Hindawi
Security and Communication Networks
Volume 2021, Article ID 9954951, 12 pages
https://doi.org/10.1155/2021/9954951

*Research Article*

# AdaGUM: An Adaptive Graph Updating Model-Based Anomaly Detection Method for Edge Computing Environment

**Xiang Yu**[1], **Chun Shan**[2], **Jilong Bian**[3], **Xianfei Yang**[1], **Ying Chen**[1] and **Haifeng Song**[1]

[1]*School of Electronics and Information Engineering, Taizhou University, Taizhou 318000, China*
[2]*School of Electronics and Information, Guangdong Polytechnic Normal University, Guangdong 510006, China*
[3]*School of Information and Computer Engineering, Northeast Forestry University, Harbin 150001, China*

Correspondence should be addressed to Xiang Yu; yuxiang@tzc.edu.cn, Chun Shan; shanchun@gpnu.edu.cn, and Jilong Bian; bianjilong@nefu.edu.cn

Received 1 April 2021; Accepted 3 May 2021; Published 15 May 2021

Academic Editor: Muhammad Shafiq



With the rapid development of Internet of Things (IoT), massive sensor data are being generated by the sensors deployed everywhere at an unprecedented rate. As the number of Internet of Things devices is estimated to grow to 25 billion by 2021, when facing the explicit or implicit anomalies in the real-time sensor data collected from Internet of Things devices, it is necessary to develop an effective and efficient anomaly detection method for IoT devices. Recent advances in the edge computing have significant impacts on the solution of anomaly detection in IoT. In this study, an adaptive graph updating model is first presented, based on which a novel anomaly detection method for edge computing environment is then proposed. At the cloud center, the unknown patterns are classified by a deep leaning model, based on the classification results, the feature graphs are updated periodically, and the classification results are constantly transmitted to each edge node where a cache is employed to keep the newly emerging anomalies or normal patterns temporarily until the edge node receives a newly updated feature graph. Finally, a series of comparison experiments are conducted to demonstrate the effectiveness of the proposed anomaly detection method for edge computing. And the results show that the proposed method can detect the anomalies in the real-time sensor data efficiently and accurately. More than that, the proposed method performs well when there exist newly emerging patterns, no matter they are anomalous or normal.

## 1. Introduction

With the rapid development of Internet of Things and wireless technologies, more and more applications, which are enabled by IoT, such as smart home appliances, self-driving, and intelligent temperature control, appear in our daily life. As an indispensable component of IoT, all kinds of sensors are widely used for data collection in the tasks of various fields [1–3]. According to the research on the anomaly detection for sensor data [4], it is hard to determine the occurrence time and frequency of the anomalies. And it is inevitable that both normal data and anomalous data are involved in the massive collected sensor data. Hence, the anomaly detection in IoT scenarios brings us new challenges. On the one hand, anomalies should be detected in real-time rather than being detected at the cloud center after a period of time. For example, anomalous temperature data from the sensors deployed in a forest might represent a fire warning which means that immediate action is required. On the other hand, the accuracy of anomaly detection should be ensured because high false-positive rate will lead to frequent false alarm and high false-negative rate will lead to anomalies undetected.

In most traditional anomaly detection methods, a behavior is considered as anomalous when its features are the same as or similar to the features of a known anomalous pattern. First, a behavior is represented as feature vector that consists of part or all features of the behavior. Similarly, all the known patterns, either anomalous or normal, are represented as feature vectors. Second, the similarity between the behavior to be detected and each known anomalous pattern is calculated, and the behavior is considered as anomalous when the similarity exceeds a

predefined threshold. Generally, the anomaly detection methods can be roughly classified as distance-based detection methods, statistics-based detection methods, density-based methods, or the methods based on neural network [5–8]. However, almost all the traditional anomaly detection methods suffer from the disadvantage of over reliance on the pretrained static detection model, which will lead to the degradation of detection performance when the anomalies of new type occur. In view of this, more and more machine learning algorithms are employed in anomaly detection tasks due to the characteristics of improving the detection model adaptively according to the incremental learning results [9–11]. Nevertheless, due to the requirement of real-time detection, the anomaly detection methods, in which corresponding machine learning algorithms are employed, still cannot detect anomalies from massive sensor data efficiently.

According to the statistics, the number of IoT devices is estimated to grow to 25 billion by 2021. In order to implement the anomaly detection for sensor data in IoT, the cloud computing model is widely employed where the collected sensor data are analyzed. Actually, not only the anomaly detection model is deployed at the cloud platform but also all the sensor data are collected and transmitted to the cloud platform, so that the detection model can be conveniently improved according to the changes of the collected sensor data. However, some anomalies of certain types need to be detected in real-time, rather than being detected at the cloud platform. Suppose such a scenario, and the sensor data collected from a moving vehicle show that there exist some anomalies in the components of the vehicle which makes the vehicle in danger. Obviously, the sensor data should be detected in real-time, and the corresponding measures should be taken immediately so as to avoid the occurrence of an accident. In addition, with the growing scale of IoT, more and more wireless devices are involved, which means an increase of the computing pressure on cloud platform [12, 13]. In order to alleviate the pressure on both the network bandwidth and the computing power of cloud platform, a possible solution is the edge computing, which migrates computation-intensive tasks from the cloud platform to the edge severs where the delay-sensitive detection tasks can also be performed in real-time.

The remainder of the study is organized as follows. First, the related work on anomaly detection is introduced. Second, we propose an adaptive graph updating model (AdaGUM), which records the features of each known patterns. Third, the architecture that can dynamically orchestrate the anomaly detection method for edge computing and the feature graph updating model is described in detail. Fourth, several experiments are conducted in order to validate both the efficiency and the effectiveness of the proposed method. Finally, we conclude the study and discuss the research focus of our future work.

## 2. Related Work

*2.1. Traditional Anomaly Detection Methods.* The basic principle of anomaly detection is to detect the data which are deviated from the known normal patterns according to a predefined similarity threshold. Currently, the common anomaly detection methods include the anomaly detection methods based on distance [14, 15], the anomaly detection methods based on density [16, 17], the anomaly detection methods based on classification [18, 19], and the anomaly detection methods based on models [20–22]. With the development of machine learning, lots of advanced algorithms and models have been proposed successively and integrated with the existing traditional anomaly detection methods in order to improve their detection performance on both accuracy and efficiency [23–25], such as the detection methods based on the neural network model, the detection methods based on $k$ nearest neighbor ($k$-NN), and the detection methods based on the support vector machine (SVM).

After years of research, lots of anomaly detection methods for different application scenarios have been proposed in succession. In year 1994, based on a statistical model, Barnett et al. proposed an anomaly detection method, whereby the data different from the given distribution are detected as outliers. In year 1999, a classic anomaly detection method, which is based on the local outlier factor (LOF), is first proposed by Breunig et al. After that, based on the LOF, a series of similar anomaly detection methods are proposed in succession [26–29]. In year 2000, based on $k$-NN, Ramaswamy et al. proposed a detection method whereby anomalies can be detected without any prior knowledge [30]. In year 2001, an anomaly detection method based on the support vector machine (SVM) is proposed by Schlkopf et al. [31]. In year 2009, Angiulli et al. propose a novel algorithm which can detect outliers from very large datasets [32]. In year 2011, on the basis of several different similarity measurements, Moshtaghi et al. propose an anomaly detection method [33]. In year 2014, according to the calculation results of top-k distance, Shaikh et al. proposed an anomaly detection method [34]. In year 2014, Shaikh et al. proposed a method for anomaly detection, which can detect outliers from uncertain datasets [1]. In year 2014, Huang et al. proposed an anomaly detection model, and different machine learning algorithms can be separately integrated with the proposed model [35]. However, with the emergency of more and more big data applications, traditional anomaly detection methods might suffer from the following disadvantages in the big data processing tasks. First, the performance of some traditional methods fluctuates with the data distribution. Second, the performance of some traditional methods deteriorates with the increase of data dimension. Third, the performance of some traditional methods is limited by the predetermined models. In addition, the performance of some traditional methods depends too much on the prior knowledge.

