance issues of the service are labeled by experienced domain engineers. From Table 2, we can see that the anomaly ratio is very low. Particularly, we use the first day as the anomaly-free input, whose anomalies (if any) are simply ignored. The next three days are used for offline anomaly detection. Finally, we conduct online anomaly detection on the remaining three days, where we also evaluate the adaptability of different approaches to unseen anomaly patterns.

4.1.2 *Evaluation Metrics.*

As anomaly detection is essentially a binary classification problem, i.e., normal and abnormal, we employ *precision*, *recall*, and *F1 score* for evaluation. They can gauge the performance of an anomaly detection algorithm at a fine-grained level. A satisfactory algorithm should be able to quickly and precisely detect both the occurrence and duration of performance anomalies. Specifically, precision measures the percentage of anomalous metric points that are successfully identified as anomalies over all the metric points that are predicted as anomalous: $precision = \frac{TP}{TP+FP}$. Recall calculates the portion of anomalous metric points that are successfully identified by ADSketch over all the actual anomalous points: $recall = \frac{TP}{TP+FN}$. Finally, the F1 score is the harmonic mean of precision and recall: $F1\ Score = \frac{2\times precision\times recall}{precision+recall}$. TP is the number of anomalous metric points that are correctly discovered by ADSketch; FP is the number of normal metric points that are wrongly predicted as an anomaly by ADSketch; FN is the number of anomalous metric points that ADSketch fails to notice. Since there are multiple metrics in each dataset, we report their average weighted by the size of each metric time series.

4.1.3 *Comparative Methods.*

The following methods are selected for comparative evaluation of ADSketch. As all baselines have open-sourced their code, we directly borrow the implementations and follow the procedure of model training and parameter tuning introduced in each method.

- *LSTM* [15, 37]. This method employs Long Short-Term Memory (LSTM) network to capture the normal behaviors of metrics in a forecasting-based manner. Specifically, it predicts the next values of a metric based on its past observations. The predicted values are then compared with the actual values. Anomaly warnings will be raised if the differences exceed the pre-defined thresholds.
- *Donut* [34]. Donut adopts the Variational Autoencoder (VAE) framework to properly reconstruct the normal metric subsequences. The trained model will have a large reconstruction loss when it meets anomalous instances, which serves as the signal to alert anomalies.
- *LSTM-VAE* [25]. Similar to Donut, this work detects anomalies based on metric subsequence reconstruction. It combines LSTM and VAE in the model design.
- *LODA* [26]. LODA is an online anomaly detector based on the ensemble of a series of one-dimensional histograms. Each histogram approximates the probability density of input data projected onto a single projection vector. LODA calculates the likelihood of an anomaly based on the joint probability of the projections.
- *iForest* [19]. Isolation Forest (iForest) is composed of a collection of isolation trees, which isolates anomalies based on random subsets of the input features. The height of an input sample, averaged over the trees, is a measure of its normality. Samples with noticeably shorter heights are likely to be anomalies. We use metric subsequences as the input samples.
- *DAGMM* [39]. DAGMM utilizes a deep autoencoder to generate a low-dimensional representation for each input data point, which is further fed into a Gaussian Mixture Model to estimate the anomaly score.
- *SR-CNN* [29]. SR-CNN first applies Spectral Residual to highlight the most important regions for seasonal metric data where anomalies often reside. It then trains a Convolutional Neural Network (CNN) through synthetic anomalies to detect the real anomalies.

