Edge computing empowered anomaly detection framework with dynamic insertion and deletion schemes on data streams

Haolong Xiang · Xuyun Zhang

Received: 2 November 2021 / Revised: 17 March 2022 / Accepted: 7 April 2022 / Published online: 12 May 2022 © The Author(s) 2022

Abstract
Anomaly detection plays a crucial role in many Internet of Things (IoT) applications such as traffic anomaly detection for smart transportation and medical diagnosis for smart healthcare. With the explosion of IoT data, anomaly detection on data streams raises higher requirements for real-time response and strong robustness on large-scale data arriving at the same time and various application fields. However, existing methods are either slow or application-specific. Inspired by the edge computing and generic anomaly detection technique, we propose an isolation forest based framework with dynamic Insertion and Deletion schemes (IDForest), which can incrementally update the forest to detect anomalies on data streams. Besides, IDForest is deployed on edge servers in parallel through packing each tree into a subtask, which facilitates the fast anomaly detection on data streams. Extensive experiments on both synthetic and real-life datasets demonstrate the efficiency and robustness of our framework for anomaly detection.

Keywords Anomaly detection · Data streams · Large-scale data · Edge computing · Efficiency and robustness

1 Introduction

The application of Internet of Things (IoT) technologies to the smart world has improved life quality and attracted significant attention in academia [4, 28]. With fast development and wide deployment of IoT technologies, the size of the data has exploded, which
comes from various intelligent applications, such as smart city, smart home, smart hospital and smart farm [13, 22]. Large-scale IoT data increase the difficulty to detect, quantify and understand the surrounding environments, where the criminals are more likely to invade [29]. For instance, identifying hacker intrusions in massive network data or detecting anomalous trends in industrial data that indicate a pending system failure requires accurate and fast anomaly detection. In real life applications, these data get sampled over very short time intervals and keep flowing in infinitely leading to data streams, which raise the requirement for real-time response to the abnormal event. Therefore, developing effective and real-time anomaly detection techniques among the data stream with large-scale data should be a research priority [12, 23].

Streams can be a time-series or multidimensional, and the data stream does not have a fixed length compared with the static data [10]. For an infinite data stream, anomaly detection is performed by a sliding window, which confines the data instances within the fixed-size context. As the window slides, the expired data points are removed from the window while an equal number of new data points are added to the window. Besides, the anomalies are detected in each sliding window. For example, monitor the click rate of shopping websites and find the anomalous click times is a typical time-series anomaly detection on data streams. Besides, the real-time cardiac monitoring produces a kind of multidimensional data streams, which collects the medical information from implanted or wearable sensors and transmitted this information to a server for diagnosis [17]. However, these methods are all application-specific, executing anomaly detection on one field of application or one type of data streams. With the increasing of application types, it is a trouble work to design different kinds of anomaly detection methods. So, it is meaningful to design a generic framework for anomaly detection on data streams and improve the robustness on various application fields.

Monitoring on data streams often requires real-time response to the anomalous events, which increases the difficulty to execute efficient anomaly detection on data streams with large-scale data instances. Limited by the capability of resource storage and computation on sensor-equipped devices, these intensive data are offloaded to cloud/edge servers for storage and processing. Since edge servers are closer to devices in geography compared to cloud servers and the resources in the edge servers provide sufficient computing and storage power for data streams, model deployment on edge servers is regarded as a practical method to shorten the processing time on the data stream with massive data. To illustrate the efficiency of edge computing, Mehnaz et al. recently made an experiment and found the processing time in smart devices is around 5 times longer than that in edge servers over a data stream containing 100000 data points [21]. In this case, the windowed Gaussian (W-Gaussian) anomaly detection method is used to detect anomalies. As a statistic-based method, W-Gaussian has good accuracy on the data following a distribution while it may act poorly on the data not belonging to a normal distribution. This work provides us with the idea of combining anomaly detection methods with edge computing. However, it remains a big challenge to build an accurate anomaly detection method over a data stream with complex data and deploy it to edge servers to monitor data in real time.

Considering the distributed characteristics of edge computing, the distributed processing method is feasible to be deployed on edge servers to speed up. Among all types of anomaly detection methods, ensemble-based anomaly detection methods can be broken down into multiple concurrent tasks that can be handled independently. So, we consider an integrated approach of ensemble-based anomaly detection method and edge computing to solve the above problem. In the previous researches, ensemble-based isolation forest
(iForest) is proposed to provide fast anomaly detection in big data. Benefiting from the nature of the sampling-based ensemble, iForest possesses good detection accuracy with short processing time over extensive datasets [18]. Based on this remarkable scheme, Guha et al. proposed robust random cut forest (RRCF) to detect anomalies in dynamic data stream [9]. RRCF method improves the original data partitioning of iForest and update the tree structure through inserting and deleting leaves. Although the experiment showed that RRCF can capture the beginning and end of the anomalous event on a single data stream, it fails to provide perfect detection accuracy and real-time response on the data stream with multidimensional data instances. Therefore, we aim to design a generic anomaly detection framework with ensemble-based iForest technique to provide robust anomaly detection over different types of data streams. Besides, by means of edge computing, this framework is feasible to speed up the anomaly detection on the data stream with massive data.

In this paper, we propose a generic and real-time anomaly detection framework (IDForest) with locality sensitive hashing (LSH) based isolation forest in edge computing. Specifically, our framework utilizes dynamic insertion and deletion to incrementally update the tree structure, which can capture the change of data distribution on a data stream. Then, IDForest is deployed to edge servers in parallel, which divides the forest into multiple subtasks for concurrent processing. Based on this, the detection rate is enhanced hundreds of times, so IDForest can produce real-time alarms in most abnormal events. Compared with RRCF, our method is empirically more competitive on both detection accuracy and applied range, since IDForest takes LSHiForest as the theoretical basis while RRCF takes isolation forest as the theoretical basis. LSHiForest has been approved to cover the action scope of iForest, which corresponds to an instantiation of $l_1$ distance in LSHiForest [30]. The main contributions of this paper are three-fold:

1) **We propose a generic anomaly detection framework for the data stream with large-scale data and make a detailed analysis on insertion and deletion of our tree structure.**

2) **To detect anomalies and raise the alarm in real time, we deploy the proposed framework to edge servers in parallel and transmit the data from device node to edge node for processing.**

3) **Extensive experiments demonstrate the robustness and efficiency of our framework.** Specifically, we compare three instantiations of IDForest with the state-of-the-art methods. Our framework can detect the beginning and end of an anomaly in different types of data streams and the detection rate with edge computing is real-time in large-scale data within a specific window.