*2.2. Anomaly Detection for Edge Computing.* With the development of the technologies of Internet of Things and wireless sensor networks (WSN), when analyzing the massive sensor data, traditional methods become more and more inapplicable. Evidently, due to the resource constraints of each sensor node, it is impracticable for traditional methods to detect anomalies from the collected data at each sensor node under the real-time requirements. In view of the existing problem, some promising anomaly detection methods for IoT

devices have been proposed in succession. And it should be noted that most existing anomaly detection methods for the sensor data from IoT devices are based on the cloud computing model and big data processing platform [36, 37]. With the powerful computing and analyzing power of the cloud computing center, the data collected from IoT devices are transmitted to the cloud computing center for further detection. However, as estimated, the IoT devices will grow to 25 billion by 2021, and it is impracticable to transmit all the data, which are collected from IoT devices, to the cloud computing center due to the limitation of network bandwidth; moreover, it is also impracticable to process or analyze the massive collected data at the cloud center due to the limitation of computing power.

As the considerable transmission cost and the pressure on computing power may degrade the performance of anomaly detection, especially for the computing-intensive and delay-sensitive tasks, some anomaly detection methods based on the edge computing strategy are proposed, whereby some computing tasks are migrated from the cloud center to network edge devices (nodes). In year 2017, Cai et al. integrated the strategy of distributed recursive computing with a selection method based on $k$-NN, so as to detect anomalies from time series data [38]. In year 2018, in order to detect anomalies from sensor data in real-time, a novel anomaly detection method is proposed by Zhang et al. [39]. In year 2020, Anand et al. proposed an edge-based intrusion detection method for IoT devices, which can distinguish evolving forms of attacks from normal behaviors [40]. Additionally, some research studies on anomaly detection methods based on the deep learning model are proposed in succession whose performance is better than that of the aforementioned methods based on shallow learning models [41, 42]. Although most existing anomaly detection methods for edge computing can effectively achieve service decentralization and computing tasks migration, they still suffer from the detection accuracy, especially when there exist the sensor data of some unknown patterns that cannot be identified by the pretrained static model.

In this study, we propose AdaGUM, an adaptive graph updating model, based on which a novel anomaly detection method for edge computing is proposed. From the perspective of edge nodes, anomalies can be detected on each edge node in real-time according to the feature graph which is preserved and periodically updated. From the perspective of cloud computing center, the collected data of some unknown patterns can be first identified by a deep learning model, and then, the identified patterns are not only transmitted to the cache of each edge node constantly but also stored on the cloud computing center for the coming periodic updating of feature graph.

## 3. AdaGUM Model

In this section, according to the formulation of anomaly detection for edge computing, the related definitions are first introduced and then follows the description of the deep learning model together with the AdaGUM model for edge computing; finally, the solution of feature graph updating together with the corresponding algorithms is proposed in detail.

*3.1. Problem Formulation.* Suppose the data collected from different sensors, which consist both normal data and anomalies, are denoted as $D_t = \{s_1(t), s_2(t), \ldots, s_n(t)\}$, where the subscript $i$ means that the sensor data are collected from the $i^{\text{th}}$ sensor, and $t$ denotes the generation time of the data. In addition, each $s_i(t)$ can be further denoted as $s_i(t) = \{s_{i1}(t), s_{i2}(t), \ldots, s_{ij}(t)\}$, where $s_{ij}(t)$ denotes the value of data $s_i(t)$ on the $j^{\text{th}}$ dimension. And $s_{ij}(t)$ can be abbreviated as $s_{ij}$ when the data generation time is not considered.

*Definition 1.* Anomalous value: if the values of the sensor data on one or more dimensions exceed the normal range, the values are considered as anomalous values. Actually, an anomaly may contain more than one anomalous values.

*Definition 2.* Anomalous pattern: an anomalous pattern consists of the corresponding anomalous range on each dimension, which can be represented as $\xi = \{\xi_1, \xi_2, \ldots, \xi_k\}$, $1 \le k \le j$, where $\xi_k$ denotes an anomalous range on the $k^{\text{th}}$ dimension.

*Definition 3.* Anomalous feature vector: an anomalous feature vector can be represented as $\text{AVec} = \{s_{ik} | s_{ik} \notin \xi'_k, 1 \le k \le j\}$, where $\xi'_k$ denotes the normal range on the $k^{\text{th}}$ dimension.

*Definition 4.* Anomalous feature graph: all the anomalous feature patterns can be integrated into a anomalous feature graph, whereby each existing anomalous feature vector corresponds to a series of nodes in the graph. Moreover, the anomalous feature graph can be updated periodically at the cloud computer center according to the feature changes of anomalies.

*3.2. The AdaGUM Model for Edge Computing.* As discussed before, the IoT anomaly detection solutions can be broadly classified into the centralized anomaly detection methods based on the cloud computing center and the distributed anomaly detection for edge computing. Evidently, with the increasing of IoT devices, which bring more and more sensor data, it is necessary to take the considerable transmission cost and the latency of data processing or analyzing into account. In this study, we propose the AdaGUM model for edge computing which can perform the computation-intensive and delay-sensitive anomaly detection tasks effectively. As the functional structure of AdaGUM shown in Figure 1(b), at the sensors layer, the data are first collected from various of sensors in real-time and then transmitted to the corresponding edge layer for detection. Concurrently, the emerging unknown patterns, which cannot be identified, are reported to the cloud layer for further processing. With the powerful capability of analyzing and processing, the cloud computing center plays a crucial role and achieves a more intelligent way of work. And the main functions of the cloud computing center include data storage, data integration, and data analysis.