4.2 Experimental Results

4.2.1 ***RQ1*** *The Effectiveness of ADSketch's Offline Anomaly Detection.*

To answer this research question, we compare ADSketch with the baselines in the offline setting. The results are shown in Table 3, where we can see the average F1 score of ADSketch outperforms all baseline methods in all datasets. In AIOps18 and Industry, the improvement achieved by ADSketch is more significant. In particular, the patterns of anomalies in Yahoo are relatively simple. By iterating over all possible values of the anomaly threshold, the baselines can find the best setting for the dataset under study. Among them, LSTM [15, 37] and Donut [34] achieve comparable performance compared to that of ADSketch (i.e., 0.541), whose average F1 scores are 0.53 and 0.524, respectively. Moreover, LSTM [15, 37] has the best recall (i.e., 0.706), while the best precision (i.e., 0.754) goes to LODA [26]. DAGMM and SR-CNN turn out to be the worst methods in this dataset. In terms of AIOps18 and Industry datasets, we can see ADSketch surpasses the baselines by a larger margin. Specifically, the average F1 score of ADSketch in AIOps18 is 0.677, while that of the second-best method (i.e., LSTM-VAE) is 0.537. ADSketch also attains the best precision and recall. In AIOps18, the anomaly patterns are much more complicated. Baselines tend to predict more data points as anomalous, leading to a lower precision. Different from them, ADSketch is able to precisely capture them and outperforms other methods. The situation is similar in Industry. Particularly, this dataset is collected from online services, and many of its metric curves possess more perceivable and regular patterns. Thus, all methods perform better in this dataset than in the other two. The average F1 scores of ADSketch and the second-best method (i.e., LSTM) are 0.740 and 0.632, respectively.

In Table 3, we can see among all comparative methods, LSTM and LSTM-VAE have better overall performance, which are forecasting-based and reconstruction-based methods, respectively. They both try to model the normal patterns of a metric time series and alert anomalies once the metric significantly deviates from the learned patterns. The difference is that a forecasting-based method aims to predict the next metric values and a reconstruction-based method tries to encode and regenerate metric subsequences. We can see

Table 3: Experimental Results of Offline Anomaly Detection

	Yahoo			AIOps18			Industry		
Method	**precision**	**recall**	**F1 score**	**precision**	**recall**	**F1 score**	**precision**	**recall**	**F1 score**
LSTM	0.598	**0.706**	0.530	0.499	0.531	0.518	0.704	0.656	0.632
LSTM-VAE	0.622	0.634	0.484	0.510	0.625	0.537	0.717	0.639	0.622
Donut	0.530	0.658	0.524	0.405	0.527	0.382	0.693	0.628	0.604
LODA	**0.754**	0.583	0.428	0.553	0.429	0.401	0.583	0.498	0.529
iForest	0.713	0.597	0.437	0.555	0.439	0.413	0.616	0.567	0.538
DAGMM	0.643	0.517	0.401	0.590	0.477	0.461	0.597	0.542	0.530
SR-CNN	0.433	0.618	0.307	0.424	0.387	0.363	0.519	0.471	0.434
ADSketch	0.511	0.673	**0.541**	**0.744**	**0.670**	**0.677**	**0.811**	**0.813**	**0.740**

Table 4: Experimental Results of Online Anomaly Detection

	AIOps18			Industry		
Method	**prec.**	**rec.**	**F1**	**prec.**	**rec.**	**F1**
LSTM	0.425	0.462	0.408	0.612	0.606	0.592
LSTM-VAE	0.336	0.521	0.389	0.624	0.598	0.601
Donut	0.431	0.326	0.376	0.662	0.581	0.590
LODA	0.407	0.397	0.355	0.653	0.526	0.503
iForest	0.397	0.334	0.322	0.576	0.507	0.487
DAGMM	0.392	0.367	0.378	0.557	0.538	0.502
SR-CNN	0.329	0.288	0.307	0.438	0.422	0.410
ADSketch	**0.543**	**0.575**	**0.507**	**0.705**	**0.603**	**0.606**

Table 5: Experimental Results of Adaptive Pattern Learning

	AIOps18			Industry		
Method	**prec.**	**rec.**	**F1**	**prec.**	**rec.**	**F1**
LODA	0.424	0.405	0.387	0.623	0.512	0.548
ADSketch	**0.594**	**0.557**	**0.548**	**0.882**	**0.856**	**0.832**

except for LSTM-VAE in Yahoo, these two methods attain the best results compared to other baseline counterparts in the other two datasets. However, LSTM lacks the ability to explicitly detect anomalies in the level of subsequence. Many anomalies are composed of a collection of anomalous points corresponding to the period of performance issues. LSTM-VAE does not take into account the relationship among subsequences. Many suspicious subsequences are not necessarily anomalies if they often occur in the history of the service systems. Compared to them, ADSketch is able to simultaneously learn the subsequence-level features and consider the context of metric time series.

