Hindawi
Mobile Information Systems
Volume 2020, Article ID 6633252, 16 pages
https://doi.org/10.1155/2020/6633252

Research Article

A Deep Random Forest Model on Spark for Network Intrusion Detection

Zhenpeng Liu,[1,2] **Nan Su**,[1] **Yiwen Qin**,[3] **Jiahuan Lu**,[1] **and Xiaofei Li**[2]

[1]*School of Cyber Security and Computer, Hebei University, Baoding, China*
[2]*Information Technology Center, Hebei University, Baoding, China*
[3]*School of Electric and Information Engineering, Lanzhou Jiaotong University, Lanzhou, China*

Correspondence should be addressed to Xiaofei Li; lixiaofei@hbu.edu.cn

Received 12 October 2020; Revised 6 December 2020; Accepted 11 December 2020; Published 22 December 2020

Academic Editor: Salvatore Carta

This paper focuses on an important research problem of cyberspace security. As an active defense technology, intrusion detection plays an important role in the field of network security. Traditional intrusion detection technologies have problems such as low accuracy, low detection efficiency, and time consuming. The shallow structure of machine learning has been unable to respond in time. To solve these problems, the deep learning-based method has been studied to improve intrusion detection. The advantage of deep learning is that it has a strong learning ability for features and can handle very complex data. Therefore, we propose a deep random forest-based network intrusion detection model. The first stage uses a slide window to segment original features into many small pieces and then trains a random forest to generate the concatenated class vector as rerepresentation. The vector will be used to train the multilevel cascade parallel random forest in the second stage. Finally, the classification of the original data is determined by voting strategy after the last layer of cascade. Meanwhile, the model is deployed in Spark environment and optimizes cache replacement strategy of RDDs by efficiency sorting and partition integrity check. The experiment results indicate that the proposed method can effectively detect anomaly network behaviors, with high F1-measure scores and high accuracy. The results also show that it can cut down the average execution time on different scaled clusters.

1. Introduction

The rapid development of cloud computing, edge computing, and 5G technologies have widely infiltrated our politics, economy, culture, and other aspects of life. The massive data generated from these everyday scenarios will boost more valuable output from big data; meanwhile, these extensive applications could make the prospect of big data complicated and unsafe. Considering the complexity, high dimension, heterogeneity, and processing speed of large data volumes, potential risks exist not only in the system architecture but also in the data itself. Most traditional protection solutions can no longer satisfy the requirements in big data environment because the distributed data source makes it difficult to define the boundaries of the dataset, which will threaten the authenticity of the data being analyzed.

Outliers are also known as anomalies or deviants in data mining and statistical analysis. In cyberspace security, outlier detection is a process to analyze suspects whose key-value or behavior pattern is significantly different from the normal objects. Detection algorithm recognizes the abnormalities and then cleans the confirmed data to ensure data security. Outlier detection is now a hot topic in academia and industry. For example, anomaly spots that appear in magnetic resonance imaging or other types of medical diagnosis devices typically indicate disease conditions, and the outlier records in product payment from unusual locations or frequent large transactions would help detect credit card fraud in financial situations. Other examples are rumor detection in social media and congestion detection in urban traffic management [1–3]. Among these challenges, network intrusion detection is critical for cyberspace security [4].

Network intrusion detection is a technology designed and configured to ensure the security of computer systems that can detect violations of security policies in computer networks. The combination of software and hardware for intrusion detection is an intrusion detection system (IDS). An intrusion detection system (IDS) is aimed to analyze the network traffic or the activity of a single machine in order to discover nonauthorized activities. Such activities can be originated by a malware or can be related to a human attack operated locally or remotely [5]. There are already many machine learning algorithms that have been widely used to identify outliers in networks [6–8]. But most conventional methods are mainly limited with unacceptable accuracy in detection when network data are often complex and high-dimensional, such as back propagation (BP), support vector machine (SVM), and random forest (RF) [9]; the accuracy of the UNSW-NB15 dataset does not exceed 91% [10]. That reveals that the shallow structure of machine learning has been vulnerable to respond. Despite the fact that intrusion detection is a key issue, research studies on the methods which combine deep learning and machine learning are still insufficient [11]. Though deep neural networks are powerful, they are very complicated with too many hyperparameters, and the learning performance depends seriously on the careful tuning of these hyperparameters, so the training costs are huge. The above scenarios usually make the training of deep neural networks very hard; sometimes, it is like an art rather than engineering. This inspires us to explore other deep learning structures for network anomaly detection.

Ensemble learning is an important approach of machine learning, and random forest is one of the classic algorithms in ensemble learning. It fits to high-dimensional data and has only a few parameters, and the training of the RF is not complicated. So, we consider using the forest as a layer to replace the neurons in our deep network structure. Besides, the training of each decision tree in the random forest is independent, which is natural for parallel deployment. Each layer of this deep forest structure can be deployed in parallel to speed up the training process.

In order to reduce the obstacles caused by the numerous parameters of the present deep learning-based method in intrusion detection and to further improve the classification accuracy and scalability, this paper proposes a detection model based on feature segmentation and deep structure of parallelized random forest (FS-DPRF). The main contributions of this paper are as follows.

(1) A deep cascade structure of random forest is proposed, and each layer is parallelized to improve the accuracy and scalability and to fit for massive data in detection task. Various types of attacks can be classified.

(2) A slide window is introduced to segment the high-dimensional features into small size feature vectors for training, which can reduce the calculation volume of each computing and keep the integrity of original information.

(3) Compared with the classic parallel random forest in Spark, the approach optimizes the replacement for RDD loaded in memory with efficiency sorting and partition integrity check, which can improve cluster task execution efficiency.

The performance of the proposed model is verified and compared with other algorithms in four network intrusion datasets, and the experiment results fully prove the effectiveness of the proposed model on network anomaly detection.

The remainder of the paper is organized as follows. Section 2 reviews the related work. In Section 3, we illustrate the detection model designed for intrusion detection. Section 4 introduces the memory optimization designed for model parallelization. Evaluate the model with a series of experiments in Section 5. Finally, the conclusion of this paper is presented in Section 6.

2. Related Work

At present, many scholars have studied the intrusion detection issue. A recent survey by Buczak and Guven [12] made a comprehensive review of the current data mining and machine learning methodologies of intrusion detection; the survey described the strengths and weaknesses of the algorithm and provided a clear outlook for future work. The classic algorithms can be categorized into artificial neural networks [13, 14], clustering-based, Bayesian network, ensemble learning [15], SVM-based [16], and hidden Markov models (HMMs). Khalvati et al. [17] proposed the SVM hybrid learning (distance sum-based SVM, DSSVM) method. In DSSVM, the distance sum is calculated based on the correlation between each data sample and the clustering feature dimension obtained from the dataset, and then, the SVM is used for classification and has a high detection rate. Vinayakumar et al. [18] used the convolutional neural network (CNN) for network intrusion detection, the research models network traffic as a time series, and then used supervised learning methods to model TCP/IP protocol packets within a predefined time range. The effectiveness of this network structure in intrusion detection has been proved on the KDD99 dataset. Potluri and Diedrich [19] proposed an accelerated deep neural network (A-DNN) structure, and it is used to identify anomalies in network data and process them with an accelerator platform. Experimental results show that this method is feasible and effective in NSL-KDD. Gao et al. [20] introduced a deep belief network into the field of anomaly detection. A multilayered Boltzmann machine is used to form a neural network classifier. When using deep belief networks in comparison with SVM on the KDD99 dataset, the former shows a better performance. Dominguez et al. [21] evaluated unsupervised algorithms from various research fields by doing lots of comparative experiments and unsupervised feature learning works in most cases, but still lack interpretability and require manual analysis. Hundman et al. [22] proposed a model based on LSTM and a novel dynamic threshold approach. The model does not rely on scarce labels or false parametric assumptions to deal with time series data and achieves high accuracy with good interpretability. Manzoor et al. [23] introduced a density-based ensemble method for feature-evolving streams problem, which measures outliers at multiple scales or granularities and especially works well in high noise environments.