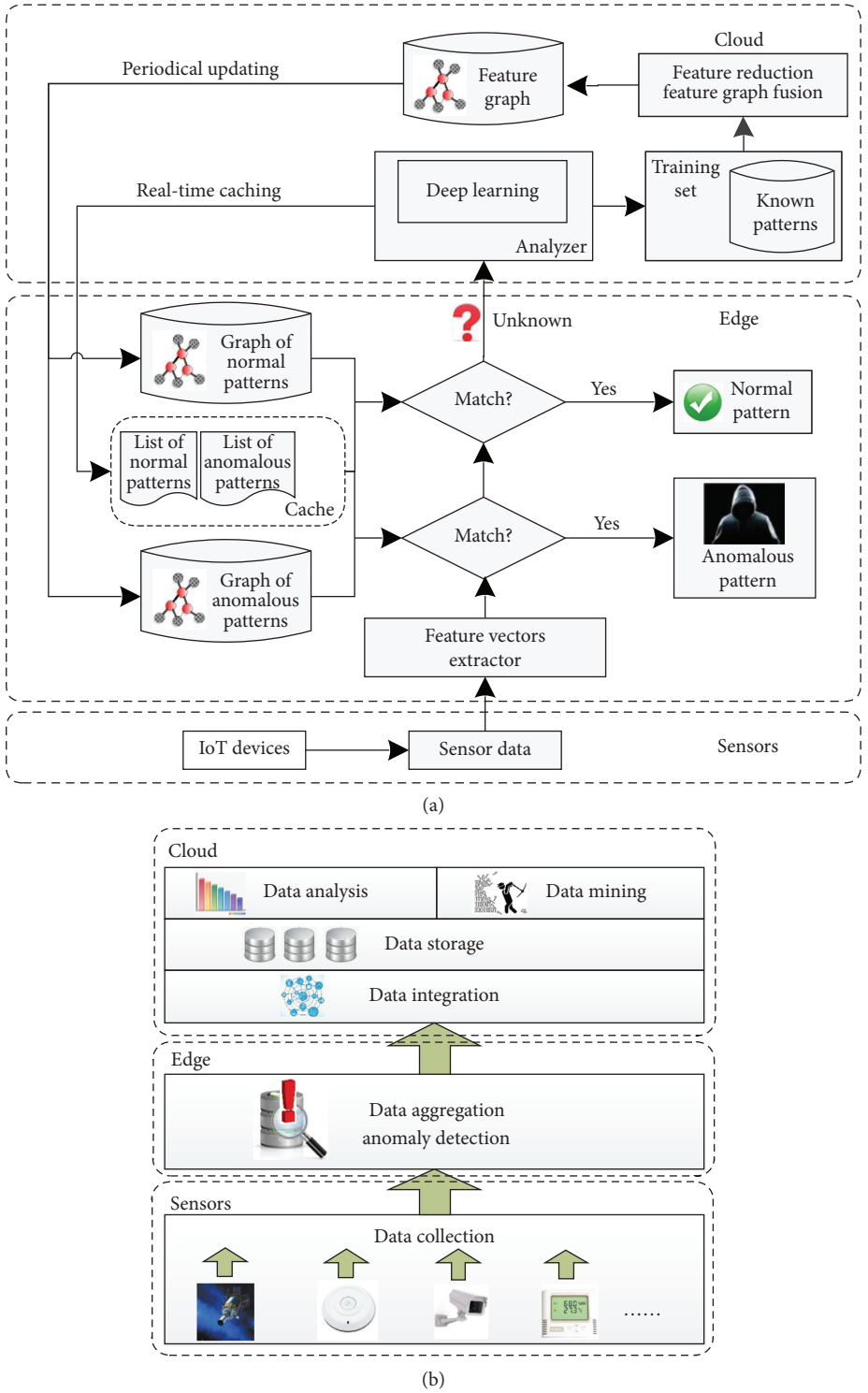


Figure 1: The AdaGUM model. (a) The logical deployment of AdaGUM. (b) The functional structure of AdaGUM.

As the logical deployment of AdaGUM shown in Figure 1(a), the main function of the sensors layer is to collect and transmit the data generated by various emerging applications, which are enabled by the IoT and wireless technologies. Hence, we put focus on the logical functions and deployment details of the edge layer and cloud layer.

*3.2.1. The Edge Layer in AdaGUM.* As the edge layer shown in Figure 1(a), the sensor data to be detected are first preprocessed, if necessary, and transformed into the feature vectors by the feature vectors extractor. Next, the vectors are compared with both the list of anomalous patterns, which is temporarily kept in cache until the next periodical feature graph updating, and the graph of anomalous patterns where

the existing anomalous patterns that have emerged up to the latest updating is recorded. When the similarity between the feature vector to be detected and an existing anomalous pattern exceed a certain threshold, the vector is identified as a potential anomaly. Otherwise, the feature vector is further compared with the list of normal patterns in cache and the graph of normal patterns. When the similarity between the feature vector to be detected and an existing normal pattern matches, the feature vector is considered as normal. If neither of the aforementioned conditions is satisfied, the feature vector is transmitted to the cloud computing center for further analysis.

*3.2.2. The Cloud Layer in AdaGUM.* As the cloud layer shown in Figure 1(a), the set of initial known patterns, including both the anomalous patterns and the normal ones, can be obtained by learning from the training set, and the corresponding feature graphs are generated based on the known patterns. During the periodical updating, the feature graphs, which record the anomalous patterns and the normal patterns, are allocated to each edges node. Next, the unknown feature vectors reported from edge nodes are analyzed and labelled by the analyzer, where a deep learning model is built to judge whether there exist anomalies or not. As a binary classification problem, the input of the model is a n-dimensional time series data, and its output is a 2-value classification which are anomalous or normal. As Figure 2 of model structure shows, each convolution module consists of 1-dimension convolution, batch normalization, and dropout.

In equation (1), $S$ and $O$ denote the input and output, respectively, and in equation (2), $B$ denotes the bias, $W$ denotes the matrix of weights, $p$ denotes the probability, $\mu$ denotes the mean, $\sigma$ denotes the variance, and $\epsilon$ is a small value which prevents the denominator from being zero. And the function of convolution is to detect the patterns in the sequence and output the corresponding features. The input of the batch normalization layer is the output of the convolution layer, and the input is normalized subsequently. Then, the negative values are filtered by the rectified linear unit, and nonlinearity is introduced. In order to avoid overfitting, part of the outputs are removed by dropout randomly. And the decision layer consists of several fully connected layer modules which are constructed by the linear layer, batch normalization, and dropout.

$$O = \text{Dropout}\left(\text{ReLU}\left(\text{batch}\left(\text{conv}\left(S\right)\right)\right)\right), \tag{1}$$

$$\begin{cases} \text{Conv}\left(S\right)_{k,i} = B_k + \sum_{l=1}^{m_k} \sum_{j=1}^{C} S_{i+l-1,j} W_{k,l,j}, \\ \text{Batch}\left(X\right) = \dfrac{X - \mu_X}{\sqrt{\sigma_X^2 + \varepsilon}}, \\ \text{ReLU}\left(x\right) = \max\left(0, x\right), \\ R \sim \text{Bernoulli}\left(p\right), \\ \text{Dropout}\left(X\right) = R * X. \end{cases} \tag{2}$$

In equation (3), $S_f$ is the output of feature layers, the results of classification $P$ denotes the probability outputs that consist of two values. $B$ denotes the bias and $W$ denotes the matrix of weights.

$$\begin{aligned} P &= \text{Dropout}\left(\text{ReLU}\left(\text{batch}\left(\text{linear}\left(S_F\right)\right)\right)\right), \\ \text{Linear}\left(S_F\right) &= B + W \times S_F. \end{aligned} \tag{3}$$

After being classified, the reported feature vectors together with their labels are transmitted to the lists of both normal patterns and anomalous patterns on each edge node, respectively. Simultaneously, the classification results of the reported feature vectors are also integrated with the collection of known patterns at the cloud computing center. Subsequently, the feature graph can be updated later based on the new collection of known patterns. It is worth noting that $m$ the sensor data transmitted to the edge nodes are detected in terms of both the periodically updated feature graphs from the cloud computing center and the lists of patterns which is constantly updated.