4.2.2 ***RQ2*** *The Effectiveness of ADSketch's Online Anomaly Detection.* We also compare ADSketch against the selected methods for online anomaly detection. Table 4 presents the experimental results. Except for Donut in AIOps18, all models and algorithms encounter an obvious performance degradation in both datasets. Nevertheless, ADSketch manages to maintain the best ranking (0.507 in AIOps18 and 0.606 in Industry), which is followed by LSTM (0.408 in AIOps18) and LSTM-VAE (0.601 in Industry). Particularly, in AIOps18, the average F1 score of different methods drops by 11%-27%. This observation demonstrates the existence of unprecedented metric patterns in online scenarios. By relying on the "outdated" data and parameters (e.g., ADSketch's metric patterns and baselines' anomaly thresholds) learned from the offline stage, the methods cannot accommodate them. In addition, by plotting the metric time series, we observe the emergence of concept drift on metrics. This can be caused by software upgrades or the integration of new service components (e.g., virtual machines, containers). In the industrial dataset, the evaluation results of the baselines are more promising (i.e., the average F1 score drops by less than 10%). This is because the anomalies are triggered by real-world performance issues. The issues have a more natural distribution, and the collected metrics exhibit relatively stable patterns. ADSketch presents a significant performance degradation. We found it is because in some cases, the two metric time series fed to the offline stage are often both anomaly-free. Consequently, no abnormal patterns will be learned, disabling ADSketch to detect anomalies in the online stage. Therefore, when designing an anomaly detection algorithm, adaptability is indispensable.

4.2.3 ***RQ3*** *The Effectiveness of ADSketch's Adaptive Pattern Learning.* This research question looks into the issue of online adaptability. Particularly, we only compare ADSketch with LODA, which is the only baseline method with the design of online learning. Similar to RQ2, we only conduct experiments with AIOps18 and Industry datasets. Table 5 shows the experimental results, where we can see ADSketch's adaptive pattern learning indeed brings performance gains. With more anomalous patterns identified, ADSketch is able to detect anomalies more accurately, i.e., a better precision (0.594 in AIOps18 and 0.882 in Industry). The average F1 score also enjoys some improvements, i.e., 0.548 in AIOps18 and 0.832 in Industry. Particularly, in the industrial case, adaptive ADSketch achieves a performance of over 0.8 in all evaluation metrics (even in some cases without any abnormal patterns learned from the offline stage). Such an achievement indicates its potential to meet the industrial requirements of performance anomaly detection. On the other hand, the online version of LODA does not show much performance improvement (i.e., an average F1 score of 0.387 in AIOps18 and 0.548

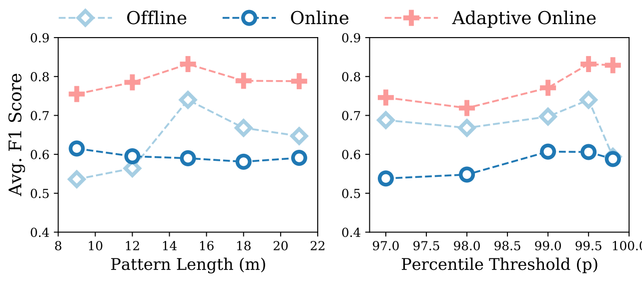

Figure 6: Parameter Sensitivity

in Industry), which even falls behind some methods without the capability of online learning.