In recent years, the stacking method [24] and boosting method have become popular in ensemble learning. Liu et al. [25] proposed the isolation forest algorithm to establish an anomaly index based on the path length from leaf node to root node. The detection effect of global outliers is good, but it is weak at dealing with local sparse points. The gradient boosting decision tree (GBDT) proposed by Friedman [26] generates a prediction model in the form of a set of basic learners and combines the basic learners into a strong learner through iteration. Each time the model is established, the gradient descent direction of the loss function of the model would be established first. In successive iterations, the residual is continuously reduced to produce a vertically deepened tree. It has the advantages of high prediction accuracy and strong robustness to outliers. Chen and Guestrin [27] put forward a scalable tree boosting system (XGBoost); its main idea is also boosting according to the negative gradient direction of the loss function. The biggest difference is that the empirical error was expanded by second-order Taylor expansion, and some regular items were added, which make loss function scalable, and have a high precision and a good fitting effect. But there were too many hyperparameters which make classification quite dependent on the tuning result. Su et al. [28] proposed an intrusion detection method using the XGBoost algorithm on an unbalanced dataset; it uses the improved SMOTE algorithm to oversample the minority samples and downsample majority samples. The method is based on the premise of changing the original feature distribution of the data, which not only increases the calculation burden of the model but also easily loses some important information in the sample and affects the final detection performance. Farnaaz and Jabbar [8] proposed to use the random forest algorithm to detect various types of attacks and verify the model on NSL-KDD data. The results prove that the detection accuracy of DOS, PROBE, U2R, and R2L is improved, but the capability of feature processing is weak. In the latest research, Roberto et al. [5] proposed a probabilistic-driven ensemble model (PDE) that uses logistic regression algorithms to evaluate the effect of ensemble learning classifiers. The model excludes predictors with lower probabilities from the classification process and combines the most effective algorithms by weighted probability criteria. Experiments on the NSL-KDD dataset show that the PDE has a high performance in detecting intrusions. Zhou et al. [29] proposed a novel ensemble system based on the modified AdaBoost with the area under the curve (M-AdaBoost-A) algorithm. Strategies such as SMV and PSO are applied to combine multiple M-AdaBoost-A based classifiers. It shows better performances for two intrusion detection issues: 802.11 wireless networks and traditional enterprise networks, but it lacks evaluation of model time consumption. Khan et al. [30] proposed a deep learning model (TSDL) based on stacked autoencoder with a soft-max classifier. Their deep learning model works in a cascade manner; the model uses a probability score value as an additional feature in the final decision stage in order to detect the normal state and other classes of attacks. TSDL has achieved impressive results in the accuracy of multiclass detection on UNSW-NB15 and KDD99.

The deep learning model usually has a good performance, but it has too many complicated hyperparameters to be adjusted. In most cases, it seems difficult to have a good performance with less complexity. To solve the problem, the model we propose introduces a sliding window and a deep structure into random forests to enhance the diversity of decision trees, thereby improving the generalization ability of ensemble learning and the accuracy in network intrusion detection, also with much fewer parameters. At the same time, our method optimizes the data cache replacement of RDDs on the Spark cluster and cuts down the execution time of detection tasks.

2.1. Algorithm Selection Criteria. The selection of the algorithm needs to refer to the IDS architecture, which can be divided into a centralized structure and a distributed structure.

Most IDS algorithms use a single-machine centralized structure, that is, data collection and analysis are performed on a host. This method performs detection based on host audit data. The centralized structure has the advantages of simple structure and easy implementation. The disadvantage is that the processing time is slow. So, it is suitable for small network systems.

Distributed structure comprises hierarchical structure and collaborative structure. The hierarchical structure is a tree-type hierarchical system, like the proposed model in this paper, and it combines the simplification of a centralized structure and the robustness of a distributed structure. The distributed structure also makes the detection time faster, which is suitable for larger-scale network systems.

3. Model Description

The proposed detection model is described as feature segmentation, deep parallel random forest, and voting strategy. Feature segment is the first stage of the model, which segments original features to reduce the calculation volume of high-dimensional data in a single compute and generates a concatenated class vector as a new representation. In the second stage, the concatenated class vector will be used to train deep parallel random forest which predicts a probability distribution of original data type. Finally, the voting strategy after the last layer of cascade will confirm the outlier. Figure 1 shows the overview of the FS-DPRF model.

3.1. Feature Segment. The first stage in the model simplifies the original data features which are shown in Figure 2, by using a slide window to segment features into many same sized feature vectors; the data dimension of each feature vector is less than the original feature, and it reduces the

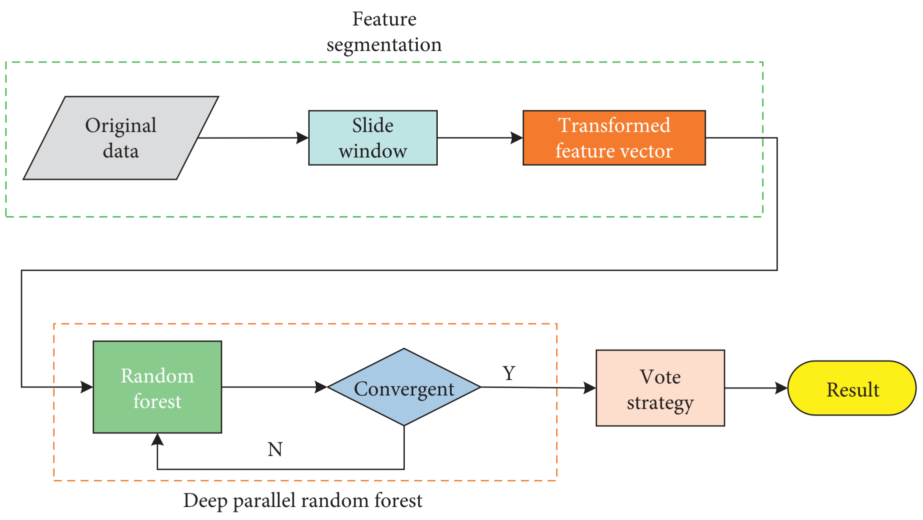

FIGURE 1: Overview of the FS-DPRF model. Slide window is created to reduce the dimension of the data being processed.

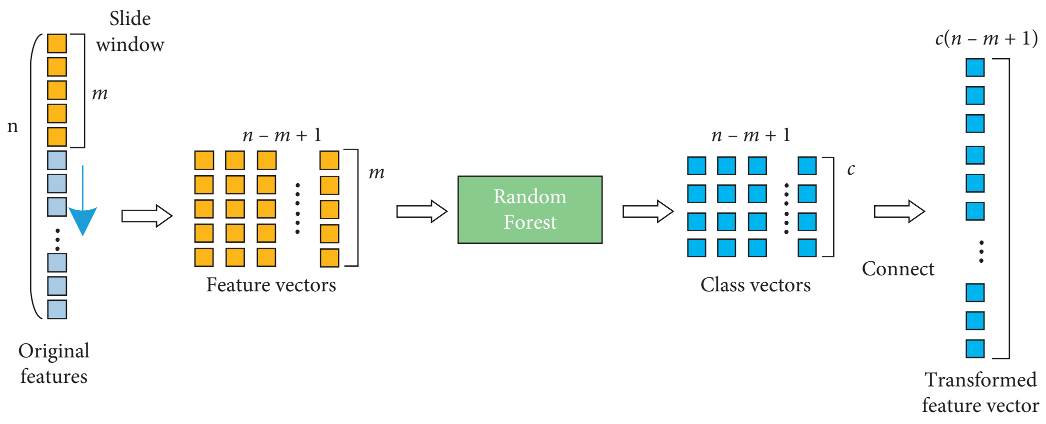

FIGURE 2: Illustration of original feature split. The original features will be divided into many same sized feature vectors by the sliding window where the dimension is reduced in this part.

calculation volume during every single compute in random forest. Assume that a linear feature vector of length is n, the window length of a feature slice is m, and each time slides 1 unit length, $n-m+1$ m-dimensional feature vectors will be generated. Suppose that there is a detection task that contains c categories, after feature processing, a linear feature vector of length n will generate a new feature vector of length $c\ (n-m+1)$. Similarly, for an image data, feature segment will generate a new feature vector of length $c\ (n-m+1)^2$. For instance, there is an intrusion sample data including 40 features, and there are four types of attacks such as DOS (denial of service), R2L (remote to local), U2R (user to root), and PROBE (surveillance and probing). And the slice window size is set to 10. Then, there will be a total of 31 feature vectors where each one is 10-dimensional.