The rest of this paper is organized as follows. Related work is described in Sect. 2. Section 3 introduces the theoretical background and formulation of the approach. In Sect. 4, we elaborate the framework IDForest and describe the core processing in this framework. Then, IDForest is deployed to edge servers to produce real-time anomaly detection. In Sect. 5, extensive experiments demonstrate the validity of our method. Finally, we conclude the paper and give an outlook on possible future work in Sect. 6.
2 Related work

Much work about anomaly detection in the data stream with congestion data has been done in recent years, including general methodology, applications in different fields, formal methods model, etc. This paper focuses on robust and real-time anomaly detection methods for data streams, so we survey the research about ensemble-based anomaly detection methods, anomaly detection for data streams, anomaly detection with edge computing.

The subsampling scheme in ensemble-based anomaly detection contributes a lot to the efficiency of detecting anomalies in the large-scale dataset, and the average value of the ensemble results mitigates the influence of errors caused by individual judgment. Firstly, the subspace ensemble method was proposed by Lazarevic and Kumar, who used the average anomaly score, generated from lots of subspaces, as the final result to improve prediction accuracy [16]. Based on the ensemble scheme, Liu et al. proposed isolation forest for anomaly detection, which builds isolation trees through sampled data and uses depth of leaf to represent the anomaly score [19]. Then, isolation forest is proved to suffer from a lack of ability to detect complicated anomalies such as axis-parallel anomalies, so Zhang et al. proposed a generic framework, called LSHiForest, to deal with different kinds of anomalies. Recently, Xiang et al. proposed a learning-to-hash based isolation forest, which learns more information of data to partition the data instances by isolation tree [26]. This method improves the detection accuracy of LSHiForest at the expense of training time.

The early anomaly detection on data streams is to use a sliding window to slice the data over a period of time, then, some original anomaly detection methods are used to detect anomalies in the fixed time. Milad et al. proposed an online clustering algorithm to detect anomalies in clustering data streams. This method considers both temporal and spatial proximity to incrementally update the model and calculate the anomaly score of data in a sliding window. However, the application scope of this method is very limited. To deal with the concept drift of data stream, Tan et al. proposed MIR_MAD method, based on multiple incremental robust Mahalanobis estimators, to learn data stream incrementally [24]. Besides, to counter Advanced Persistent Threat (APT) attacks, Gao et al. built a novel stream-based query system (SAQL) to query whether the events are abnormal [8]. This system is specific for an enterprise data stream, which plays an important role in practical production and management. For broad anomaly detection on data streams, Robust Random Cut Forest (RRCF) is an effective method, which combines the iForest scheme and incremental learning to rapidly detect the change of data distribution or anomalies [3]. However, the detection rate for the data stream with congestion data is not real-time. Besides, limited by the data partition of isolation forest, the detection accuracy can be improved by LSHiForest. Manzoor et al. proposed a density-based ensemble outlier detector (xStream), which addresses the outlier detection problem for feature evolving streams [20]. This method can deal with different types of data streams while the efficiency needs to be improved.

To detect anomalies on data streams in real time, some efforts had been made by deploying an anomaly detection model on edge computing. On the Internet of Vehicles, anomaly detection models are widely deployed on edge computing to detect traffic congestion and violations in real time [25]. In Video Surveillance System, Chen et al. proposed a Distributed Intelligent Video Surveillance (DIVS) system by deploying deep learning algorithms on edge servers in parallel [6]. The experimental results showed that the execution time is much less than the original model deployment. However,
these methods have specific applications, which limits the scope to detect different types of anomalies. Recently, Mehnaz et al. proposed a privacy-preserving and real-time anomaly detection framework (W-Gaussian) on data stream [21]. Based on the statistic model, this method may have poor performance when the dataset does not satisfy the normal distribution.

The questions are still open on building a robust anomaly detection method over the data stream with complicated data as well as model deployment on edge computing. Inspired by RRCF and W-Gaussian, we improve the LSHiForest to be a dynamic anomaly detection framework on data streams through incrementally update the tree structure. Simultaneously with the efficiency, we use edge computing to speed up anomaly detection, so that the proposed framework can detect real-time anomalies on the data stream with massive data at a time.

3 Preliminaries and formulation

In this section, the basic model of the data stream is formulated, and the former work about LSHiForest is described. Firstly, data streams sent from multiple device nodes are merged to the edge node, so we can take the merged data as a data stream sorted by time order. Data streams are separated into two categories: one-dimensional time-series data stream and multidimensional data stream. For the first case, given a data stream \( D := [\ldots, x_{t-1}, x_t, x_{t+1}, \ldots] \), \( x_t \) represents the data instance entered at time \( t \). The idea is to partition the data stream into sub-sets of equal length over a sliding window of size \( w \), represented by \( d_t = < x_t, x_{t-1}, \ldots, x_{t-w} > \). Here, \( d_t \) is a new data instance with \( w \) dimensions, which represents the original state at time \( t \) on the data stream. In the second case, the data stream is more complicated, which is closer to real IoT applications. Given a data stream \( D := [\ldots, \alpha(t-1), \alpha(t), \alpha(t+1), \ldots] \), each data instance \( \alpha_t \) is multi-feature, denoted as \( \alpha(t) = < \alpha(t)_1, \alpha(t)_2, \ldots, \alpha(t)_n > \). For instance, several sensors collect humidity, temperature, CO2, and light in a green house, respectively. At intervals, these sensors will send all features simultaneously, which are merged on the edge node to be a complete data instance of time \( t \). Through continuous data collection and transmission, a complete data stream is formed.