4.2.4 Parameter Sensitivity. In ADSketch, there are only two parameters to tune (both in Algorithm 1), i.e., the pattern length m and the percentile threshold p for identifying deviated metric subsequences. We evaluate the sensitivity of ADSketch to these two parameters by conducting experiments with different settings. Due to space limitations, we only show the results of the Industry dataset. The default value of m and p for the dataset is 15 and 99.5th, respectively. We fix one parameter and employ a different setting for the other one. Specifically, m ranges from 9 to 21, and p varies from 97th to 99.8th. Fig. 6 presents the results. Performance degradation is observed in both offline and online stages when the two parameters deviate from their default setting. The offline stage exhibits a greater sensitivity, and thus, less anomalous metric patterns are captured. Nevertheless, both the online anomaly detection and adaptive pattern learning algorithms achieve stable performance with a smaller set of abnormal patterns. This further confirms ADSketch's capability of new pattern discovery.

5 INDUSTRIAL PRACTICE

5.1 Online Deployment

Since October 2020, ADSketch has been successfully incorporated into the performance anomaly detection system of a large-scale online service system in Huawei Cloud. The deployment process can be easily done by leveraging the existing data analytics pipeline, for example, data consumption by Apache Kafka [16], and online parallel execution by Apache Flink [9]. After months of usage, ADSketch has demonstrated its effectiveness on metric-based system troubleshooting. A lot of positive feedback has been received from on-site engineers. Particularly, engineers confirmed its superiority in anomaly detection over the current algorithms (e.g., fixed thresholding, moving average) in operation. One typical case is multiple benign spikes arriving suddenly and consecutively. ADSketch is able to quickly figure out that such recurrent spikes have happened before, which reduces the number of false alerts. In terms of issue understanding, engineers benefited from ADSketch by having readily-available descriptions about the anomaly symptoms. Therefore, we have initialized a project of metric pattern database construction. ADSketch is continuously accumulating anomalous patterns in the database. Moreover, engineers also expressed the need for metric pattern auto-correlation across different metrics. This is because multiple anomalies collectively could constitute a stronger performance issue indicator. We leave the identification of such correlations to our future work.

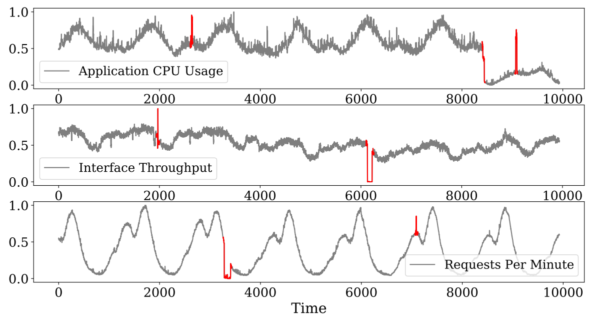

Figure 7: Case Study of ADSketch

5.2 Case Study

We provide some case studies of ADSketch collected from production systems in Fig. 7, where anomalies are indicated by the red lines. Due to space limitations, we only showcase three metric time series. Clearly, all anomalous metric patterns have been successfully located regardless of shape, scale, and length. Each metric time series possesses at least two types of anomalous patterns, e.g., level shifts and spikes. Interestingly, we found the depression in the second metric can help catch a similar pattern in the third metric, demonstrating the feasibility of cross-metric pattern sharing. Moreover, engineers confirmed that these patterns are typical, based on which they can make a good guess about the ongoing issues. For example, the spikes often come from user request surge or network attack; the depressions in the second and third metrics often indicate service restart or link flap. To quantify the interpretability of ADSketch, we label the recurrent performance issues and employ the learned metric patterns to identify them. As performance issues may contain uncertainty [33], we allow one pattern to be associated with multiple labels simultaneously (Sec. 3.2.2). During the evaluation, an anomaly interpretation is considered correct if the predicted performance issue appears in the label set. In our experiments, ADSketch attains a promising F1 score of 0.825. This demonstrates the potentials of ADSketch in providing interpretable results to engineers, which can greatly accelerate the investigation of service performance issues.

5.3 Threats to Validity

We have identified the following major threats to validity.

Internal threats. The implementation and parameter selection are two critical internal threats to the validity. To reduce the implementation threat, we directly borrow the codes released by the baseline approaches. For the proposed approach, we employ peer code review, i.e., the authors are invited to carefully check the implementation for mistakes. In terms of parameter selection, we conduct multiple comparative experiments with different parameters for all methods. We choose the parameter settings empirically based on the best results.