After that, each feature vector will sequentially be put into a single-layer random forest, and then, class probability vectors [31] will be generated. A detailed explanation of the generation process of the class probability vector is depicted in Figure 3.

The entropy of feature vector will be calculated by the Gini index before node split. The Gini index is a model for calculating the entropy defined in the following equation.

$$\text{Gini}\,(t) = 1 - \sum_{k=1}^{K} \left[p\left(C_k | t\right)\right]^2, \tag{1}$$

where t is the target split node, and $p(C_k|t)$ represents the probability that node t belongs to class C_k.

The class probability is derived from the group of values that eventually fall on the leaf node and then averages the predictions of all decision trees in the forest to get the output class vector. The 31 feature vectors before will transform into 31 class vectors in which each one is 4-dimensional. Finally, as shown in Figure 2, all class vectors will connect to form a rerepresented feature vector as an enhanced representation corresponding to original data features. And the new feature will be used as input to train the cascade random forest in the next stage.

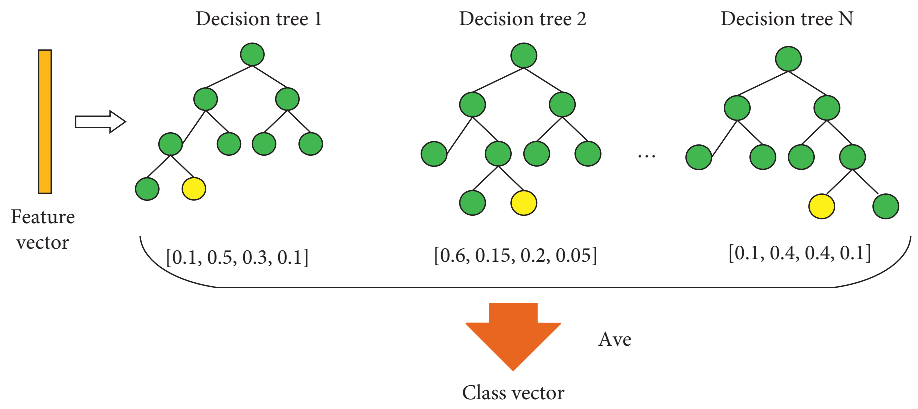

FIGURE 3: Example of a class vector generation in a random forest. The rectangle on the left represents an instance of input feature vector, and the yellow leaf is a probability distribution of each independent decision tree.

3.2. Deep Parallel Random Forest. The parallel random forest forms the deep forest structure by cascade stacking. Each new layer in the cascade structure concatenates the rerepresented feature vector and the class vector of the previous layer as input. Specifically, each layer of the cascade PRF will count the prediction results of all decision trees on input samples and generates the probability of different class distribution, as a class vector. Subsequently, the class probability vector will connect with the transformed feature which is formed by feature segmentation to train the next layer. For example, the rerepresented feature vector in the first stage will be input to train the cascade random forest. The first layer of the cascade will output a 4-dimensional class vector according to the previous assumption; then, it will connect with the input feature vector to train the next layer and so on. The structure of the deep parallel random forest is shown in Figure 4. Compared with the parallel random forest, the cascade PRF can improve the generalization ability of ensemble learning. It is worth noting that each time a new cascade layer is expanded, the cascade structure will randomly extract 80% of the training set for growing and the remaining 20% as the validation set to verify the performance gain of the new cascade layer. When the performance improvement is lower than the threshold, the training process will be terminated automatically and the number of cascaded PRF layers will be finally determined.

3.3. Voting Strategy. In ensemble learning, individual learners will output the final prediction after combining the independent judgment through the voting method. For an actual outlier detection task, it can be simplified as an anomaly classification task and identifies outliers by using voting strategy. The prediction of the last layer in cascade PRF will be the final result where the output classes of all decision trees in the last layer are counted, and then, the decision is made by using voting strategy based on the probability distribution. Majority voting is used in anomaly detection tasks with high reliability requirements. If a sample receives more than half of the votes, it is predicted as an outlier, otherwise it is rejected. However, if the task prediction result is necessary, the majority voting method will degenerate to the plurality voting method; in this condition, if many prediction results get the same votes at the time, one would be selected. The majority voting and plurality voting are defined as

$$H(x) = \begin{cases} c_j, & \text{if } \sum_{i=1}^{T} h_i^j(x) > 0.5 \sum_{k=1}^{N} \sum_{i=1}^{T} h_i^k(x), \\ \text{reject}, & \text{otherwise}, \end{cases} \tag{2}$$

$$H(x) = c_{\operatorname{argmax}_j \sum_{i=1}^{T} h_i^j(x)}, \tag{3}$$

where h_i represents the decision tree$_i$, and T represents the number of decision tree in forest. N is the dimension of probability vector. c_j is the one of the class labels in collection $\{c_1, c_2, c_3 \ldots c_N\}$. The basic learner h_i will make a prediction which belongs to the set of class labels $\{c_1, c_2, c_3 \ldots c_N\}$, and probability distribution of h_i on the sample x is an N-dimensional vector $(h_i^1(x), h_i^2(x), h_i^3, \ldots, h_i^N(x))$, where $h_i^j(x)$ is the probability output of h_i on class label c_j.

The detailed steps of FS-DPRF are described in Algorithm 1.

4. Parallelization on Spark

Spark is a distributed computing framework developed by UC Berkeley AMP Lab. Spark supports a variety of ways to combine with other big data platforms, which enables it to process large-scale data efficiently. Its memory-based Resilient Distributed Dataset (RDD) mechanism allows data intermediately cached in memory [32], saving a lot of I/O operation overhead, and is well qualified for iterative and ensemble algorithms. The framework has unique advantages in processing. Each decision tree of RF is built independently of each other, and each subnode of decision tree is also split independently. The structures of the FS-DPRF model and decision-tree based forest enable the computing tasks have natural parallelism [33]. However, the training data in the parallel random forest generation process requires multiple iterations, and a large number of RDD data blocks need to be reused in the iteration until the convergence are met. Spark's default least recently used replacement algorithm (LRU)

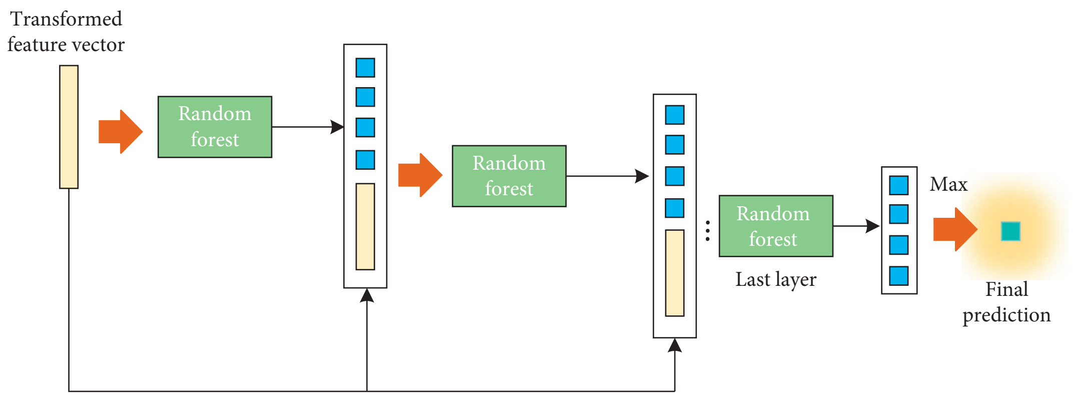

FIGURE 4: The architecture of deep PRF, the number of layers, is automatically determined based on the calculated performance gain, and the training process will be terminated if improvement is lower than the threshold.

cannot cope with our model's requirement on the reuse of RDD data block because it could easily swap high-reuse block out of the cache, causing inefficiency job execution [34]. Based on these facts, a cache hierarchical replacement optimization for RDD objects is presented, which can effectively improve the cluster execution efficiency during the process of building FS-DPRF.

