Network Intrusion Detection Based on Convolutional Recurrent Neural Network, Random Forest, and Federated Learning

Qianying Zou[1], Yushi Li[2], Xinyue Jiang[2], Yuepeng Zan[2] and Fengyu Liu[1]

[1]Geely University of China, Chengdu, China
[2]Chengdu College of University of Electronic Science and Technology of China, Chengdu, China

This paper presents a novel network intrusion detection framework that combines convolutional recurrent neural networks (CRNN) and random forest (RF) models within a federated learning setting. The proposed approach aims to address the challenges of data privacy, computational efficiency, and model generalization in traditional network intrusion detection methods. By leveraging the spatial feature extraction capabilities of CRNN and the feature selection and noise reduction properties of RF, the framework enhances the accuracy and robustness of attack detection. The integration of federated learning enables collaborative model training without compromising data privacy. Extensive experiments on benchmark datasets demonstrate the superiority of the proposed method compared to state-of-the-art techniques, achieving high performance metrics such as accuracy, precision, recall, F1 score, and AUC. The proposed framework offers a promising solution for secure and efficient network intrusion detection in real-world scenarios, contributing to the advancement of cybersecurity practices.

ACM CCS (2012) Classification: Security and privacy → Intrusion/anomaly detection and malware mitigation → Intrusion detection systems

Keywords: federated learning framework, convolutional recurrent neural networks, network security enhancement, temporal data processing, random forest integration, feature selection optimization

1. Introduction

Cybersecurity is the core of national strategy in the People's Republic of China. In the face of the complex challenges brought about by the development of Internet technology, China strengthens the protection of network resources, data security and user privacy through the formulation of laws and regulations on cybersecurity, ensures the lawful compliance of network activities and services, and strengthens the supervision of cybersecurity management. The Cybersecurity Law of the People's Republic of China [1], as the basic law of cybersecurity, covers a wide range of aspects such as the basic system of cybersecurity, key technologies, data protection, personal information protection, cybersecurity review, cybersecurity level protection, and cybersecurity incident disposal. A network intrusion detection system (NIDS) is an important security tool [2] that improves network security by monitoring network traffic and devices, analyzing communication characteristics, status, and anomalies to identify malicious activities, and issuing alerts or taking measures when a network attack is detected. NIDS is widely used in the government, enterprises, education and scientific research fields.

In the research of network intrusion detection, some of them adopt the idea of federated learning, which allows multiple nodes to collaboratively train network intrusion detection models without sharing data [3-10], *e.g.*, FL-SEResNet [11] and DFC-NID [12]. Federated learning protects data privacy, improves the performance of the model, and adapts to distributed and heterogeneous data environments. However, during the training process of FL-SEResNet, the lack of a global data perspective may limit the model's generalization ability. Additionally, this method consumes significant network resources, which affects efficiency in practical applications.

The DFC-NID approach, by introducing an attention mechanism, enhances the feature extraction and classification capabilities of deep residual networks, thus improving the efficiency and accuracy of intrusion detection. However, DFC-NID has high model complexity and computational overhead when handling high-dimensional, dynamically changing data, making it difficult to deploy in resource-constrained environments.

On the other hand, deep neural networks are used to extract effective features from network traffic to identify normal and abnormal behaviors [13-14]. For example, a network traffic anomaly detection model based on multiscale memory residual networks (MMRNs) [15] utilizes the multiscale feature fusion and residual connectivity capabilities of MMRNs to improve the complexity and reliability of detection. However, the MMRN model has high computational and storage demands when processing large-scale data, which can easily increase the system's burden.

In addition, there are studies that utilize traditional machine learning methods such as KNN and genetic algorithms to achieve network intrusion detection [16-20]. For example, the network anomaly detection technique combining TCM-KNN and genetic algorithm [21], improves the accuracy, robustness and flexibility of network anomaly detection by optimizing the K-value and the feature subset. However, traditional machine learning methods typically rely on feature engineering, with the model's performance heavily dependent on manually designed features. This makes it difficult to automatically adapt to different data distributions and attack patterns.

Deep learning approaches have also been proposed to address the core challenges in network intrusion detection such as DBN-ELM model [22], BiLSTM model [23], S-NDAE-RF model [24] and TL-NID model [25]. These methods effectively improve the detection performance by combining different models, showing the potential of deep learning in network intrusion detection. However, these methods generally face issues of high computational complexity and resource consumption during training. Additionally, the process of tuning model parameters is quite complex, which affects the convenience of practical applications.