*3.3. The Feature Graph.* Generally, each sensor data generated from IoT devices can be regarded as a vector that consists of one or more values on the corresponding dimensions. As shown in Figure 3, the range of sensor data on each dimension is partitioned into different intervals denoted as $\xi_{m1}, \xi_{m2}, \ldots, \xi_{mn}$, where $m$ denotes the serial number of the dimension, and $1, 2, \ldots, n$ denotes the serial number of the intervals on the $\text{m}^{\text{th}}$ dimension. Then, each sensor data can be located in the corresponding intervals on each dimension. For example, suppose there exist sensor data $s_i = \{s_{i1}, s_{i2}, \ldots, s_{im}\}$, which is transmitted to a nearest edge server/node. First, the $m$-dimensional data space is partitioned into $m * n$ intervals, where each dimension consists of $n$ intervals. Second, the components in $s_i$ is reordered according to a predefined order, so as to obtain the corresponding feature vector. Assuming feature vector 1 in Figure 3 is the feature vector which is obtained after the reordering step. Then, feature vector 1 can be located in the corresponding interval of each dimension according to its values on all $m$ dimensions, as the red dotted lines indicate. Similarly, all sensor data can be transformed into feature vectors. After all anomalies in the training set are located in the corresponding intervals, a feature graph of anomalous patterns can be obtained by extracting all the nonempty intervals on each dimension. Similarly, the feature graph of normal patterns can be constructed in the same way. It is worth noting that the links between intervals are preserved as the edges in feature graph.

Evidently, from the construction process of feature graph, we can draw the conclusion that a newly emerging anomalous pattern can be easily merged into the a existed feature graph. On the contrary, an old anomalous pattern can also be removed from a existed feature graph when it is no longer considered as anomalous. As shown in Figure 4, it is flexible to preserve anomalous patterns with feature graph. On the one hand, different feature graphs can be fused as needed. On the other hand, obsoleted patterns can be easily removed from feature graph.

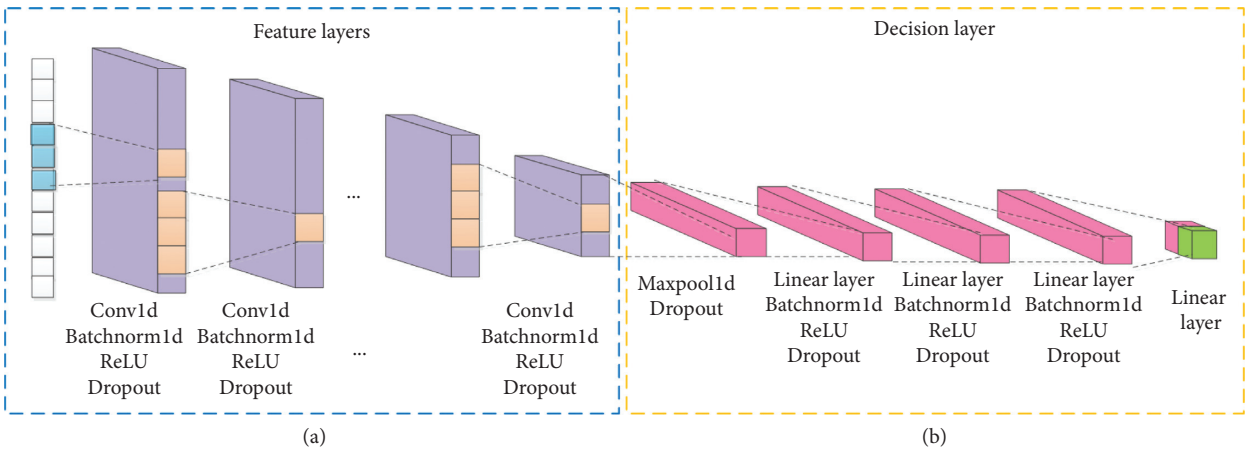


Figure 2: The 2-value classification model.

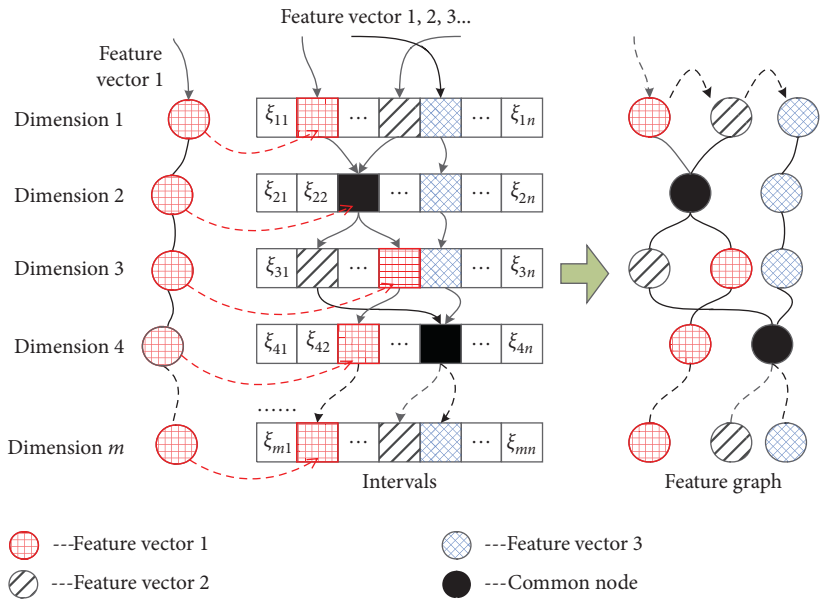


Figure 3: The construction of feature graph.

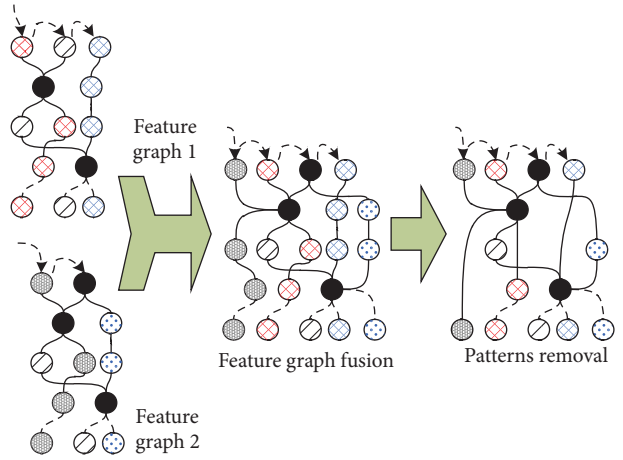


Figure 4: The feature graph fusion and the existed patterns removal.

### 3.4. Anomaly Detection Algorithm

*3.4.1. Anomaly Detection on Edge Nodes.* In this section, the anomaly detection algorithms on edge nodes and the cloud computing center are described in detail, respectively. Explicitly, the anomaly detection algorithm deployed on edge nodes is different from the one which is deployed at the cloud computing center. As the anomaly detection algorithm on edge nodes described in algorithm 1, sensor data are first transformed into feature vectors, and then, the vectors are compared with not only the graphs of anomalous patterns and normal patterns but also the lists of anomalous patterns and normal patterns which are preserved in cache. If the vector does not match all the patterns, it is reported to the cloud computing center for further detection.