4.1. High Reuse Caching. First, Spark's cache mechanism assigns a cache manager to each worker to manage the RDDs and calculate the cache size. The RDD data size requires a storage which is no larger than the remaining memory. Otherwise, the replacement will be implemented.

$$S_i = \{S_{i1}, S_{i2}, S_{i3}, \dots, S_{ij}\},$$
$$\sum_{j=1}^{n_1} S_{1j} + \sum_{j=1}^{n_2} S_{2j} + \dots + \sum_{j=1}^{n_m} S_{mj} < S_{\text{cache}}, \tag{4}$$

where S_i represents the total size of all RDD$_i$ partitions, and S_{ij} is the size of the partition j of RDD$_i$. The computing cost between RDD partitions is another very important factor, which is defined as

$$CT_{ij} = ET_{ij} - ST_{ij}, \tag{5}$$

where ST_{ij} is the start time, and the ET_{ij} is the end time of each RDD partition; both are obtained by the partition dependency mechanism in Spark. Note that CT_{ij} already includes communication overhead. Thus, we can get the weight of each RDD, which is defined as follows:

$$W(R_{ij}) = \mu \cdot \frac{f(R_{ij}) \cdot CT_{ij}}{S_{ij}},$$
$$W(R_i) = \sum_{j=1}^{n} W(R_{ij}). \tag{6}$$

Here, $W(R_{ij})$ is the weight of partition j of RDD$_i$, and $W(R_i)$ is the weight of RDD$_i$. μ is an impact factor defined by a different work environment. $f(R_{ij})$ represents the usage count of partition j of RDD$_i$.

Second, processing time is linearly related to the size of the data block in general. The execution time of the RDD can be represented by the percentage of the RDD size which occupies the memory size in the Spark cluster environment.

$$T(R_{ij}) = \frac{S_{ij}}{S_{\text{cache}}}, \tag{7}$$

$$T(R_i) = \max\{T(R_{i1}), T(R_{i2}), \dots, T(R_{in})\}, \tag{8}$$

where $T(R_{ij})$ represents the execution time of partition R_{ij}. S_{ij} is the partition size of R_{ij}, and S_{cache} is the cluster memory. Since each partition of the RDD under the task set is executed in parallel, the total execution time of the RDD is the longest among all partitions. Finally, the execution efficiency of RDD can be quantified as the ratio of the weight value of RDD to the execution time, and $\varepsilon(R_i)$ is used to represent the execution efficiency of each RDD, which is defined in the following equation:

$$\varepsilon(R_i) = \frac{W(R_i)}{T(R_i)}. \tag{9}$$

The directed acyclic graph (DAG) of Spark divides the RDD stage and generates the RDD structure tree; we then calculate the execution efficiency of each RDD and cache high reused RDDs in the Map$_{\text{cacheList}}$ (rdd$_i$, ε). The steps of the high reuse cache method are described in Algorithm 2.

4.2. Hierarchical Replacement. Hierarchical replacement is the second step of optimization for parallel. It classifies the RDD target before the replacement, giving priority to incomplete RDDs. As it is shown in Figure 5, we design the IntegrityCheck function to verify RDD, and the function will check the partition and mark down the integrity in a collection where the flag records the partition status. If the partition of RDD is incomplete, it will be marked as FALSE and be replaced; otherwise, it will be marked as TRUE. Then, the RDD with less efficiency will be replaced according to Map$_{\text{cacheList}}$(rdd$_i$, ε). The process of hierarchical replacement is presented in Algorithm 3.

```
     Input: training dataset $D = \{(x_1, y_1), (x_2, y_2) \ldots (x_m, y_m)\}$;
       $x$: potential anomaly data.
     Output: $H(x)$: voting result of sample $x$;
       $C_{PRF}$: Deep random forests where $\{\mathrm{PRF}_i \mid i = 1, 2, \ldots, N\}$.
 (1) $C_{PRF} = \{\varnothing\}$
 (2) Initialize hyperparameters: tolerance $t$ and slice window size winSize
 (3) $D' =$ Feature Grained $(D)$; //$D'$ is newly generated feature vector.
 (4) do
 (5)   $i = 1$//layer $i$ of cascaded PRF.
 (6)   for $j = 1, 2, \ldots, T$ do
 (7)     $PRF_i = \{\varnothing\}$
 (8)     $D'j \longleftarrow$ Bootstrap sampling $(D')$
 (9)     $\mathrm{Tree}_j \longleftarrow$ decision tree $(D'_j)$
(10)     $PRF_i += \{\mathrm{Tree}_j\}$
(11) end for
(12) if (tolerance $\geq t$)
(13)     $C_{PRF} += \{\mathrm{PRF}_i\}$
(14) else
(15)     Break
(16) $i = i + 1$
(17) while (TRUE)
(18) $H(x) =$ voting method $(x)$//the last layer votes for classification
(19) Return CPRF
```

Algorithm 1: FS-DPRF.

```
    Input: $\mathrm{Tree}_{\mathrm{rdds}}$: RDD structure.
    Output: $\mathrm{Map}_{\mathrm{cacheList}}(\mathrm{rdd}_i, \varepsilon)$: Cache collection of RDD.
(1)   for ($i = 0$ to $\mathrm{Tree}_{\mathrm{rdds}}$.Length-1)
(2)     calc $\varepsilon(R_i)$ //Calculate RDD execution efficiency
(3)     if ($\varepsilon(R_i) > 1$)
(4)     $\mathrm{Map}_{\mathrm{cacheList}}(\mathrm{rdd}_i, \varepsilon)$.add$(\mathrm{Tree}_{\mathrm{rdds}}[i], \varepsilon)$
(5)   end if
(6) end for
```

Algorithm 2: High reuse cache.

5. Preliminary Assumptions and Hypotheses

There are three preliminary assumptions for the excellent performance of deep learning models:

(i) Layer-by-layer processing
(ii) Feature transformation
(iii) Sufficient model complexity

Traditional machine learning methods such as decision trees are processed layer by layer, but they lack sufficient complexity. The ensemble method can increase the complexity, such as random forest, but it is still not complex enough because there is no feature transformation process, and the processing is always performed in the same feature space. Therefore, our main hypothesis is that the feature segmentation and cascading structure can make the random forest increase the feature transformation ability and sufficient complexity on its original basis, thereby improving generalization ability. Another hypothesis is that the optimization of the spark-based RDD cache replacement strategy can reduce the training and detection time of the proposed model. The experiment requires the conversion of character features to numerical feature training models. The results and analysis in following subsections prove the hypothesis claims.