External threats. The selection of the service system and the baselines are two main external threats to validity. We choose a large-scale online service of Huawei Cloud, which produces millions of metrics with diverse patterns. Moreover, we detect anomalies by following the basic definition of an anomaly, i.e., the data point that deviates from the majority in a dataset. Thus, ADSketch is generalizable to other systems. For baselines, we select the representative ones in the literature, covering a wide spectrum of techniques.

Construct threats. The main construct threat to validity is that the anomaly-free input (i.e., $\mathcal{T}_n$) to Algorithm 1 actually contains anomalies. Although anomaly-free data are easily obtainable in reality, false negatives could happen if the data are contaminated. We alleviate this issue by applying percentile thresholding to $\mathcal{T}_n$. Specifically, after obtaining the closest subsequence pairs in $\mathcal{T}_n$, we break the connection between those having a distance above the percentile threshold. Thus, the set of anomaly candidates, i.e., N_i, becomes larger. If $\mathcal{T}_n$ is indeed clean, this operation is harmless as the (isolated) normal metric subsequences can be grouped with other similar ones again; if not, they will stay isolated and eventually be recognized as anomalies. We have also conducted experiments on some cases where $\mathcal{T}_n$ contains anomalies, and the results show its effectiveness.

6 RELATED WORK

Performance anomaly detection on time series has been a hot topic. Monitoring metrics used to profile the runtime status of a system are usually denoted as multiple univariate time series. In the literature, anomaly detection methods on time series can be categorized into statistical, traditional machine learning, and deep learning approaches. In industry, Autoregressive Moving Average Model (ARMA) [5] remains the most popular statistical method to detect obvious anomalous data points from univariate time series. To capture complex anomalous patterns, Ma et al. [22] summarized several type-oriented patterns from the metrics of cloud databases to diagnose the performance degradation in associated online services.

More complex pattern recognition methods utilize machine learning based models. For example, unsupervised clustering methods can be used to detect anomalous points in time-series data. Similar to our work, Pang et al. [24] proposed a clustering-based statistical model called LeSiNN to detect anomaly patterns from history. However, it is not robust in real industry practices due to complicated parameter tuning. With the assumption that anomalous data should be in smaller numbers and isolated from a large number of normal observations, Isolation Forest (iForest) [19] employs multiple binary trees to distinguish anomalies in non-linear space. Extreme Value Theory (EVT) [31] learns the hidden state of a random variable around the tails of its distribution to adaptively enhance the performance of many statistical and machine learning methods. However, EVT heavily relies on hyperparameter tuning.

In recent years, there has been an explosion of interest in applying neural networks to conduct anomaly detection on time-series data. For example, Zong et al. [39] proposed a deep autoencoding Gaussian mixture model (DAGMM) to detect anomalous data points from each observed data without considering the temporal dependencies in time series. To detect complex anomalies in spacecraft monitoring systems, LSTM-NDT [15] leverages Long Short-Term Memory (LSTM) networks with nonparametric dynamic thresholding to pursue interpretability throughout the systems. Zhao et al. [37] and Lin et al. [17] also employed LSTM to predict performance anomalies in software systems. Inspired by the Spectral Residual algorithm in other domains, Ren et al. [29] proposed SR-CNN to detect anomalies from seasonal metric data for large-scale cloud services, which contain the periodic recurrence of fluctuations. DONUT [34] designs an unsupervised anomaly detection method based on the Variational Auto-Encoder (VAE) framework to detect anomalies from low-qualified seasonal metric time series with various patterns. DONUT provides a theoretical explanation compared to other deep learning methods. LSTM-VAE [25] combines LSTM networks and the VAE framework to reconstruct the probability distribution of observed data in time series. However, LSTM-VAE ignores the temporal dependencies in time series. Omni-Anomaly [32] learns the normal patterns using a large collection of historical data. The anomalous patterns are located from the large margin of reconstruction loss to the normal patterns. However, the aforementioned deep learning-based methods usually follow an end-to-end style and play as a black box inside. Due to poor interpretability, the detection results cannot provide engineers with actionable suggestions for fault diagnosis. Furthermore, all these methods have difficulties handling unseen metric patterns brought by the frequent updates of online services.