*3.4.2. Anomaly Detection at Cloud Computing Center.* As the anomaly detection algorithm at the cloud computing center described in algorithm 2, with the intelligent machine learning methods, the reported unknown vectors are clustered and labelled as normal or anomalous. Subsequently, the newly emerging patterns that consist of both anomalous patterns and normal ones are distributed to the cache on each edge node. After a updating cycle, the importance of each feature in all known patterns are recalculated, whereby the feature graph is updated and distributed to all edge nodes.

## 4. Experiments

In this section, we evaluate the performance of the proposed anomaly detection method for edge computing through extensive simulations. First, we provide details on the hybrid datasets which include both normal data and anomalies. Second, the indexes, which are employed to evaluate the results of performance comparison between AdaGUM and the baseline methods, are introduced. And the formulas for the calculation of index values are outlined. Third, we further introduced the baseline anomaly detection methods whose performance are compared with the proposed method. Finally, we compare the performance of AdaGUM with that of the baseline methods based on the experiments which are conducted on the same datasets and analyze the experimental results in detail.

*4.1. The Datasets.* In this study, the experiments for the performance evaluation of AdaGUM are conducted on several representative datasets. The first one is the dataset named detection of IoT botnet attacks, which is abbreviated as BaIoT. There are 7062606 instances, each of which has 115 attributes, involved in the BaIoT dataset. Authentically, the data in BaIoT are gathered from 9 commercial IoT devices which are infected by Mirai and Bashlite. And there does not exist missing values in the data of BaIoT. The second one is the El Nino dataset whose data are collected from the Tropical Atmosphere Ocean (TAO) array, which consists of over 70 moored buoys. There are 178080 instances, each of which has 12 attributes, involved in the El Nino dataset. Since there exist missing values, the experimental data in the El Nino dataset need to be preprocessed. In addition, in order to further test the flexibility and scalability of AdaGUM, we construct a synthetic dataset with 20000 instances from the Mulcross data generator, and the proportions of known anomalies and unknown anomalies are both 5%. The basic information of the experimental datasets is given in Table 1.

*4.2. The Baseline Methods.* Due to the outstanding performance on clustering and classification, two classic machine learning methods, $k$-nearest neighbor and one-class support vector machine (SVM), are chosen as the baseline methods for the performance comparison with AdaGUM. As to the parameter settings, the best parameters for these algorithms are selected by the leave-one-out cross-validation and a grid search strategy. Moreover, based on the adaptive graph updating strategy, we develop an anomaly detection method for cloud computing, which is recorded as AdaGUM_CL in order to distinguish from the proposed anomaly detection method for edge computing, AdaGUM. Then, we compare the performance between the four methods on both the detection effects and the detection efficiency.

*4.3. The Evaluation Indexes.* In the experiments, we evaluate the performance of AdaGUM with two indexes which have been generally employed based on consensus, that is, true-positive rate (TPR) and false-positive rate (FPR). In the process of anomaly detection, TPR indicates the ratio of the true anomalies which are correctly detected. Evidently, the task of anomaly detection is to detect the anomalies correctly as much as possible, as to the TPR index, the higher the better. On the contrary, FPR indicates the ratio of the data, which are actually normal and mistakenly detected as anomalies. As to the FPR index, the lower the better. The calculation formulas of TPR and FPR are listed in equations (4) and (5), where true-positive (TP) denotes the anomalies which are detected correctly, false-negative (FN) denotes the anomalies which are mistakenly detected as normal data, false-positive (FP) denotes the normal data which are mistakenly detected as anomalies, and true-negative (TN) denotes the normal data which are detected correctly.

$$\text{True} - \text{positive rate}\,(\text{TPR}) = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{4}$$

$$\text{False} - \text{positive rate}\,(\text{FPR}) = \frac{\text{FP}}{\text{FP} + \text{TN}}. \tag{5}$$

### 4.4. Experimental Results and Analysis

*4.4.1. Detection without Unknown Anomalies.* In this experiment, each sensor data to be detected must be in accordance with a certain known pattern. Then, we compare the performance of AdaGUM with the performance of the other three anomaly detection methods, which are

```
Input: θ-the threshold of similarity degree
G_Ano-the feature graph of anomalous patterns
G_nor-the feature graph of normal patterns
L_Ano-the list of anomalous patterns in cache
L_nor-the list of normal patterns in cache
Output: C_Ano-the collection of detected anomalies
C_Nor-the collection of detected normal data
C_Unk-the collection of the data whose pattern are unknown
(1)C_Ano←φ
(2)C_Unk←φ
(3)D←φ
(4)flag←UNLABELLED
(5)Q←φ;//The queue of sensor data
(6)while (GetSensorData(D))
(7){Q←Q⋃Transform(D);
(8)   while (!Empty(Q))
(9)   {s←DeQueue(Q);
(10)      for (each p in G_Ano)
(11)         if (Match(s, p, θ))
(12)            {C_Ano←C_Ano⋃s; flag←LABELLED; }
(13      for (each p in G_Nor)
(14)         if (Match(s, p, θ))
(15)            {C_Nor←C_Nor⋃s; flag←LABELLED; }
(16)      for (each l in L_Ano)
(17)         if (Match(s, l, θ))
(18)            {C_Ano←C_Ano⋃s; flag←LABELLED; }
(19)      for (each l in L_Nor)
(20)         if (Match(s, l, θ))
(21)            {C_Nor←C_Nor⋃s; flag←LABELLED; }
(22)         if (flag! = LABELLED)
(23)            {C_Unk←C_Unk⋃s; }
(24)      flag←UNLABELLED;
(25)   }
(26)}
(27)returnC_Ano, C_Nor, C_Unk;
```

Algorithm 1: Anomaly detection on edge nodes.

AdaGUM_CL, one-class SVM, and $k$-NN. The experiment is conducted on both BaIoT dataset and El Nino dataset. As to kernel of one-class SVM, the widely approved radial basis function kernel is employed. As to $k$-NN, we set the parameter $k = 30$, which is larger enough compared with the size of any anomaly cluster. As to AdaGUM and AdaGUM_CL, we set $\theta = 0.8$, which means that sensor data are considered to be a pattern only if more than 80% of its dimension values are in accordance with the corresponding ranges of the pattern. And the feature graph is updated once when every 1000 sensor data are detected. As to AdaGUM and AdaGUM_CL, we focus on the average TPR and FPR within each updating cycle. As the performance comparison shown in Figure 5, all the four methods perform well; however, the performance of $k$-NN is slightly inferior to the other three methods, which is mainly because the performance of $k$-NN is influenced by the initial value of $k$, and the best $k$ value is hard to be determined.

*4.4.2. Detection with Unknown Anomalies.* In this experiment, 10% of the known anomalies and normal data are initially considered as unknown whose labels are removed artificially. Then, we compare the performance of AdaGUM with the performance of the other three anomaly detection methods. The experiment is conducted on both BaIoT dataset and El Nino dataset. As to kernel of one-class SVM, the widely approved radial basis function kernel is employed. As to $k$-NN, we set the parameter $k = 30$, which is larger enough compared with the size of any anomaly cluster. As to AdaGUM and AdaGUM_CL, we set $\theta = 0.8$. And the feature graph is updated once when every 500 sensor data are detected. As to AdaGUM and AdaGUM_CL, we focus on the average TPR and FPR within each updating cycle. As the performance comparison shown in Figure 6, AdaGUM outperforms the other three methods because when there exist the data of unknown patterns, AdaGUM and AdaGUM_CL can report the data to the cloud computing center where the data can be further analyzed and processed; however, AdaGUM_CL cannot identify the newly emerging patterns within a updating cycle until the feature graphs are updated. As to the other two baseline methods, which are based on the static model generated by learning from the train set, they inevitably fail to identify the data of unknown patterns.