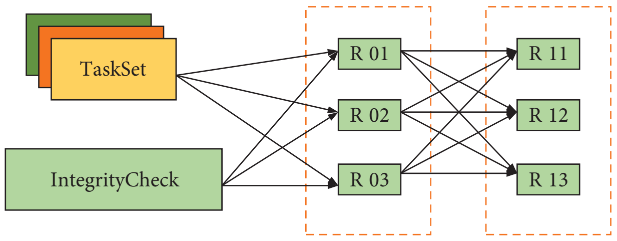

Figure 5: Detailed working principle of integrity checks function.

```
     Input: Map_cacheList (rdd_i, ε): cache collection of RDD, S_cache: cluster cache size, S_All: cached RDD size, R_new: cache candidate, S_new: size of candidate.
     Output: rdd CacheList
(1)  calc $\varepsilon(R_{new})$;
(2)  IntegrityCheck (rdds);
(3)  $Map_{rdds}(rdd_i, flag)$.add $(rdd_i, flag)$;
(4)  if $(S_{new} + S_{All} > S_{cache})$
(5)    for $((k,v) \longleftarrow Map_{rdds}(rdd_i, flag))$
(6)      if $(v == FALSE)$
(7)      Replace $(k = R_{new}, v = TRUE)$
(8)      end if
(9)    end for
(10)   if $(Map_{rdds}(rdd_i, flag)$.contains$(R_{new}))$
(11)     $Map_{rdds}(rdd_i, flag)$.key To List ( )
(12)   else
(13)     for $((k,v) \leftarrow Map_{cacheList}(rdd_i, \varepsilon))$
(14)     v.QuickSort ();//Sort by execution efficiency
(15)     if $(v < \varepsilon(R_{new}))$
(16)     Replace $(k = R_{new}, v = \varepsilon(R_{new}))$
(17)     end if
(18)   end for
(19)     $Map_{cacheList}(rdd_i, \varepsilon)$.key To List
(20)   end if
(21) end if
```

Algorithm 3: Hierarchical replacement.

6. Experiment

6.1. Dataset and Preprocessing. In order to evaluate the proposed model and report the experiment results, four intrusion datasets are selected, i.e., NSL-KDD [35], UNSW-NB15 [36], CICIDS2017, and CICIDS2018 [37].

NSL-KDD is an improvement of the KDD 99 dataset which was collected from a simulated US Air Force network environment over 9 weeks. The train set does not contain redundant records. In addition, there are no duplicate records in the test set, which makes the detection rate more accurate. Each piece of data contains 43 features including a label. The labels are divided into 5 classes, including attack and normal. The types of attacks are divided into four categories: DOS (denial of service attack), R2L (unauthorized access from the remote master), U2R (unauthorized local super user privileged access), and PROBE (port monitoring or scanning). Normal represents normal data.

The second dataset used in the experiment is UNSW-NB15. The dataset was collected in 2015 under the real network environment of Australian Center for Cyber Security (ACCS). The network traffic record contains true normal activity and attack behavior. The network record of this dataset contains 49 network features including a class label, and there are 10 types of network including normal behavior and 9 abnormal intrusion attacks. CICIDS2017 and CICIDS2018 datasets are the recent datasets that were developed by the Canadian Institute of Cyber Security. These two datasets are closer to the real network environment. The CICIDS2017 contains 83 original features. We have removed some features, such as source and target IP, ID, and timestamp, because using this information may lead to overtraining. Finally, we got a dataset containing 80 features and selected 2515416 samples for experiments. Similarly in CICIDS2018, we also removed some unnecessary features and selected an unbiased subset of the original dataset. All datasets details are shown in Tables 1–5.

The features of the four datasets are composed of many numerical features and several character features that the characters cannot be directly used in the proposed intrusion detection model, and the experiment uses the one-hot encoding method to convert it from character to number. For example, the second column of the NSL-KDD dataset "protcol_type" contains three different values: "tcp," "udp," and "icmp," and encoding represent them as [0, 0, 1], [0, 1, 0], [1, 0, 0]. After encoding, the data are normalized to avoid that the size relationship between values will affect the training results, and all the feature values are mapped to the interval [0, 1].

$$y_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}, \tag{10}$$

where y_i represents the value after the feature is normalized, x_i represents the feature value, and $\min(x)$ and $\max(x)$ represent the minimum and maximum values within the range of feature values, respectively.

The experiment cluster is deployed in High-Performance Computing Center of Hebei University, which consists of a master node and 50 slave nodes. The hardware conditions of each slave node are 2 ∗ Intel Xeon E5-2680 v2 (Ivy Bridge| 10C | 2.8 GHz), 64 GB DDR3 ECC

Table 1: Information of the experimental datasets.

Datasets	Instances	Features	Classes
NSL-KDD	148517	43	5
UNSW-NB15	257673	49	10
CICIDS2017	2515416	80	13
CICIDS2018	288909	80	15

Table 2: Information of the experimental NSL-KDD.

Train/Test	Total	Normal	DOS	U2R	R2L	PROBE
Train set	125973	67343	45927	11656	995	52
Test set	22544	9711	7458	2421	2754	200

Table 3: Information of the experimental UNSW-NB15.

Class	Train set	Test set
Normal	56000	37000
Analysis	2000	677
Backdoor	1746	583
DoS	12264	4089
Exploits	33393	11132
Fuzzers	18184	6062
Generic	40000	18871
Reconnaissance	10491	3496
Shellcode	1133	378
Worms	130	44
Total counts	175341	82332

Table 4: Information of the experimental CICIDS2017.

Class	Number	Proportion
Normal	2089692	83.07
DoS Hulk	172838	6.87
PortScan	128008	5.08
DDoS	90694	3.6
Dos GoldenEye	10283	0.41
FTP-Patator	5931	0.23
SSH-Patator	5385	0.21
DoS Slowloris	5228	0.2
DoS SlowHTTPTest	3219	0.12
Web attacks	2143	0.085
Bot	1948	0.08
Infiltration	36	0.0014
Heartbleed	11	0.0004

Table 5: Information of the experimental CICIDS2018.

Class	Train set	Test set
Normal	50791	12698
SSH-BruteForce	184	46
FTP-BruteForce	489	122
BruteForce-XSS	7504	1876
BruteForce-Web	15469	3867
SQL injection	70	17
DoS attacks-Hulk	18667	4667
DoS attacks-SlowHTTPTest	55956	13989
DoS attacks-Slowloris	4396	1099
DoS attacks-GoldenEye	16603	4151
DDOS attack-HOIC	27441	6860
DDOS attack-LOIC-UDP	1384	346
DDOS attack-LOIC-HTTP	23048	5762
Bot	11448	2862
Infiltration	6478	1620
Total counts	231127	57782

1866 MHz four-channel memory. Moreover, the master node is equipped with 4 ∗ Intel Xeon E7-4850 (Ivy Bridge|10C|2.0 GHz) and 512 GB DDR3 REG 1333 memory. Internal connection bandwidth is 56 Gbps IB, and chip transmission delay 100 ns. The system setup for all nodes is CentOS-7-GenericCloud-1503.qcow2, Hadoop 2.6.3, Scala-2.10.5, and Spark 1.6.1.

6.2. Hyperparameter Setting. In this part, NSL-KDD is taken as an example dataset to illustrate the influence of hyperparameters in the proposed model and to demonstrate the process of parameter tuning.

Equation (11) interprets that about 1/3 of samples will not appear in the collection set whenever bootstrap, which is called out-of-bag (OOB) data.

$$\lim_{n \longrightarrow \infty} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e} \approx 0.368. \tag{11}$$

These data will not participate in the establishment of the decision tree and can replace the validation set to verify the model.

OOB error rate is calculated to evaluate the effect of different sliding window sizes on the model. As can be seen from Figure 6, when the window size is $d/4$, the average error rate is the lowest, and the average OOB error rate is the highest at $d/16$, where d is the raw feature length. This result interprets that a more fine-grained window size is not necessarily better when trying to enhance the generalization performance of the model. With the increase of decision trees, the error rate starts to converge at about 0.06. So, it is finally recommended that $d/4$ is the suitable size.

Then, the remaining parameters are adjusted by using 10-fold cross-validation. For instance, n_estimators is the number of decision trees. Generally speaking, more trees make the model more robust and have better performance. Considering a wider availability, we searched it on the range of (0, 500] by step-size 50 and compared the results after 10-fold cross validation to get the optimum value. Finally, Table 6 summarizes the rest of hyperparameters setting of FS-DPRF.