Our framework improves the basic LSHiForest method and makes it learn the isolation tree incrementally to adapt to the complicated data stream. The core part of the method is LSH, which hashes closer data instances into the same “bucket” with a higher probability than data instances that are far apart [11]. Given a distance measure \( d(\cdot, \cdot) \), let \( d_1 < d_2 \) be two distance values, and \( p_1 > p_2 \) be two probability values. A family of functions \( \mathcal{F} \) is regarded as \((d_1, d_2, p_1, p_2)\)-sensitive, if for any hash function \( f \in \mathcal{F} \), \( x_i, x_j \in D \), the following two conditions hold:

1) \( d(x_i, x_j) \leq d_1 \Rightarrow P[f(x_i) = f(x_j)] \geq p_1 \);
2) \( d(x_i, x_j) \geq d_2 \Rightarrow P[f(x_i) = f(x_j)] \leq p_2 \).

Here, \( P[f(x_i) = f(x_j)] \) means the probability that instances \( x_i \) and \( x_j \) are hashed into the same “bucket” in a hash table. After getting the hash indices of data instances, the isolation tree can be built through different hash values (each value corresponding to a node in the tree). Finally, the anomaly score \( AS_i \) of a data instance \( x_i \) can be calculated by the path length \( pl_i \) of the leaf node, which contains \( x_i \). But considering the different tree structures and multi-branches in the forest, LSHiForest normalises the path length based on a reference path length denoted as \( \mu(S) \) [30], which is approximately calculated by:
\[
\mu(S) = \begin{cases} 
  x = (\ln(S) + \ln(v - 1) + \gamma)/\ln(v) - 1/2, & S > v \\
  1, & 1 < S \leq v \\
  0, & o.w.
\end{cases}
\]

where \(\gamma \simeq 0.5772\) is the Euler constant and \(v\) is the average number of branch factors in a tree.

Based on the reference path length and real path length, the anomaly score can be gotten by:

\[
AS(x) = 2^{-pl(x)/\mu(S)},
\]

Then, the symbols and notations used throughout the paper are described in Table 1.

In our framework, the pre-trained isolation forest is not constant. Since concept drift often occurs on data streams, a constant tree structure can perform poorly after the distribution of data changes. Thus, dynamic LSH-based iForest is designed for anomaly detection on data streams, through incremental insertion and deletion processing in the tree.

## 4 Framework design

In this section, we design a generic framework to detect anomalies of data streams in real time. Firstly, the architecture of the IDForest is demonstrated. Then, we present the core process of IDForest, which incrementally learns the stable tree structure by inserting and deleting leaves. Finally, the distributed deployment of our framework and the interaction between edge servers and devices are described.

### 4.1 Framework overview

To detect anomalies of dynamic data streams in real time, we propose a generic framework, called IDForest, which deploys the distributed LSH-Forest in edge servers and incrementally update the forest structure. The core part of IDForest is on how to incrementally learn
this forest structure and detect anomalies in testing data. The overview of our framework is shown in Figure 1.

Figure 1 shows the interaction and processing of data streams from sensors to edge servers. Specifically, multiple sensors collect the data of the same application and transmit the time-ordered data to edge servers. If the data is one-dimensional time-series data, different sensors send data to edge servers in time order and all these data build a queuing data stream in edge node. If the data is the multidimensional point, each sensor will collect one feature of a point and all the features collected at a time combine as a data instance. Then, the continuous arrival of data instances constitutes a queuing data stream. The edge node distributes the queuing data stream to multiple training tasks in parallel. In each training task, an isolation tree is pre-trained according to the historical data. Then, the isolation tree will incrementally update the tree structure as each data arrives. When the amount of data in the tree exceeds the maximum, it deletes the first arrived data before insertion a new data instance. Otherwise, the new coming data will be inserted to the tree directly. According to the pre-defined formulation of anomaly score, we can calculate the scores of data instances by the path length in a tree. Finally, all results of one data instance are globally aggregated to get the average anomaly score, which will be dispensed to corresponding sensors.

![Figure 1](image-url)
4.2 IDForest

In this part, the detailed structure of IDForest is introduced, including the node information in a tree, data processing, and model deployment on edge computing. Then, FIFO-based insertion and deletion processes are presented to maintain a stable forest structure, so as to detect anomalies of data streams in real time. This forest model is deployed to edge servers to deal with the curse of big data and dimensionality.

4.2.1 The improvement over the basic LSHiForest

In our framework, the core algorithm improves the LSHiForest method and changed it to detect anomalies of data streams. The key point in LSHiForest is to use a random hash projection to project the data instance into a lower-dimension space, then, the index values of the hash function are used to build the isolation tree. As has been discussed in last section, we calculate the average anomaly score in the forest by:

\[ \text{Ave}_{-} AS(x) = \frac{1}{t} \sum_{i=1}^{t} AS(x)_i. \]  

(3)

We also use this scheme to calculate the anomaly scores of data instances, but the tree structure is improved through incremental learning. Specifically, the tree structure of LSHiForest is fixed, which is incapable to deal with the dynamic data stream directly. In our framework, we introduce the insertion and deletion operations to improve the tree structure and update the data distribution on forest in real time to deal with complex data streams.

4.2.2 Update of the forest structure

To incrementally update the forest structure, we use the FIFO scheme to insert and delete leaves in a tree. Different from RRCF method, the forest in our framework consists of multifolk trees rather than binary trees. So, the insertion and deletion are more complicated.

Figure 2 An example of insertion operation
**Insertion of a Point:** Given an isolation tree with a data distribution \( LSHF(S) \), where \( S \) is the number of sampled data in this tree. The insertion of a new point \( p \) will produce a new data distribution \( LSHF(S \cup p) \). The example of insertion operation is shown in Figure 2.

There are six points in the original tree and data instance ‘6’ represents the new point from the data stream, shown in Figure 2. If point ‘6’ can find a search path in the tree, it will find the path until the leaf. If the point can find a leaf, which contains the same index path, this point is added to the leaf directly. Otherwise, a new leaf, containing this point, will be produced to the parent branch. When the depth reaches 3 through the search path, labeled in the red line, the point ‘6’ gets the different index path to point ‘2, 5 and 0’ in figure. So, a new “leaf(6)” is inserted to maintain the tree structure. If the number of points is larger than the limited number in a tree, the “first-in” point ‘0’ will be deleted. The detailed insertion operation algorithm is shown in Algorithm 1.