Input: $\delta$-the threshold of feature importance
$t_0$-the latest updating time
$T_I$-the time interval of updating
$C_{\text{Unk}}$-the collection of the data whose patterns are unknown
$G_{\text{Ano}}$-the feature graph of anomalous patterns
$G_{\text{Nor}}$-the feature graph of normal patterns
Output: $G_{\text{Ano}}'$-the updated graph of anomalous patterns
$G_{\text{Nor}}'$-the updated graph of normal patterns

(1) $G_{\text{Ano}}' \leftarrow \phi$;
(2) $G_{\text{Nor}}' \leftarrow \phi$;
(3) result$\leftarrow$DeepLearning$(C_{\text{Unk}})$;
(4) Distribute2Cache$(G_{\text{Ano}}', G_{\text{Nor}}')$;
(5) **while** ((GetCurrentTime()-$t_0$) $\geq T_I$)
(6) **for** (each node $n$ in $G_{\text{Ano}}$)
(7) {**if** (Calculate$(n) \leq \delta$)
(8) {Remove$(n)$; $G_{\text{Ano}} \leftarrow G_{\text{Ano}} - n$; }
(9) **for** (each node $n'$ in $G_{Nor}$)
(10) **if** (Calculate$(n') \leq \delta$)
(11) {Remove$(n')$; $G_{\text{Ano}} \leftarrow G_{\text{Ano}} - n'$; }
(12) }
(13) $G_{\text{Ano}}' \leftarrow G_{\text{Ano}} \bigcup G_{\text{Ano}}'$;
(14) $G_{\text{Nor}}' \leftarrow G_{\text{Nor}} \bigcup G_{\text{Nor}}'$;
(15) Distribut2Edge$(G_{\text{Ano}}', G_{\text{Nor}}')$;
(16) **return**$G_{\text{Ano}}', G_{\text{Nor}}'$;

Algorithm 2: Anomaly detection on edge nodes.

Table 1: The experimental datasets.

| Datasets | Instances | Dimensions | Anomaly |
|---|---|---|---|
| BaIoT | 7062606 | 115 | Attack |
| El Nino | 178080 | 12 | Temperature and humility |
| Mulcross | 20000 | 20 | Data |

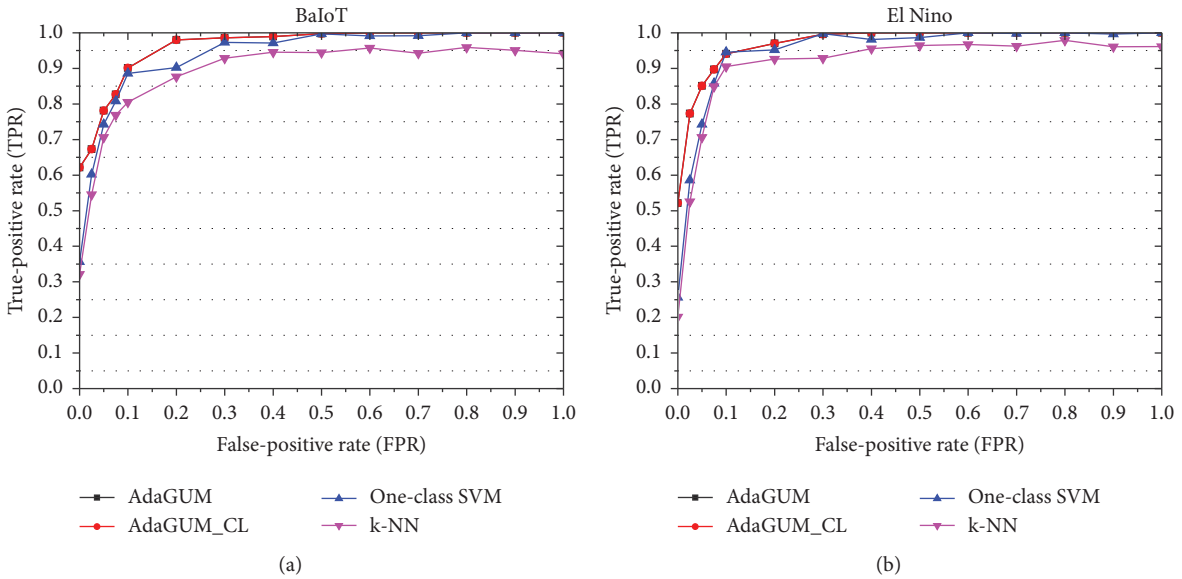


Figure 5: Performance comparison for anomaly detection. (a) Anomaly detection on BaIoT. (b) Anomaly detection on El Nino.

*4.4.3. Comparison of the Cumulative Average Detection Time.* In this experiment, the data generated by Mulcross are separated into 20 groups each of which involves 1000 data. And the cumulative average detection time of AdaGUM and AdaGUM_CL are compared. As to the parameter settings, we set $\theta = 0.8$. And the feature graph is updated once when

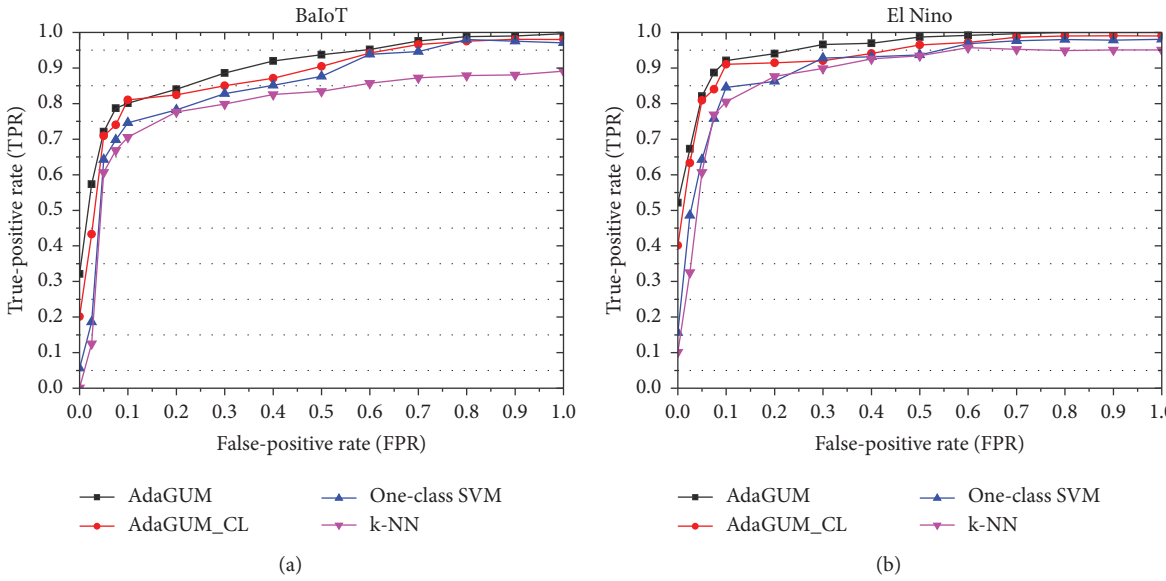


Figure 6: Performance comparison for anomaly detection. (a) Anomaly detection on BaIoT. (b) Anomaly detection on El Nino.