6.3. Model Evaluation. In this section, we compare FS-DPRF with parallel random forest (PRF), DSSVM [17], and A-DNN [19]. The classification performance is measured by the accuracy, recall precision and F1, detection rate (DR), and false alarm rate (FAR). The F1 score is an evaluation index that comprehensively considers the recall rate and precision, and its definition is shown in equation (13). The higher F1 score value means the better the classification performance of the algorithm. The evaluation metrics are defined as follows:

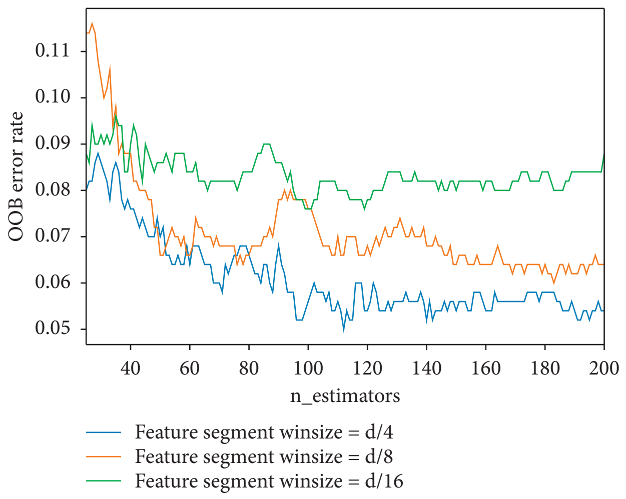

FIGURE 6: Impact of feature slice window size.

$$\begin{aligned}
\text{Accuracy} &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}},\\
\text{DR} &= \frac{\text{TN}}{\text{TN} + \text{FP}},\\
\text{FAR} &= \frac{\text{FP}}{\text{FP} + \text{TN}},\\
\text{Recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}},\\
\text{Precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}},
\end{aligned} \tag{12}$$

$$F1 - \text{measure} = 2 \cdot \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}, \tag{13}$$

where TP represents the number of true attacks predicted as attack type, FN represents the number of true attacks predicted as normal, FP represents the number of true normal predicted as attack type, and TN represents the number of normal predicted as normal. The comparison results are the average value obtained from ten experiments under different datasets. In order to verify the performance of the model under two classifications: normal and attack. In this part, all attack types are regarded as abnormal types. We mark all data as two types: normal and abnormal. Table 7 shows the comparison results of normal and abnormal on given datasets.

On the NSL-KDD dataset, the accuracy of the comparison algorithm is more than 90%, and the deep neural network has achieved a good result of 98%, but it is still 1% lower than FS-DPRF. Similarly, precision and recall reflect that our model's ability to confirm normal network behavior is the best. On the UNSW-NB15 dataset, shallow machine learning algorithms began to perform weak, only 80%–85%

on accuracy. The two methods based on deep learning carry a high accuracy rate on above 94.2% and 97.7%. FS-DPRF is better than the deep neural network. As data become more and more complex and closer to the real network environment, the average accuracy of shallow algorithms on the two datasets of CICIDS2017 and CICIDS2018 has been lower than 90%. Precision and recall have also lost their competitiveness. The accuracy rates of A-DNN and FS-DPRF on the CICIDS2017 dataset reached 96.5% and 97.4%, respectively. Although the precision of A-DNN is higher, leading by 1%, the recall rate of the FS-DPRF is higher. The higher the recall rate, the higher the probability that the attack is predicted, which means that FS-DPRF has better attack detection capabilities on CICIDS2017. The last group of data is the result of the CICDIS2018. The accuracy of FS-DPRF is 3% higher than that of A-DNN, and the precision and recall rate are also higher than A-DNN by 1.3% and 2.4%, respectively.

Figure 7 shows the comparison results of FS-DPRF, parallel random forest, DSSVM, and deep neural network on F1 index. The score of the new method on the NSL-KDD dataset is higher than that of shallow machine learning, with an advantage of 3%–5%, and it is also 1% higher than A-DNN. The results on the UNSW-NB15 dataset show that there is a gap between the performance of deep learning-based methods and PRF and DSSVM. Deep learning-based methods perform better, and the score of FS-DPRF is 3% higher than that of deep neural networks. On CICIDS2017 and CICIDS2018, the F1 scores of the four methods are slightly lower than the experimental data of the first two groups, but the deep learning-based method still maintains the lead over PRF and DSSVM. The F1 score of FS-DPRF on CICIDS2017 is 1% higher than that of A-DNN, and the F1 score of FS-DPRF on CICIDS2018 is 1.9% higher than that of A-DNN. The score of the F1 index on those four datasets shows that the forest-based deep learning network in this normal/abnormal two-type classification experiment is better than the shallow machine learning method and is competitive compared with the deep neural network.

In order to verify the detection ability of the model in multiclassification, we did another set of experiments on the NSL-KDD dataset. According to the original label, all data are divided into five classes as shown in Table 8. The data preprocessing and the parameter settings are the same as the previous part, and ten experiments are performed to take the average. It is worth noting that, considering the support vector machine's binary classification limitation, we spend a lot of labeling works to test DR and FAR on "a certain attack/other" separately. It can be seen from the experimental results in Table 8 that the method proposed in this paper has improved the detection rate on 5 class labels, and the FR is relatively low. Although the detection results of U2R attack types are not particularly ideal, this is also related to the imbalance in the distribution of categories in the dataset. In summary, the method in this paper has shown the best strength in multiclass attacks and normal classification experiments.

We then tested our model by average execution time and speedup, which are used to measure the scalability of parallel cluster. The speedup is defined as

$$S_p = \frac{T_1}{T_P}, \tag{14}$$

where p is the number of CPUs, T_1 refers to the execution time of the sequential execution algorithm, and T_p represents the execution time of p nodes parallel algorithms. As it can be seen from Figure 8, with the increase of slave nodes, the average execution time of the model on four given datasets is reduced. The decrease trend of execution time on different datasets is not exactly the same due to the data size. The average execution time and the number of cluster nodes show a strong correlation in all cases, which indicates that the proposed method has good scalability.

The speedup experiment tested the model in different numbers of slave nodes. As shown in Figure 9, the speedup of each dataset all increase routinely when the number of nodes increases from 1 to 25 and tends to slow down by the number of nodes from 25 to 50. The result shows that the model has a good speedup performance in datasets of different volumes and dimensions. However, it does not show a perfect linear growth like the definition shown above, and it can be interpreted that the communication overhead and task scheduling costs would become larger as the cluster scale increased.

Finally, we setup 20 slave nodes for experiment and compared the cache performance between Spark's default LRU algorithm and our optimized method in FS-DPRF. It can be seen from Figure 10 that the FS-DPRF has less execution time, where the time reduces by 8.8% in NSL-KDD and 8.9% in UNSW-NB15 compared to LRU. The execution time of FS-DPRF on the CICIDS2017 and the CICIDS2018 also decreases by 7.2% and 13.3%, respectively, compared with LRU. The column chart indicates the cache replacement strategy proposed in this paper can cut down the execution time of anomaly detection task successfully. Even if the efficiency sort and partition integrity check in the replacement sacrifice a part of memory, as shown in Figure 11, that FS-DPRF is a little bit higher than the LRU algorithm in memory occupancy rate, and acceptable real-time performance is more important to intrusion detection. Therefore, the optimization for parallelization successfully improves the task execution efficiency of the proposed intrusion detection model.

7. Limitations of the Research

The limitation of the research is that the model will consume a lot of memory, so to get a well-trained intrusion detection model requires powerful computing equipment. Although

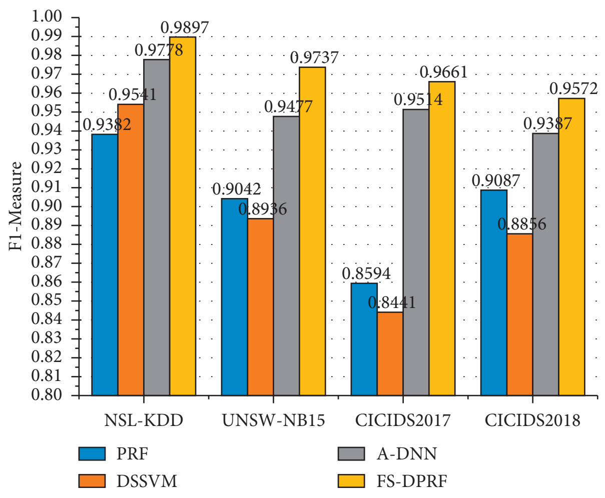

Figure 7: Comparisons of F1 score between the proposed model and previous classic algorithms.