---

**Algorithm 1 Insertion operation**

**Input:** Index of the new point: \( Ind \), New point: \( p \), Original points in the tree: \( S \).

**Output:** Leaf.

1. if \( |S| = 0 \) then
2. return Leaf;
3. end if
4. Check for the duplicate of the point \( p \);
5. if is duplicate then
6. Add the point into the duplicated leaf;
7. return Leaf;
8. Otherwise, circularly search the index path of the point \( p \);
9. end if
10. if root is Leaf then
11. Create a new leaf node containing the new point;
12. Set the original node to be the leaf of the root;
13. Create a new root that contains these two leaves;
14. return Leaf;
15. end if
16. for \( i \) ← to Max_depth do
17. if node is Branch then
18. for \( k \) in node.k do
19. if key of point is equal to \( k \) then
20. Get the child node of the current node;
21. Set decision = False
22. end if
23. end for
24. if decision = True then
25. Create a new leaf node containing the new point;
26. Set the new leaf to be the child of current node;
27. return Leaf;
28. end if
29. else if node.key = point.key then
30. Add the point to the current node;
31. return Leaf;
32. else
33. Create a new leaf node containing the new point;
34. Create a new parent to replace the current node, and set the node and new leaf to be the children of this parent;
35. return Leaf;
36. end if
37. end for
Algorithm 1 illustrates the insertion operation on a new data instance from the queued data stream. If the root of a tree is empty, this point is inserted as a new leaf, shown in lines 1-3. Otherwise, check the duplicate of the new point. If there is a duplicate, the point is directly added to the duplicated leaf. If not, we will search the index path of the point until finding a leaf node or no index path. This process is shown in lines 16-37 of Algorithm 1. Observed that this point will be added to an existing leaf or newly created leaf.

Deletion of a Point: Given an isolation tree with a data distribution $LSHF(S)$ and the point $p$ to be the first point embedded to the tree. Then, a new data distribution $LSHF(S \cap p)$ will be produced with $p$ deletion. The example of deletion operation is shown in Figure 3.

There are two situations on deletion operation. Each leaf in the tree has a chronological label, which represents the order of point insertion. The “first-in” point will be deleted according to our FIFO strategy. So the leaf with point ‘0’ will be deleted in this case. If there are several siblings of the deleted leaf, the leaf with point ‘0’ is deleted straightly, shown in Figure 3(a). Otherwise, the sibling leaf will be the parent node with deleting the current leaf with point ‘0’, shown in Figure 3(b). Next, the detailed deletion algorithm is shown in Algorithm 2.

**Algorithm 2** Deletion operation

**Input:** Index of the deleting point: $ind$, Original points in the tree: $S$.

**Output:** All Leaves.

1. Find the $Leaf_{ind}$ of the deleting point.
2. if $Leaf_{ind}.n \geq 1$ then
3. return All leaves without $Leaf_{ind}$;
4. end if
5. if $Leaf_{ind}$ is the root then
6. Set the root node to None;
7. return All leaves without $Leaf_{ind}$;
8. end if
9. Find the parent node of $Leaf_{ind}$;
10. if parent.k $\geq 2$ then
11. Delete $Leaf_{ind}$ from the parent;
12. return All leaves without $Leaf_{ind}$;
13. end if
14. if parent node is the root then
15. Find the sibling of $Leaf_{ind}$;
16. Delete $Leaf_{ind}$ and set the sibling to be the root.
17. return All leaves without $Leaf_{ind}$;
18. end if
19. Find the sibling of $Leaf_{ind}$;
20. Find the grandparent of $Leaf_{ind}$;
21. Delete $Leaf_{ind}$, set the sibling to be the parent node, and build new relationship of grandparent and sibling;
22. return All leaves without $Leaf_{ind}$;
According to the storage structure of the pre-built tree, the deleted leaf can be found through the index number. If the number of points in this leaf is larger than 1, the point corresponding to the index will be removed directly. If the leaf is the root node, the root will be deleted and set to “None”, shown in lines 5-8 of Algorithm 2. After the above conditions are excluded, the next step is to find the sibling nodes. From lines 10-12, we can find it is easy to just delete the leaf while it has several sibling nodes. Otherwise, the relationship between their grandparent node and sibling node should be changed, since the sibling node will be a new parent node with the deletion of the current leaf.

4.2.3 Framework deployment on edge servers

To speed up the anomaly detection on data streams, the framework IDForest is deployed on edge computing in parallel. Benefit from the structure of forest, it is very convenient to design the parallel structure. Firstly, devices collect data from IoT applications and transmit the data to Edge servers. Data is queued in time order in which it arrives at the edge servers. In our framework, there are 100 trees. So, 100 training tasks start in parallel to process the same queuing data stream. When all training tasks are detected to be complete, the core management task will aggregate 100 results from the trees and average the results. Finally, the average anomaly score is determined to be a normal event or alarm, which will return a signal to the corresponding device.

The processing time in our framework is determined by data transmission time and execution time in edge servers. The data transmission time is associated with the channel gain, the transmit power, bandwidth, and Gaussian white noise power [7]. Denote the channel gain between the edge server and the smart device as $h_i$, then the achievable transmit rate is:

$$r_i = B \log_2(1 + P_i h_i/N_0 B),$$

(4)

where $P_i$ is the transmit power of the smart device, $N_0$ is the Gaussian white noise power. Given a dataset with size $D_i$, the latency is calculated by:

$$T_{\text{delay}} = D_i / r_i.$$  

(5)

Since the trees are processed in parallel in edge serves, the execution time of the above data is given by:

$$T_{\text{exec}} = \max\{D_i \times \log_2(S_i) \times m(1 <= i <= 100 & i \in \mathbb{N})\},$$

(6)

where $S_i$ is the number of points in $i$-th tree, $m$ is the number of dimensions of a data instance. Finally, we have the complete data processing time:

$$T = T_{\text{delay}} + T_{\text{exec}}.$$  

(7)

From Eq. 7, we can determine the importance to shorten the execution time of the framework, since the delay of device-edge structure is minimal. Especially, the execution time of large-scale data at a time outdistances the transmission time. In our framework, we not only shorten the time complexity of the method, but also speed up the anomaly detection by hundreds of times with parallelization technique.
5 Empirical evaluation

In this section, extensive experiments are conducted to verify the efficiency and scalability of our method against state-of-the-art methods. In Sect. 5.1, the experimental setups are listed and three instantiations (ImALSH, ImL1SH and ImL2SH) of our framework are introduced. Then, a synthetic dataset that captures the classic diurnal rhythm of human activity and multiple datasets from Numenta Anomaly Benchmark (NAB) [1] are used to validate the effectiveness of different methods. In Sect. 5.3, experiments are conducted on six real datasets. Finally, we analyze the latency and processing time of our framework, which is deployed on edge servers in parallel.