The three main core challenges faced in this paper are cyber security, data privacy and data utilization. To address these challenges, this paper proposes a scheme that combines convolutional recurrent neural networks and random forest classification models with federated learning to address the limitations of traditional models for network intrusion detection, such as data dependency, computational overhead, and model generalization. The main contributions of this paper include:

- A federated learning scheme combining convolutional recurrent neural networks (CRNN) and random forest (RF) classification models: This approach fully leverages the strengths of both models. CRNNs can effectively extract and process high-dimensional, nonlinear, and dynamically changing data features, while RF models reduce the risk of overfitting through the ensemble of multiple decision trees, thereby enhancing the model's generalization ability. Moreover, this scheme can dynamically adjust model parameters to accommodate the data characteristics and network environments of different nodes.
- Training models through federated learning without sharing raw data: only encrypted gradient-related data is transmitted, effectively protecting data privacy. This method not only ensures the privacy and security of data owners but also meets compliance requirements, which is especially valuable in the current context where data privacy protection is increasingly emphasized.
- Utilizing federated learning for collaborative training of multi-source data: this approach addresses the issue of data silos and improves data utilization efficiency. By enabling collaborative model training across multiple nodes, it fully leverages dispersed data resources, enhancing the model's training effectiveness and detection performance. This method is particularly significant in distributed network environments, as it can effectively enhance overall network security protection capabilities.

- Through comprehensive performance evaluation, including accuracy, recall, F1 score, and ROC curve, the effectiveness and superiority of the model presented in this paper are demonstrated. Specifically, the model's generalization ability and adaptability are improved by collaborative training with data from multiple nodes. The experimental results show that the model performs excellently in various network intrusion detection tasks, indicating strong practical applicability.

Potential impacts of this paper include:

- Enhancing network security: The proposed method significantly improves the accuracy and efficiency of network intrusion detection, providing a more robust defense mechanism for network security. By combining CRNNs, RFs, and federated learning techniques, it can accurately identify and prevent network attacks in complex network environments, thereby reducing network security risks.
- Protecting data privacy: Against the backdrop of increasing importance of data privacy protection, the method proposed in this paper offers an effective solution for model training while safeguarding data privacy. This is particularly valuable for fields that handle sensitive data, such as finance, healthcare, and government sectors.
- Promoting the application of distributed computing: By implementing federated learning for collaborative training of distributed data, the proposed method facilitates the application of distributed computing in the field of network security. By fully utilizing dispersed data resources, it enhances the model's training effectiveness and detection performance, thereby improving network security protection on a larger scale.
- Cross-domain application potential: The method proposed in this paper is not only applicable to the field of network security but can also be extended to other areas, requiring distributed data processing and privacy protection, such as IoT security, smart manufacturing, and intelligent transportation. Applying this method in these fields can effectively improve data utilization efficiency and system security.

2. Related Work

2.1. Convolutional Neural Network

A convolutional neural network (CNN) is specialized in processing multidimensional data and shows high effectiveness in network intrusion detection, where the convolutional layer captures the local features of the network traffic data by means of a convolutional kernel [26]. This step is performed automatically to enable efficient extraction of key features that may indicate malicious attacks. With this hierarchical structure and operation, CNN is able to provide high accuracy and good reliability in network intrusion detection analysis [27], as shown in equation (1):

$$o_{\{i,j\}} = \sum_m \sum_n \left(I_{i+m,j+n} * k_{m,n} \right) \tag{1}$$

where o is the output feature map, I is the input feature map, k is the convolution kernel, and m and n are the width and height of the convolution kernel.

In this paper, we use mean pooling, as shown in equation (2):

$$o_{\{i,j\}} = mean\left(I_{i:i+k',j:j+k'} \right) \tag{2}$$

Here, k' is the size of the pooling window. The main working mechanism of mean pooling is to generate the output by calculating the mean value of all elements within a given window.

The fully connected layer is located at the end of the neural network, and its main task is to integrate the local features that have been extracted and optimized by the previous layers into a one-dimensional vector. In this way, the fully connected layer ensures that the network can effectively classify and regress based on the important features that have been learned, as shown in equation (3):

$$o' = I' * w + b \tag{3}$$

where o' is the output data, I' is the input data, w is the weight matrix, and b is the bias vector.

2.2. Recurrent Neural Network

Structurally, a recurrent neural network (RNN) is mainly composed of an input layer, a hidden layer and an output layer [28]. First of all, the input layer is tasked with transforming the original input data into a feature vector format that is more suitable for network processing, providing a basis for subsequent information processing and decision making, as shown in equation (4).

$$x_t = E\left(w_t\right) \tag{4}$$

Here, x_t is the input vector at the t-th time step, E is the word embedding matrix, and w_t is the input vector at time step t.