7 CONCLUSION

In this paper, we propose ADSketch, a performance anomaly detection approach based on pattern sketching. By extracting normal and abnormal patterns from metric time series, anomalies can be quickly detected through a comparison with the identified patterns. By associating metric patterns with typical performance issues, ADSketch can provide interpretable results when any known patterns appear again. Moreover, we design an adaptive learning algorithm to help ADSketch embrace unprecedented metric patterns during online anomaly detection. We have conducted experiments on two public datasets and one production dataset collected from a representative online service system of Huawei Cloud. For offline anomaly detection where models' parameters are still being tuned, ADSketch has achieved the highest F1 score, outperforming the existing methods by a significant margin. For online anomaly detection where models are fixed, ADSketch safeguards its best rankings. Finally, the adaptive pattern learning brings noticeable performance gains, especially in the industrial dataset. From our industrial practice, we have witnessed it shedding light on accurate and interpretable performance anomaly detection, which confirms its practical benefits conveyed to Huawei Cloud. We believe ADSketch is able to assist engineers in service failure understanding and diagnosis.

For future work, we will extend our algorithms to multivariate metric time series. We will also try to provide more detailed information about failures by exploring the correlations among the metric patterns.

ACKNOWLEDGMENTS

The work was supported by the Guangdong Key Research Program (No. 2020B010165002), the Research Grants Council of the Hong Kong Special Administrative Region, China (CUHK 14210920), and Australian Research Council (ARC) Discovery Projects (DP200102940 and DP220103044).

REFERENCES

[1] 2018. KPI Anomaly Detection Competition. Retrieved April, 2021 from http://iops.ai/competition_detail/?competition_id=5&flag=1

[2] 2018. KPI Anomaly Detection Dataset. Retrieved April, 2021 from http://iops.ai/dataset_detail/?id=10

[3] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. 2004. *Convex optimization*. Cambridge university press.

[4] Álvaro Brandón, Marc Solé, Alberto Huélamo, David Solans, María S Pérez, and Victor Muntés-Mulero. 2020. Graph-based root cause analysis for service-oriented and microservice architectures. *Journal of Systems and Software* 159 (2020), 110432.

[5] Marcus J Chambers and Michael A Thornton. 2012. Discrete time representation of continuous time ARMA processes. *Econometric Theory* (2012), 219–238.

[6] Zhuangbin Chen, Yu Kang, Liqun Li, Xu Zhang, Hongyu Zhang, Hui Xu, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, et al. 2020. Towards intelligent incident management: why we need it and how we make it. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1487–1497.

[7] Yingnong Dang, Qingwei Lin, and Peng Huang. 2019. AIOps: real-world challenges and research innovations. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 4–5.

[8] Mostafa Farshchi, Jean-Guy Schneider, Ingo Weber, and John Grundy. 2015. Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis. In *2015 IEEE 26th international symposium on software reliability engineering (ISSRE)*. IEEE, 24–34.

[9] Apache Flink. 2011. [Online]. https://flink.apache.org/.

[10] Brendan J Frey and Delbert Dueck. 2007. Clustering by passing messages between data points. *science* 315, 5814 (2007), 972–976.

[11] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM computing surveys (CSUR)* 46, 4 (2014), 1–37.

[12] Shujie Han, Patrick PC Lee, Zhirong Shen, Cheng He, Yi Liu, and Tao Huang. 2020. Toward adaptive disk failure prediction via stream mining. In *Proceedings of IEEE ICDCS*.

[13] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R Lyu, and Dongmei Zhang. 2018. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 60–70.

[14] Peng Huang, Chuanxiong Guo, Lidong Zhou, Jacob R Lorch, Yingnong Dang, Murali Chintalapati, and Randolph Yao. 2017. Gray failure: The achilles' heel of cloud-scale systems. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. 150–155.