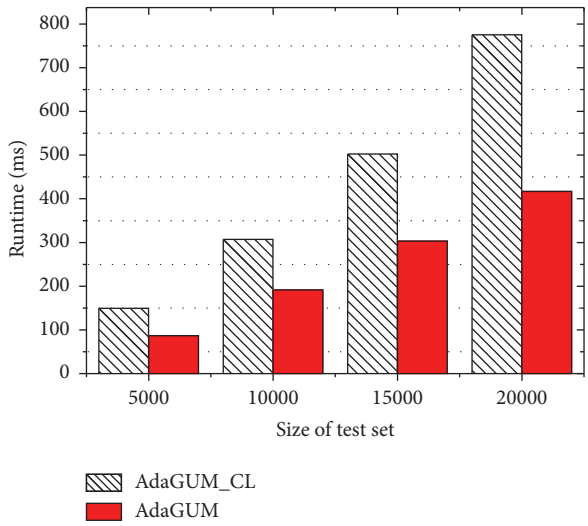


Figure 7: Comparison of total time consumption.

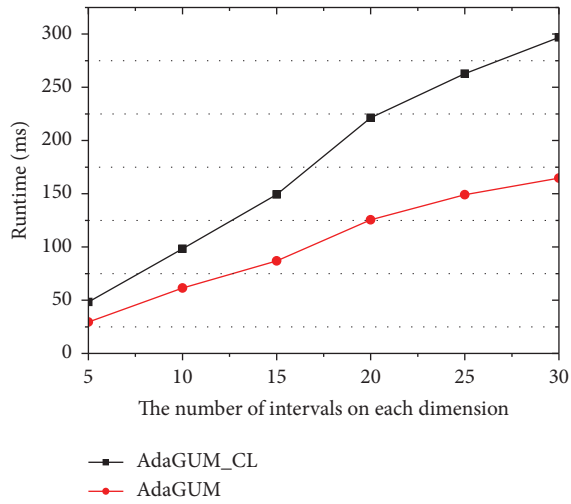


Figure 8: Comparison of total time consumption.

every 500 sensor data are detected. As the performance comparison shown in Figure 7, AdaGUM outperforms AdaGUM_CL; this is mainly because AdaGUM migrates part of the anomaly detection tasks to the edge nodes which reduces the considerable transmission cost and relatively long latency. Other than AdaGUM, AdaGUM_CL performs all the detection tasks at the cloud computing center, which degrades the detection efficiency, especially for delay-sensitive tasks.

*4.4.4. Detection Performance under Different Number of Intervals.* In the following experiment, the influence on the runtime of both adaGUM and adaGUM_CL, which is caused by the changes of intervals on each dimension, is tested. Then, the data generated by Mulcross are separated into 20 groups, each of which involves 500 data, and the cumulative average detection time of AdaGUM and AdaGUM_CL are compared. As to the parameter settings, we set $\theta = 0.8$. And the feature graph is updated once when every 1000 sensor data are detected. As the runtime shown in Figure 8, the runtime of AdaGUM and AdaGUM_CL both increase approximately in a linear manner, and AdaGUM still outperforms AdaGUM_CL due to its advanced detection strategy for edge computing.

## 5. Conclusion and Future Work

In this study, we propose AdaGUM, an anomaly detection method based on the adaptive graph updating model for edge computing. With feature graphs, both anomalous patterns and normal patterns can be explicitly recorded, and the feature graph updating strategy ensures that the features of newly emerging patterns can be integrated with the

feature graph, and the features of obsoleted patterns can be removed from the feature graph. Different from the cloud computing-based anomaly detection methods, the proposed method has the following advantages:

(1) In AdaGUM, a part of the computing tasks is migrated from the cloud computing center to edge nodes where the sensor data can be detected in real-time. Then, the pressure on data processing and data transmission can be alleviated.

(2) On each edge node, the sensor data collected from IoT devices are not only compared with the known patterns recorded in the feature graph but also compared with the newly emerging patterns recorded in the lists, which ensure the detection accuracy.

(3) At the cloud computing center, with its powerful abilities of data computing and data processing, the reported unknown patterns can be analyzed and identified by intelligent machine learning methods; then, the processing results are distributed to all edge nodes which realizes the real-time detection.

Actually, in the proposed model, we focus on the cooperation between the cloud computing center and the edge node rather than the cooperations between the edge nodes. As to our future work, on the one hand, we will take the cooperations between the edge nodes into account, whereby the resources on edge nodes can be further fully utilized. On the other hand, we will try to find an efficient way of the fusion between the existing graph and a small part of newly emerged patterns.

## Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## Acknowledgments

This work was supported by the National Natural Science Foundation of China (U20B2046 and 61976149), the National Key Research and Development Plan (2018YFB0803504), Guangdong Province Universities and Colleges Pearl River Scholar Funded Scheme (2019), Heilongjiang Provincial Natural Science Foundation of China (F2018002), the Natural Science Foundation of Zhejiang Province(LZ20F020002), the Research Project of Public Technology (LGF19F020009) the Cultivating Science Foundation of Taizhou University (2019PY014 and 2019PY015), and the Science and Technology Project of Taizhou (2003gy15 and 20ny13).

## References

[1] Q. Zhang, Y. Hu, C. Ji et al., "Edge computing application: real-time anomaly detection algorithm for sensing data," *Journal of Computer Research and Development*, vol. 55, no. 3, pp. 524–536, 2018.

[2] Z. Tian, W. Shi, and Y. Wang, "Real time lateral movement detection based on evidence reasoning network for edge computing environment," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4285–4294, 2019.

[3] M. Shafiq, Z. Tian, A. K. Bashir et al., "CorrAUC: a malicious bot-IoT traffic detection method in IoT network using machine learning techniques," *IEEE Internet of Things Journal*, vol. 57, 2020.

[4] P. Y. Chen, S. Yang, and J. A. McCann, "Distributed real-time anomaly detection in networked industrial sensing systems," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 6, pp. 3832–3842, 2015.

[5] J. Zhao, K. Liu, W. Wang, and Y. Liu, "Adaptive fuzzy clustering based anomaly data detection in energy system of steel industry," *Information Sciences*, vol. 259, no. 3, pp. 335–345, 2014.

[6] S. A. Shaikh and H. Kitagawa, "Efficient distance-based outlier detection on uncertain datasets of Gaussian distribution," *World Wide Web*, vol. 17, no. 4, pp. 511–538, 2014.

[7] F. T. Liu, K. M. Ting, and Z. H. Zhou, "Isolation-based anomaly detection," *ACM Transactions on Knowledge Discovery from Data*, vol. 6, no. 1, pp. 74–77, 2012.

[8] J. Ma, L. Sun, H. Wang et al., "Supervised anomaly detection in uncertain pseudoperiodic data streams," *ACM Transactions on Internet Technology*, vol. 16, no. 1, pp. 1–20, 2016.