5.1 Experimental setups

For testbed experiments, five virtual machines are used to represent the device whereas the edge server is represented by an Intel(R) Core(TM) i7-11700KF CPU machine (3.6 GHz Dual-Core, 8 cores, 64 GB DRAM, and 2 NVIDIA GeForce RTX 3090 GPUs). Experimental codes are implemented in Python v3.7 and the communication between devices and edge servers is embodied by ZeroMQ [2]. All results are averaged over multiple runs (10 times).

Shingling is useful for detecting anomalies in data streams. For one dimensional data stream with the shingling 4, the values received on the stream at time $t_1, t_2, t_3, t_4$ are treated as a 4-dimensional point. If the sliding window take one unit at each time step, the next four-dimensional point consists of the values at time $t_2, t_3, t_4, t_5$. For multidimensional data stream, one data point arriving at a time is treated as a processing unit, and anomaly detection methods generate anomaly score for each data point. The sliding window contains multiple data points, where we calculate the response time for processing data under one sliding window. To save large-scale data in a sliding window, the length of sliding window is equal to the total time of the dataset in our experiments.

There are four instantiations in LSHiForest, including ALSH, L1SH, L2SH, and KLSH. Specifically, KLSH is much slower in detecting anomalies on large-scale datasets, compared with the former three instances. To detect the anomalies in real time, we only instantiate three efficient instantiations of our framework: ImALSH, ImL1SH, and ImL2SH. According to Zhang et al. [30], isolation forest is corresponding to the instantiation L1SH, so RRCF is corresponding to the instantiation ImL1SH in our framework. Since our method and RRCF are both based on the isolation forest, we use the same tree structure with 100 trees in the forest and 256 maximal data instances in a tree. The length of shingle is set to four, which is the same as the setting in RRCF method. Besides, windowed Gaussian anomaly detection and xStream use the same settings like that in the proposed papers [20, 21].

5.2 Comparison on synthetic datasets

In real life, the dataset in data streams implicitly reflect human circadian rhythms. For instance, shopping sites tend to receive more orders per hour at night while search engines may monitor more queries per minute by day. But an anomaly may reflect an unexpected dip or spike in these activities. To verify the validity of our method, we generate
synthetically generated a sine wave to represent the above dataset on a data stream, where a dip between time 205 and 255 is inserted [3]. Similarly, we inject different types of anomaly to the data that arrives in a time series. Specifically, four different types of anomaly cases, including flat-middle, jumps-down, jumps-up, and no-jump, are evaluated to illustrate the superiority of different methods, shown in Figure 4(b), (c), (d), and (e), respectively. All the datasets are from NAB [21].

Then, the different results in five datasets are illustrated and analyzed. The comparison in these datasets is to determine whether methods can detect the beginning and end of the injected dip.

In Figure 5, we show the results of running different methods on the dataset “sine wave”. From time 205-255, there is a straight line in Figure 5(a), which illustrates the beginning to ending of the anomalies. So, the windowed Gaussian method is useful to detect this kind of simple time-series anomaly. Similarly, the two highest scoring moments represent the beginning and end of the artificially injected anomaly in Figure 5(b), (d), (e), and (f). Only xStream can not detect this single injected anomaly in a sine wave.

In Figure 6, we show the results of running different methods on the dataset with an anomaly case: flat-middle. The methods windowed Gaussian and ImALSH miss the start
of the flat-middle while they both capture the end of this anomaly, shown in Figure 6(a) and (d). The rest methods can capture the beginning and end of the flat-middle, meaning they are able to find anomalies in this type of temporal data.

In “jumps-down” dataset, the windowed Gaussian and ImALSH cannot find the anomalies, since the last time of jumps down is short and the numerical variation of data instances is inapparent. Besides, RRCF can detect the end of the anomaly while it misses the beginning, shown in Figure 7(b). But it is useless to find the end of an anomaly, for the system has come back to the normal state after the end time. The best methods are ImL1SH, ImL2SH and xStream, which all detect the beginning and end of the jumps-down anomaly. Observed that there is a lower notch compared with the nearby notches in Figure 7(e), (f) and (c), corresponding to the jumps down in Figure 4(c).

In the dataset with jumps-up anomaly, ImALSH cannot detect the anomalies and the windowed Gaussian can not find the begging and end of the anomaly, shown in Figure 8(a) and (d). The rest methods can detect the jumps-up anomaly accurately. There is a relatively high peak, corresponding to the jumps-up anomaly, in Figure 8(b), (c), (e), and (f).

There is no jump anomaly but with a long trough at a certain time of the no-jump data-set, shown in Figure 4(e). Observed that the blue curves of RRCF, xStream, ImL1SH, and

---

**Figure 6** The anomaly score over time in “flat-middle”

---

**Figure 7** The anomaly score over time in “jumps-down”

---

Springer
more accurately reflect the characteristic distribution of dataset over time, while the other two methods cannot detect the longer trough time, shown in Figure 9.

With the above results of artificial temporal datasets, two instantiations (L1SH and L2SH) of our method can detect the beginning and end of different kinds of anomalies over time-series datasets. Besides, RRCF and xStream can capture the beginning and end of most different types of anomalies over time-series datasets. Next, we will compare the performance of accuracy and efficiency on various real-life data streams through these state-of-art methods.