[15] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. 2018. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 387–395.

[16] Apache Kafka. 2011. [Online]. https://kafka.apache.org/.

[17] Qingwei Lin, Ken Hsieh, Yingnong Dang, Hongyu Zhang, Kaixin Sui, Yong Xu, Jian-Guang Lou, Chenggang Li, Youjiang Wu, Randolph Yao, et al. 2018. Predicting node failure in cloud service systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 480–490.

[18] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 102–111.

[19] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *2008 eighth ieee international conference on data mining*. IEEE, 413–422.

[20] Chang Lou, Peng Huang, and Scott Smith. 2020. Understanding, detecting and localizing partial failures in large system software. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 559–574.

[21] Jian-Guang Lou, Qingwei Lin, Rui Ding, Qiang Fu, Dongmei Zhang, and Tao Xie. 2013. Software analytics for incident management of online services: An experience report. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 475–485.

[22] Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, et al. 2020. Diagnosing root causes of intermittent slow queries in cloud databases. *Proceedings of the VLDB Endowment* 13, 10 (2020), 1176–1189.

[23] Ryan Mercer, Sara Alaee, Alireza Abdoli, Shailendra Singh, Amy Murillo, and Eamonn Keogh. 2021. Matrix Profile XXIII: Contrast Profile: A Novel Time Series Primitive that Allows Real World Classification. In *The IEEE International Conference on Data Mining*.

[24] Guansong Pang, Kai Ming Ting, and David W. Albrecht. 2015. LeSiNN: Detecting Anomalies by Identifying Least Similar Nearest Neighbours. In *IEEE International Conference on Data Mining Workshop, ICDMW 2015, Atlantic City, NJ, USA, November 14-17, 2015*. IEEE Computer Society, 623–630.

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

[26] Tomáš Pevnỳ. 2016. Loda: Lightweight on-line detector of anomalies. *Machine Learning* 102, 2 (2016), 275–304.

[27] Andy Podgurski, David Leon, Patrick Francis, Wes Masri, Melinda Minch, Jiayang Sun, and Bin Wang. 2003. Automated support for classifying software failure reports. In *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE, 465–475.

[28] Thanawin Rakthanmanon and Eamonn Keogh. 2013. Fast shapelets: A scalable algorithm for discovering time series shapelets. In *proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM, 668–676.

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

[30] Yahoo! Research. 2015. A Benchmark Dataset for Time Series Anomaly Detection. Retrieved August, 2021 from https://yahooresearch.tumblr.com/post/114590420346/a-benchmark-dataset-for-time-series-anomaly

[31] Alban Siffer, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouët. 2017. Anomaly Detection in Streams with Extreme Value Theory. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*. ACM, 1067–1075.

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

[33] Catia Trubiani, Pooyan Jamshidi, Jurgen Cito, Weiyi Shang, Zhen Ming Jiang, and Markus Borg. 2018. Performance issues? Hey DevOps, mind the uncertainty. *IEEE Software* 36, 2 (2018), 110–117.

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

[35] Dragomir Yankov, Eamonn Keogh, and Umaa Rebbapragada. 2008. Disk aware discord discovery: Finding unusual time series in terabyte sized datasets. *Knowledge and Information Systems* 17, 2 (2008), 241–262.

[36] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. 2016. Matrix profile I: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In *2016 IEEE 16th international conference on data mining (ICDM)*. IEEE, 1317–1322.

[37] Guoliang Zhao, Safwat Hassan, Ying Zou, Derek Truong, and Toby Corbin. 2021. Predicting Performance Anomalies in Software Systems at Run-time. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 3 (2021), 1–33.

[38] Yan Zhu, Chin-Chia Michael Yeh, Zachary Zimmerman, Kaveh Kamgar, and Eamonn Keogh. 2018. Matrix profile XI: SCRIMP++: time series motif discovery at interactive speeds. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 837–846.

[39] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. 2018. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations*.