[9] X. Du, "QoS routing based on multi-class nodes for mobile ad hoc networks," *Ad Hoc Networks*, vol. 2, no. 3, pp. 241–254, 2004.

[10] F. Angiulli and F. Dolphin, "An efficient algorithm for mining distance-based outliers in very large datasets," *ACM Transactions on Knowledge Discovery from Data(TKDD)*, vol. 3, no. 1, pp. 777–781, 2009.

[11] Y. Duan, X. Fu, B. Luo et al., "Detective: automatically identify and analyze malware processes in forensic scenarios via DLLs," in *Proceedings of the IEEE ICC 2015*, London, UK, June 2015.

[12] M. Shen, B. Ma, L. Zhu, R. Mijumbi, X. Du, and J. Hu, "Cloud-based approximate constrained shortest distance queries over encrypted graphs with privacy protection," *IEEE Transactions on Information Forensics & Security*, vol. 13, no. 4, pp. 940–953, 2018.

[13] M. Zaharia, M. Chowdhury, M. J. Franklin et al., "Spark: cluster computing with working sets," *HotCloud*, vol. 15, no. 1, pp. 10–17, 2010.

[14] K. Zhang, M. Hutter, and H. Jin, "A new local distance-based outlier detection approach for scattered real-world data," in *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 813–822, Singapore, May 2009.

[15] W. Jin, A. K. Tung, J. Han et al., "Ranking outliers using symmetric neighborhood relationship," in *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, no. 1, pp. 577–593, Singapore, May 2006.

[16] M. Armbrust, A. Fox, R. Griffith et al., "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[17] J. Ma, L. Sun, H. Wang et al., "Supervised anomaly detection in uncertain pseudoperiodic data streams," *ACM Transactions on Internet Technology*, vol. 16, no. 1, pp. 1–20, 2016.

[18] H. P. Kriegel, P. Kröger, E. Schubert et al., "Dolphin: loop: local outlier probabilities," in *Proceedings of ACM Conference*

*on Information and Knowledge Management*, pp. 1649–1652, Hong Kong, China, November 2009.

[19] Z. Peng, P. Gurram, H. Kwon et al., "Sparse kernel learning-based feature selection for anomaly detection," *IEEE Trans on Aerospace and Electronic Systems*, vol. 51, no. 3, pp. 1698–1716, 2015.

[20] H. Huang, H. Qin, S. Yoo, and D. Yu, "Physics-based anomaly detection defined on manifold space," *ACM Transactions on Knowledge Discovery from Data*, vol. 9, no. 2, pp. 1–39, 2014.

[21] D. Mandala, F. Dai, X. Du et al., "Load balance and energy efficient data gathering in wireless sensor networks," in *Proceedings of the 1st IEEE International Workshop on Intelligent Systems Techniques for Wireless Sensor Networks, in Conjunction with IEEE MASS'06*, Vancouver, Canada, October 2006.

[22] K. Khedo, R. Doomun, and S. A. Reada, "Redundancy elimination for accurate data aggregation in wireless sensor networks," *Wireless Sensor Network*, vol. 2, no. 4, pp. 300–308, 2010.

[23] L. Cai, N. F. Thornhill, S. Kuenzel et al., "Real-time detection of power system disturbances based on k-nearest neighbor analysis," *IEEE Access*, vol. 5, no. 3, pp. 5631–5639, 2017.

[24] Y. Wang, Z. Tian, Y. Sun et al., "An IBN-based location privacy preserving scheme for IoCV," *IEEE Transactions on Intelligent Transportation Systems*, vol. 1, 2020.

[25] F. Jiang, Y. Fu, B. B. Gupta et al., "Deep learning based multichannel intelligent attack detection for data security," *IEEE Transactions on Sustainable Computing*, vol. 5, no. 2, 2018.

[26] P. Dong, X. Du, H. Zhang, and T. Xu, "A detection method for a novel DDoS attack against SDN controllers by vast new low-traffic flows," in *Proceedings of the IEEE ICC 2016*, May 2016.

[27] L. Xiao, X. Wan, C. Dai, X. Du, X. Chen, and M. Guizani, "Security in mobile edge caching with reinforcement learning," *IEEE Wireless Communications*, vol. 25, no. 3, pp. 116–122, 2018.

[28] X. Du and F. Lin, "Improving sensor network performance by deploying mobile sensors," in *Proceedings of the 24th IEEE International Performance, Computing, and Communications Conference (IPCCC)*, April 2005.

[29] D. Alassi and F. Alhajj, "Effectiveness of template detection on noise reduction and websites summarization," *Information Sciences*, vol. 219, pp. 41–72, 2013.

[30] S. Papadimitriou, H. Kitagawa, P. Gibbons et al., "Loci: fast outlier detection using the local correlation integral," in *Proceedings of the International Conference on Data Engineering*, no. 1, pp. 315–326, IEEE Press, Bangalore, India, March 2003.

[31] B. Schlkopf, J. Platt, J. Shawe-Taylor, and A. Smola, "Estimating the support of a high-dimensional distribution," *Neural Computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[32] F. Wei, "Paradigm shift from cloud computing to fog computing and edge computing," *Journal of Nanjing University of Information Science & Technology*, vol. 8, no. 5, pp. 404–414, 2016.

[33] D. Cai, X. He, J. Han, and T. S. Huang, "Graph regularized nonnegative matrix factorization for data representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 8, pp. 1548–1560, 2011.

[34] F. Ulah, M. Edwards, R. Ramdhany et al., "Data exfiltration: a review of external attack vectors and countermeasures," *Journal of Network and Computer Applications*, vol. 101, pp. 18–54, 2017.

[35] W. Zhang, B. Zhang, Y. Zhou, H. He, and Z. Ding, "An IoT honeynet based on multi-port honeypots for capturing IoT attacks," *IEEE Internet of Things Journal*, 2019.

[36] K. Shvacchko, H. Kuang, S. Radia et al., "The hadoop distributed file system,"in *Proceedings of the 26th Sympsium on Mass Storage Systems and Technologies(MSST),* vol. 1, pp. 1–10, IEEE, Incline Village, NV, USA, June 2010.

[37] W. Zhang, H. Wang, H. He, and P. Liu, "Detecting android malware by ORGB analysis," *IEEE Transactions on Reliability*, 2019.

[38] Z. Tian, C. Luo, J. Qiu et al., "A distributed deep learning system for web attack detection on edge devices," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 1963–1971, 2020.

[39] J. Qiu, Z. Tian, C. Du et al., "A survey on access control in the age of Internet of Things," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4682–4696, 2020.

[40] M. Shafiq, Z. Tian, A. Bashir et al., "IoT malicious traffic identification using wrapper-based feature selection mechanisms," *Computers & Security*, vol. 94, Article ID 101863, 2020.

[41] V. Bui, V. H. Nguyen, T. L. Pham et al., "RNN-based Deep Learning for One-Hour Ahead Load Forecasting," in *Proceedings of the 20202 International Conference on Artificial Intelligence in Information and Communication(ICAIIC)*, pp. 587–589, IEEE, Fukuoka, Japan, April 2020.

[42] W. Kong, Z. Y. Dong, Y. Jia et al., "Short-term residential load forecasting based on LSTM recurrent neural network," *IEEE Transactions on Smart Grid*, vol. 10, no. 1, pp. 841–851, 2017.