5.3 Comparison on real-life datasets

In this section, extensive experiments are conducted on real life datasets. Except for the datasets used in the compared methods, some popular time-series datasets are collected from the UCI machine learning repository. All datasets are described in Table 2.

In Table 2, \( n \) represents the size of the dataset, \( m \) represents the dimension of the dataset, and ‘Rate’ denotes the proportion of anomalous data in total data. Specifically, NYC TaxiCabs is a stream data that represents the total number of passengers aggregated in a 30-minute window, which collects the taxi ridership data from the NYC Taxi
Commission [15]. Lavin et al. labeled the Independence Day, Labor Day, Labor Day Parade, NYC Marathon, Thanksgiving, Christmas, New Years Day and North American Blizzard as the anomalies. Then, we use “Nyc” to represent this dataset. Occupancy is a multi-features temporal dataset, which involves light, temperature, humidity, CO2, and humidity ratio in a room office [5]. In the experiment, two data streams (Occupancy_1 and Occupancy_2) are used to validate the performance of different methods. Then, we use “Occu_1” and “Occu_2” to represent the two datasets. Kddcup99 contains a 9-week collection of network connection data from a simulated USAF LAN [30]. We will use “Kdd99” to represent this dataset. Smtp is a processed subset of Kdd99. Firstly, a smaller set is forged by having only 3,377 attacks (0.35%) of 976,157 records, where attribute ‘logged_in’ is positive. From this forged dataset 95,156 ‘smtp’ service data is used to construct the Smtp dataset. Forestcover is used in predicting forest cover type from cartographic variables only (no remotely sensed data). This study area includes four wilderness areas located in the Roosevelt National Forest of northern Colorado. We select 10 quantitative attributes from 54 attributes of Forestcover and name this dataset as “Fc”.

As has been used extensively in many other works, including [14, 27, 30], we also use the Area Under receiver operating characteristic Curve (AUC) and Area Under Precision-Recall Curve (PRC) as the performance measures. The higher AUC represents the better performance, since the higher AUC means the anomalies can be detected with a lower false-positive rate and truth-negative rate. similarly, the higher PRC represents the better performance. To process the data stream in real time, the efficiency of the methods is also important in our cases. So, the testing time is compared next. To exclude the contingency, we run all the experiments 10 times and take the arithmetic average as the final result.

Firstly, We compare three instantiations (ImALSH, ImL1SH and ImL2SH) in IDForest with W_Gaussian, RRCF and xStream to see the effectiveness in detecting different kinds of data streams. All experiments are averaged 10 times and the standard deviation of 10 results is shown in brackets. Then, the AUC and PRC results are shown in Table 3 and 4. Our method and the best result are in bold.

We can observe from Table 3 that the instantiations of our framework have the best AUC results over most datasets except for the Kdd99, while xStream has a better AUC result in Kdd99. Besides, W_Gaussian performs pretty average in all datasets and RRCF performs terribly in Occupancy_2. For the PRC results, our framework performs the best in most datasets while xStream outperforms other methods in Occu_2 and Smtp, shown in Table 4. The standard deviation of W_Gaussian is always 0, since this method evaluates each time with all the data instances in sliding window. However, RRCF, xStream and our framework deploy the sampling technique on the ensemble-based model, which causes fluctuations in the results. Overall, our framework has competitive AUC and PRC results in

| #  | Name      | n    | m   | Outlier vs. Inlier Labels Rate |
|----|-----------|------|-----|------------------------------|
| 1  | NYC Taxicabs | 10320 | 1   | Class 1 vs. 0 .511%           |
| 2  | Occupancy  | 6841 | 5   | Class 1 vs. 0 .624%           |
| 3  | Occupancy_2 | 9752 | 5   | Class 1 vs. 0 .2101%          |
| 4  | Kddcup99   | 494021 | 38  | others vs. “neptune”, “smurf” & “normal” 1.77% |
| 5  | Smtp       | 95156 | 3   | Class 1 vs. 0 .03%            |
| 6  | Forestcover| 286048 | 10  | Class 1 vs. 0 .96%            |
### Table 3  AUC comparison between different methods

|       | W_Gaussian (%) | RRCF (%) | xStream (%) | ImALSH (%) | ImL1SH (%) | ImL2SH (%) |
|-------|----------------|----------|-------------|------------|------------|------------|
| Nyc   | 79.10          | 71.02    | 56.89       | **92.05**  | 56.61      | 73.72      |
|       | (±0.00)        | (±0.25)  | (±1.25)     | (±1.14)    | (±4.59)    | (±5.05)    |
| Occu_1| 74.35          | 74.75    | 96.04       | 89.56      | **96.17**  | 95.09      |
|       | (±0.00)        | (±0.59)  | (±0.54)     | (±0.93)    | (±0.24)    | (±0.89)    |
| Occu_2| 71.95          | 53.51    | 81.68       | 57.69      | 82.18      | **82.57**  |
|       | (±0.00)        | (±0.44)  | (±0.30)     | (±2.30)    | (±0.24)    | (±0.10)    |
| Kdd99 | 71.76          | 71.26    | **79.21**   | 48.37      | 75.91      | 76.90      |
|       | (±0.00)        | (±0.08)  | (±1.31)     | (±0.73)    | (±0.33)    | (±0.27)    |
| Smtp  | 75.86          | 79.60    | 81.54       | 80.96      | **81.62**  | 81.26      |
|       | (±0.00)        | (±1.15)  | (±2.83)     | (±1.09)    | (±1.13)    | (±0.76)    |
| Fc    | 73.45          | 65.12    | 74.45       | **77.49**  | 53.39      | 54.88      |
|       | (±0.00)        | (±0.56)  | (±0.77)     | (±1.14)    | (±0.78)    | (±0.52)    |