Table 6: The hyper-parameter setting of FS-DPRF.

Parameters	Feature-grained random forest	Cascade layers random forest
n_estimators	150	350
max_depth	10	15
max_features	$\sqrt{n}$	$\log_2 (n+1)$
min_samples_leaf	4	2
min_samples_split	2	2
Criterion	Gini	Gini

Table 7: The experiment results on the four given datasets.

Dataset	Algorithm	Accuracy	Precision	Recall
NSL-KDD	PRF	0.93224	0.94648	0.93013
	DSSVM	0.94137	0.95763	0.95065
	A-DNN	0.98952	0.98101	0.97474
	FS-DPRF	0.99132	0.99213	0.98733
UNSW-NB15	PRF	0.85321	0.93752	0.87308
	DSSVM	0.81927	0.92931	0.86054
	A-DNN	0.94219	0.95740	0.93829
	FS-DPRF	0.97763	0.98363	0.96401
CICIDS2017	PRF	0.86532	0.85655	0.86227
	DSSVM	0.85290	0.84739	0.84075
	A-DNN	0.96527	0.96028	0.94272
	FS-DPRF	0.97430	0.95330	0.97931
CICIDS2018	PRF	0.89876	0.91010	0.90733
	DSSVM	0.87852	0.89110	0.88031
	A-DNN	0.94480	0.94754	0.93015
	FS-DPRF	0.96750	0.96031	0.95424

Table 8: The multiclassification results on NSL-KDD.

Type	PRF		DSSVM		A-DNN		FS-DPRF	
	DR	FAR	DR	FAR	DR	FAR	DR	FAR
DOS	0.942	0.057	0.960	0.040	0.983	0.017	0.990	0.010
R2L	0.947	0.053	0.892	0.109	0.924	0.076	0.961	0.039
U2R	0.930	0.073	0.921	0.078	0.925	0.075	0.976	0.024
Probe	0.863	0.136	0.911	0.088	0.887	0.113	0.946	0.053
Normal	0.955	0.048	0.960	0.040	0.948	0.052	0.983	0.016

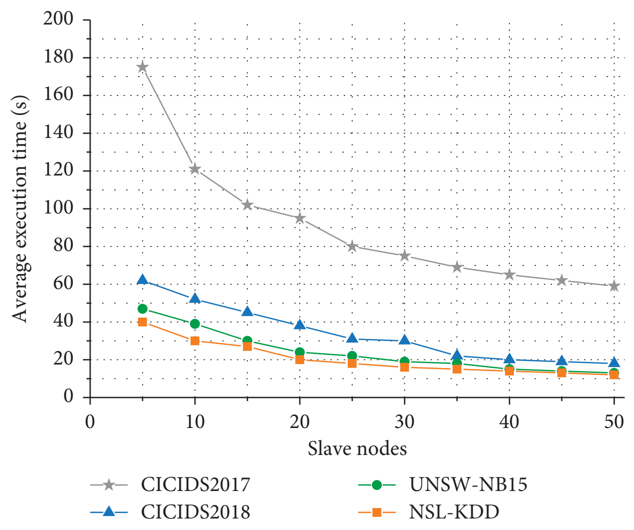

Figure 8: Average execution time of FS-DPRF in different node scales.

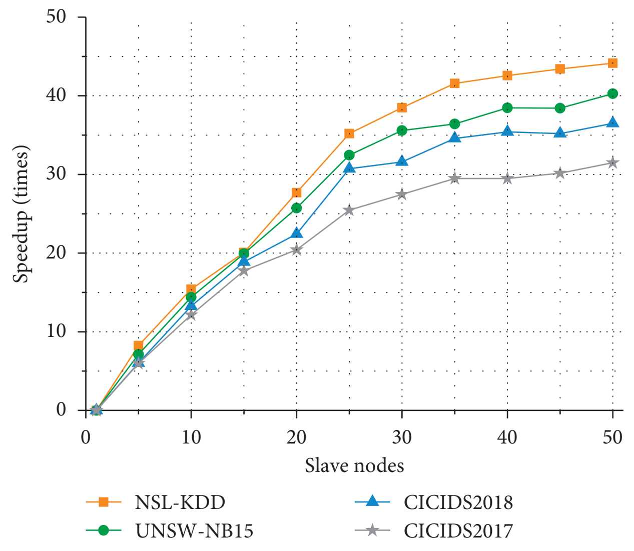

Figure 9: Speedup curve of FS-DPRF in different node scales.

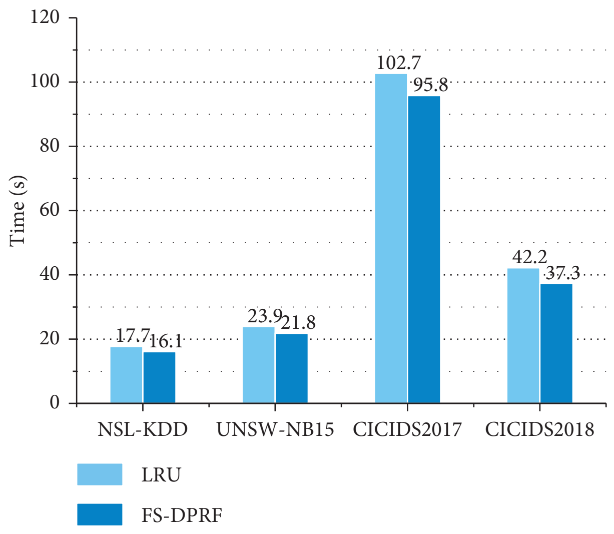

Figure 10: Execution time comparison between FS-DPRF and LRU.

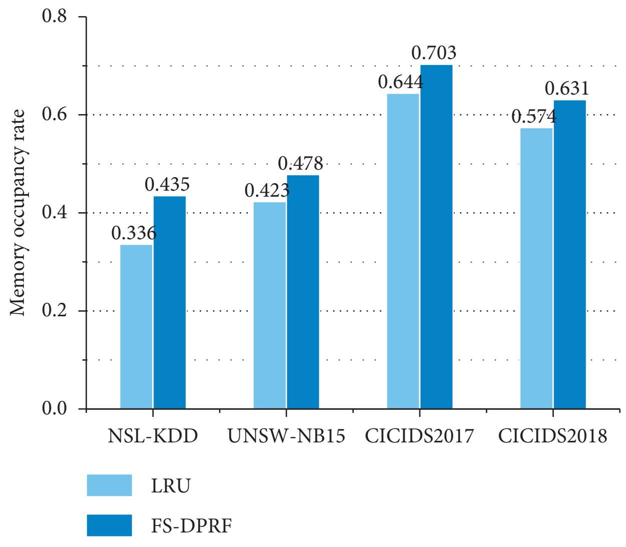

FIGURE 11: Memory usage comparison between FS-DPRF and LRU.

the model proposed in this paper has achieved good results trained by CPUs in the Spark distributed environment, unfortunately, the current structure is naturally not suitable for GPUs. This makes the model temporarily unable to be better accelerated on the GPUs like a deep neural network.