The bold value represents the best AUC result

### Table 4  PRC comparison between different methods

|       | W_Gaussian (%) | RRCF (%) | xStream (%) | ImALSH (%) | ImL1SH (%) | ImL2SH (%) |
|-------|----------------|----------|-------------|------------|------------|------------|
| Nyc   | 98.77          | 95.28    | 95.91       | **99.53**  | 95.44      | 97.59      |
|       | (±0.00)        | (±0.11)  | (±0.31)     | (±0.07)    | (±0.66)    | (±0.56)    |
| Occu_1| 97.03          | 97.33    | 99.35       | 99.11      | **99.73**  | 99.56      |
|       | (±0.00)        | (±0.09)  | (±0.19)     | (±0.12)    | (±0.02)    | (±0.12)    |
| Occu_2| 42.14          | 23.69    | **50.09**   | 39.82      | 45.47      | 44.92      |
|       | (±0.00)        | (±0.27)  | (±4.14)     | (±1.40)    | (±0.02)    | (±0.01)    |
| Kdd99 | 99.25          | 99.28    | 95.66       | 97.97      | 99.47      | **99.51**  |
|       | (±0.00)        | (±0.00)  | (±0.17)     | (±0.04)    | (±0.12)    | (±0.06)    |
| Smtp  | 4.02           | 5.56     | **35.56**   | 20.11      | 12.62      | 14.85      |
|       | (±0.00)        | (±0.51)  | (±0.75)     | (±5.23)    | (±2.03)    | (±3.05)    |
| Fc    | 3.73           | 0.67     | 12.34       | **27.75**  | 1.24       | 1.36       |
|       | (±0.00)        | (±0.01)  | (±2.62)     | (±1.31)    | (±0.01)    | (±0.03)    |

The bold value represents the best PRC result

### Table 5  Execution time comparison between different methods

|       | W_Gaussian(s) | RRCF(s) | xStream(s) | ImALSH(s) | ImL1SH(s) | ImL2SH(s) |
|-------|---------------|---------|------------|-----------|-----------|-----------|
| Nyc   | 0.28          | 3.79    | 2.61       | 0.36      | 0.59      | 0.66      |
| Occu_1| 1.43          | 162.57  | 39.59      | 6.94      | 14.03     | 10.10     |
| Occu_2| 2.82          | 201.60  | 54.35      | 8.21      | 47.36     | 33.39     |
| Kdd99 | 1943.40       | 6787.57 | 2126.27    | **326.38**| 2084.89   | 1802.70   |
| Smtp  | 6.98          | 1214.84 | 478.59     | 84.35     | 169.65    | 157.50    |
| Fc    | **70.22**     | 3527.61 | 1939.70    | 206.31    | 2092.73   | 1860.37   |

The bold value represents the least execution time
all datasets compared to other state-of-the-art methods, which validates a high robustness of the framework.

For real-life applications, we care about not only the accuracy but also the execution time of anomaly detection, which is evaluated next and shown in Table 5. For these multi-dimensional datasets on data streams, we process each data instance in time order, and the processing time of a whole dataset was calculated.

In Table 5, we can find that W-Gaussian has shorter execution time than other methods on small datasets while ImALSH has the fastest detection rate in the large dataset (Kdd99). Besides, RRCF and xStream are time-consuming in all datasets. However, iForest-based methods or ensemble-based methods can be deployed on edge servers in parallel, which can shorten the execution time hundreds of times by edge computing. So, these ensemble-based methods must be superior to W-Gaussian by parallel deployment. Finally, the execution time of our framework is around ten times faster than that of RRCF and seven times faster than that of xStream over all datasets.

Through the analysis of Sects. 5.2 and 5.3, we find that our framework can detect the beginning and end of an anomaly in time-series datasets. Besides, our framework, especially in L1SH and L2SH cases, has better accuracy to detect data stream with multidimensional dataset compared with other state-of-the-art methods. The execution time of our framework is relatively smaller than the similar ensemble-based methods, RRCF and xStream. Considering the parallel deployment on the edge servers, the real detection time of our framework can be hundreds of times less than that on a single machine. Next, we will collect the complete time of data transmission on devices and execution on edge servers to elaborate the real-time anomaly detection on data streams.

5.4 Time efficiency in edge servers

Our framework can be deployed on edge servers through parallelization technique, which reduces the execution time hundreds of times compared with that in a single server. In the simulation, the radius of each edge node is 50 m. Within the coverage area, there are several sensors randomly distributed and served by the corresponding base station through a wireless channel. The channel gains are produced by independent identical distributed. The transmission power of the device is 40 W. The detailed parameters are listed in Table 6.

The complete time of data transmission latency and data processing is detected, shown in Table 7. Then, we compare the complete time with the above execution time, shown in Table 5, to illustrate the efficiency of framework deployment on edge servers.

From Table 7, we can find that the complete transmission and processing time is less than one second in three small datasets: Nyc, Occu_1, and Occu_2. The longest latency and processing time are only around 16 seconds in the big dataset Kdd99, which is 130

| Table 6 | Simulation parameters |
| --- | --- |
| Parameters | Settings |
| Cell radius, R | 50m |
| System bandwidth, B | 15 MHz |
| Gaussian white noise power, N_0 | -174 dBm/Hz |
| Transmission power, P_i | 24 dBm |
| Path loss value, k | 0.01 |
times faster than that in a single server, shown in Table 5. Overall, our framework shortens the execution time around 100 times through parallel deployment on edge servers.

6 Conclusion

The proliferation of the data stream with large-scale data in IoT applications raises much more demand for real-time and robust anomaly detection. In this paper, we have investigated the recently developed anomaly detection methods on data streams. We have also analyzed the effect of edge computing on real-time data processing and surveyed the combination schemes of edge computing and anomaly detection. On this basis, we propose an edge computing empowered anomaly detection framework, named IDForest, which utilizes insertion and deletion schemes to update the tree structure. IDForest has the ability to detect anomalies on the data stream with massive data quickly and accurately, through incrementally learning the tree structure. Since the forest in our framework can be divided into multiple separate trees, we deploy the framework on edge servers in parallel and transmit the data from device node to edge node for processing. Finally, three instantiations of IDForest are compared with other state-of-the-art methods, which illustrates the outstanding effectiveness and efficiency of our method. Moreover, experiments on edge computing validate that the parallel deployment improves detection speed by hundreds of times.

In the future, we aim to deploy the framework in real edge computing scenarios and select the best instantiation for different applications. Besides, we will consider the effects of environmental noises on the experimental results and take measures to reduce these effects.