8. Conclusion and Future Work

From the work of predecessors, the ensemble learning-based method has shown convincing performances in challenging missions that can be abstractly understood as classification problem. The main scientific contribution of this paper is to propose a deep learning model based on ensemble decision trees. Inspired by deep neural networks, we used layers composed of random forests to imitate the hidden layers and fully connected layers in the neural network to build a cascading model of random forests. The proposed model utilizes a slide window to segment sample features into many small pieces, which can reduce calculation volume of high dimension data for every compute and keep the integrity of raw features. The cascade structure improves the generalization ability and has a higher accuracy rate. The model has only a few hyper-parameters while achieving good generalization ability. Another part of the contribution is to propose a cache replacement strategy for RDDs in the Spark environment and determine the priority order of RDD loading by calculating weights and completeness. It effectively reduces the average execution time of intrusion detection tasks on distributed clusters. The experimental results on four datasets have demonstrated that the model proposed in this paper performs better than the parallel random forest and support vector machine in F1-measure and accuracy and achieves competitive performance compared to the state-of-the-arts approach of deep neural networks. Although the model reduces the average execution time, it increases the memory consumption and does not support GPU acceleration temporarily. Therefore, the model is more suitable for deployment on a distributed cluster with sufficient memory, which also reflects the limitations of our model. In the future, the work will focus more on the optimization processes of the features of training data to improve the prediction accuracy and will further research on the issue of unbalanced data distribution in the intrusion detection task.

Data Availability

The datasets are available at https://www.unb.ca/cic/datasets/index.html, Cyber Range Lab of the Australian Center for Cyber Security (ACCS) (https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/), and Canadian Institute for Cybersecurity (https://www.unb.ca/cic/datasets/ids-2017.html https://www.unb.ca/cic/datasets/ids-2018.html).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Natural Science Foundation of Hebei Province, China (F2019201427) and Ministry of Education Fund Project of China (2017A20004).

References

[1] M. Ahmed, A. N. Mahmood, and M. R. Islam, “A survey of anomaly detection techniques in financial domain,” *Future Generation Computer Systems*, vol. 55, pp. 278–288, 2016.
[2] R. Yu, X. He, and Y. Liu, “GLAD: Group Anomaly detection in social media analysis,” *ACM Transactions on Knowledge Discovery from Data*, vol. 10, no. 18, pp. 1–22, 2015.
[3] Y. Djenouri and A. Zimek, “Outlier detection in urban traffic data,” in *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, no. 3, pp. 1–12, Novi Sad, Serbia, June 2018.
[4] K. Sequeira and M. Zaki, “ADMIT: anomaly-based data mining for intrusions,” in *Proceedings of the Eighth ACM*

SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, Canada, July 2002.

[5] S. Roberto, S. Carta, and D. R. Recupero, "A probabilistic-driven ensemble approach to perform event classification in intrusion detection system," in *Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, pp. 139–146, Seville, Spain, June 2018.

[6] M. S. Pervez and D. M. Farid, "Feature selection and intrusion classification in NSL-KDD CUP 99 dataset employing SVMs," in *Proceedings of the 8th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*, pp. 1–6, Kuala Lumpur, Malaysia, December 2014.

[7] S. Vishwakarma, V. Sharma, and A. Tiwari, "An intrusion detection system using KNN-ACO algorithm," *International Journal of Computer Applications*, vol. 171, no. 10, pp. 18–23, 2017.

[8] N. Farnaaz and M. A. Jabbar, "Random forest modeling for network intrusion detection system," *Procedia Computer Science*, vol. 89, pp. 213–217, 2016.

[9] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[10] R. K. Malaiya, D. Kwon, S. C. Suh, H. Kim, I. Kim, and J. Kim, "An empirical evaluation of deep learning for network anomaly detection," *IEEE Access*, vol. 7, pp. 140806–140817, 2019.

[11] D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim, "A survey of deep learning-based network anomaly detection," *Cluster Computing*, vol. 22, no. 3, pp. 1–13, 2019.

[12] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, pp. 1153–1176, 2016.

[13] B. Kolosnjaji, G. Eraisha, G. Webster, A. Zarras, and C. Eckert, "Empowering convolutional networks for malware classification and analysis," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pp. 3838–3845, Anchorage, AK, USA, May 2017.

[14] D. Li, D. Chen, J. Goh, and S. Ng, "Anomaly detection with generative adversarial networks for multivariate time series," 2019, http://arxiv.org/abs/1809.04758.

[15] J. Chen, S. Sathe, C. Aggarwal, and D. Turaga, "Outlier detection with autoencoder ensembles," in *Proceedings of the 2017 SIAM International Conference on Data Mining*, pp. 90–98, Houston, TX, USA, April 2017.

[16] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning," *Pattern Recognition*, vol. 58, pp. 121–134, 2016.

[17] L. Khalvati, M. Keshtgary, and N. Rikhtegar, "Intrusion detection based on a novel hybrid learning approach," *Journal of AI and Data Mining*, vol. 6, no. 1, pp. 157–162, 2018.

[18] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Applying convolutional neural network for network intrusion detection," in *Proceedings of the International Conference on Advances in Computing Communications and Informatics (ICACCI)*, pp. 1222–1228, Udupi, India, September 2017.

[19] S. Potluri and C. Diedrich, "Accelerated deep neural networks for enhanced Intrusion Detection System," in *Proceedings of the IEEE 21st International Conference on Emerging Technology and Factory Automation (ETFA)*, pp. 1–8, Berlin, Germany, September 2016.

[20] N. Gao, L. Gao, Q. Gao, and H. Wang, "An intrusion detection model based on deep belief networks," in *Proceedings of the 2014 Second International Conference on Advanced Cloud and Big Data*, pp. 247–252, Huangshan, China, November 2014.

[21] R. Domingues, M. Filippone, P. Michiardi, and J. Zouaoui, "A comparative evaluation of outlier detection algorithms: experiments and analyses," *Pattern Recognition*, vol. 74, pp. 406–421, Feb, 2018.

[22] K. Hundman, V. Constantinou, C. Laporte, L. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 387–395, London, UK, August 2018.

[23] E. Manzoor, H. Lamba, and L. Akoglu, "xStream: outlier detection in feature-evolving data streams," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1963–1972, London, UK, August 2018.

[24] H. Zheng, Y. Zhang, L. Yang et al., "A new ensemble learning framework for 3D biomedical image segmentation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 5909–5916, Honolulu, HI, USA, February 2019.

[25] F. T. Liu, K. M. Ting, and Z. H. Zhou, "Isolation-based anomaly detection," *ACM Transactions on Knowledge Discovery from Data*, vol. 6, no. 3, pp. 1–39, Mar, 2012.

[26] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[27] T. Chen and C. Guestrin, "XGBoost: a scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, San Francisco, CA, USA, August 2016.

[28] P. Su, Y. Liu, and X. Song, "Research on intrusion detection method based on improved smote and XGBOOST," in *Proceedings of the 8th International Conference on Communication and Network Security*, pp. 37–41, Qingdao, China, November 2018.

[29] Y. Zhou, T. A. Mazzuchi, and S. Sarkani, "M-AdaBoost-a based ensemble system for network intrusion detection," *Expert Systems with Applications*, vol. 162, p. 113864, 2020.

[30] F. A. Khan, A. Gumaei, A. Derhab, and A. Hussain, "TSDL: a two-stage deep learning model for efficient network intrusion detection," *IEEE Access*, vol. 7, pp. 30373–30385, 2019.

[31] J. Gao and P. Tan, "Converting output scores from outlier detection algorithms into probability estimates," in *Proceedings of the 6th International Conference on Data Mining (ICDM)*, pp. 212–221, Hong Kong, China, December 2006.

[32] M. Zaharia, "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the Conference on Networked Systems Design and Implementation (NSDI)*, pp. 15–28, San Jose, CA, USA, April 2012.

[33] J. Chen, "A parallel random forest algorithm for big data in a spark cloud computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 919–933, 2017.

[34] M. M. Khan, M. A. U. Alam, A. K. Nath, and W. Yu, "Exploration of memory hybridization for RDD caching in Spark," in *Proceedings of the ACM SIGPLAN International Symposium on Memory Management (ISMM)*, pp. 41–52, Phoenix, AZ, USA, June 2019.

[35] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Security and Defense Applications*, pp. 1–6, Ottawa, ON, USA, July 2009.

[36] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proceedings of the IEEE Military Communications and Information Systems Conference, MilCIS 2015*, pp. 1–6, Canberra, Australia, November 2015.

[37] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, pp. 108–116, Funchal, Portugal, June 2018.