Acknowledgements This work was supported in part by the ARC DECRA Fund of the Australian Government under Grant No. DE210101458.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.
References

1. Numenata anomaly benchmark. URL https://github.com/numenata/NAB
2. Zeromq. URL http://zeromq.org/
3. rcf: Implementation of the Robust Random Cut Forest algorithm for anomaly detection on streams. The Journal of Open Source Software 4(35), 1336 (2019)
4. Bhatti, F., Shah, M.A., Maple, C., Islam, S.U.: A novel internet of things-enabled accident detection and reporting system for city environments. Sensors 19(9), 2071–2099 (2019)
5. Candanedo, L.M., Feldheim, V.: Accurate occupancy detection of an office room from light, temperature, humidity and co2 measurements using statistical learning models. Energy and Buildings 112, 28–39 (2016)
6. Chen, J., Li, K., Deng, Q., Li, K., Philip, S.Y.: Distributed deep learning model for intelligent video surveillance systems with edge computing. IEEE Transactions on Industrial Informatics (2019)
7. Du, J., Zhao, L., Jie, F., Chu, X.: Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee. IEEE Transactions on Communications 66(4), 1594–1608 (2018)
8. Gao, P., Xiao, X., Li, D., Jee, K., Chen, H., Kulkarni, S.R., Mittal, P.: Querying streaming system monitoring data for enterprise system anomaly detection. In: IEEE International Conference on Data Engineering (ICDE), pp. 1774–1777. IEEE (2020)
9. Guha, S., Mishra, N., Roy, G., Schrijvers, O.: Robust random cut forest based anomaly detection on streams. In: International Conference on International Conference on Machine Learning (ICML) (2016)
10. Gupta, M., Gao, J., Aggarwal, C.C., Han, J.: Outlier detection for temporal data: A survey. IEEE Transactions on Knowledge and data Engineering 26(9), 2250–2267 (2013)
11. Indyk, P.: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: Proceedings of the 30th ACM Symposium on Theory of Computing (STOC) (1998)
12. Jiang, Q., Xu, X., He, Q., Zhang, X., Dai, F., Qi, L., Dou, W.: Game theory-based task offloading and resource allocation for vehicular networks in edge-cloud computing. In: 2021 IEEE International Conference on Web Services (ICWS), pp. 341–346. IEEE (2021)
13. Khattak, H.A., Farman, H., Jan, B., Din, I.U.: Toward integrating vehicular clouds with iot for smart city services. IEEE Network 33(2), 65–71 (2019)
14. Kurt, M.N., Yilmaz, Y., Wang, X.: Real-time nonparametric anomaly detection in high-dimensional settings. IEEE Transactions on Pattern Analysis and Machine Intelligence (2020)
15. Lavin, A., Ahmad, S.: Evaluating real-time anomaly detection algorithms – the numenata anomaly benchmark. In: IEEE International Conference on Machine Learning and Applications (ICMLA) (2015)
16. Lazarevic, A., Kumar, V.: Feature bagging for outlier detection. In: ACM Conference on Knowledge Discovery and Data Mining (SIGKDD), pp. 157–166 (2005)
17. Lin, Y., Lee, B.S., Lustgarten, D.: Continuous detection of abnormal heartbeats from ecg using online outlier detection. In: Annual International Symposium on Information Management and Big Data (SIMBig), pp. 349–366. Springer (2018)
18. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation forest. In: IEEE International Conference on Data Engineering (ICDM), pp. 413–422 (2008)
19. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation-based anomaly detection 6(1), 1–39 (2012)
20. Manzoor, E., Lamba, H., Akoglu, L.: xstream: Outlier detection in feature-evolving data streams. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (SIGKDD), pp. 1963–1972 (2018)
21. Mehnaz, S., Bertino, E.: Privacy-preserving real-time anomaly detection using edge computing. In: IEEE International Conference on Data Engineering (ICDE), pp. 469–480. IEEE (2020)
22. Nelson, A., Toth, G., Linders, D., Nguyen, C., Rhee, S.: Replication of smart-city internet of things assets in a municipal deployment. IEEE Internet of Things Journal 6(4), 6715–6724 (2019)
23. Siddiqui, M.A., Fern, A., Dietterich, T.G., Wong, W.K.: Sequential feature explanations for anomaly detection. ACM Transactions on Knowledge Discovery from Data 13(1), 1–22 (2019)
24. Tan, C.H., Lee, V.C., Salehi, M.: Mixr_mad: An efficient and on-line approach for anomaly detection in dynamic data stream. In: IEEE International Conference on Data Mining Workshops (ICDMW), pp. 424–431. IEEE (2020)
25. Wang, X., Ning, Z., Wang, L.: Offloading in internet of vehicles: A fog-enabled real-time traffic management system. IEEE Transactions on Industrial Informatics 14(10), 4568–4578 (2018)
26. Xiang, H., Salcic, Z., Dou, W., Xu, X., Zhang, X.: OphiForest: Order preserving hashing based isolation forest for robust and scalable anomaly detection. In: ACM International Conference on Information and Knowledge Management (CIKM) (2020)
27. Xu, H., Wang, Y., Cheng, L., Wang, Y., Ma, X.: Exploring a high-quality outlying feature value set for noise-resilient outlier detection in categorical data. In: International Conference on Information and Knowledge Management (CIKM), pp. 17–26 (2018)

28. Xu, X., Fang, Z., Zhang, J., He, Q., Yu, D., Qi, L., Dou, W.: Edge content caching with deep spatiotemporal residual network for iov in smart city. ACM Transactions on Sensor Networks (2021)

29. Yuan, L., He, Q., Tan, S., Li, B., Yu, J., Chen, F., Jin, H., Yang, Y.: Coopedge: A decentralized blockchain-based platform for cooperative edge computing. In: The Web Conference 2021 (WWW) (2021)

30. Zhang, X., Dou, W., He, Q., Zhou, R., Leckie, C., Kotagiri, R., Salcic, Z.: Lshiforest: A generic framework for fast tree isolation based ensemble anomaly analysis. In: IEEE International Conference on Data Engineering (ICDE), pp. 983–994 (2017)