RNN consists of multiple loop units, each unit processes time series data. The input includes the data of the current time step and the hidden state of the previous time step, which together determine the output and the new hidden state. The hidden layer synthesizes the current inputs and the historical information to effectively extract the sequence features [29], as shown in equation (5):

$$s_t = f(U * x_t + W * s_{t-1} + b_1) \tag{5}$$

where s_t is the hidden state vector of the t-th time step of the hidden state vector, f is the activation function, U is the input-to-hidden weight matrix, W is the hidden-to-hidden weight matrix, and b_1 is the bias vector of the hidden layer.

In this paper, the output layer uses a softmax activation function to transform this state information into the format required for a particular task. In classification problems, the softmax function is used to convert the output into probability distributions for each category as shown in equation (6):

$$o_t = g(V * s_t + b_2) \tag{6}$$

where o_t is the output vector of the t-th output vector at the t-th time step, g is the softmax function, V is the hidden-to-output weight matrix, and b_2 is the bias vector of the output layer.

In the field of network intrusion detection, network traffic data is usually presented in the form of a time series. This format is ideally suited for detecting temporal patterns and anomalous behaviors of network intrusions. By analyzing these time series data, potential security threats can be identified more accurately, and preventive measures can be taken accordingly.

2.3. Random Forest

Random forest is an ensemble learning model with multiple advantages. First, the model can effectively handle high-dimensional data and reduce the risk of overfitting by integrating multiple decision trees, thus enhancing the generalization ability of the model. Second, random forest allows parallelization, which is advantageous when dealing with large-scale high-dimensional datasets, such as NSL-KDD [30]. By utilizing parallel computing and the random generation property of the tree, random forest not only accelerates the training and prediction process of the model, but also enhances the robustness and interpretability of the model. The output of random forest is composed of the outputs of multiple decision trees [31], and the output formula of random forest, as shown in equation (7).

$$\begin{aligned} y &= f(x_1, x_2, \ldots, x_n) \\ &= \begin{cases} \mathit{majority\ vote}\left(\left\{f_1\left(x_1, x_2, \ldots, x_n\right),\right.\right. \\ \qquad f_2\left(x_1, x_2, \ldots, x_n\right), \\ \qquad \left.\left. f_n\left(x_1, x_2, \ldots, x_n\right), \right\}\right), \\ \qquad \mathit{forclassification} \\ \mathit{mean}\left(\left\{f_1\left(x_1, x_2, \ldots, x_n\right),\right.\right. \\ \qquad f_2\left(x_1, x_2, \ldots, x_n\right), \\ \qquad \left.\left. f_n\left(x_1, x_2, \ldots, x_n\right), \right\}\right), \\ \qquad \mathit{forregression} \end{cases} \end{aligned} \tag{7}$$

Here, y denotes the target variable, x_1, x_2, ..., x_n denote the dependent variables, f denotes the output of the random forest, M denotes the total number of trees in the forest, f_i denotes the output of the i-th tree, majority vote denotes the majority voting function, and mean denotes the averaging function.

2.4. Federal Learning

Federated learning (FL) aims to train a unified model together from multiple decentralized data sources [32]. In this architecture, individual devices (or nodes) first train the model locally using their own data. Instead of sharing the raw data, these devices only send model updates to a centralized server, which is responsible for collecting and integrating model updates from all devices to further optimize the model [33]. Upon completion of this step, the updated model is again distributed back to the individual devices for further local training. This approach allows multiple data owners to jointly participate in the training and optimization of the model while ensuring data privacy. Especially in virtual network environments, federated learning effectively addresses the issues of data privacy and model training through this distributed and collaborative approach.

Vertical federation learning is a form of federation learning for datasets with the same sample space but different feature spaces. It centers on federating features and is particularly suitable for scenarios where there is a lot of user overlap but little feature overlap [34].

Suppose there are K data owners, each owner k has n_k samples $\{x_{ik}, y_{ik}\}$, where x_{ik} is the feature vector and y_{ik} is the label. Suppose the model is a linear regression model with parameters w. Then, the equations of federated learning are shown in equations (8), (9) and (10).

Objective function:

$$\min_{w} \sum_{k=1}^{K} \frac{n_k}{n} \sum_{i=1}^{n_k} \left(w^T x_{ik} - y_{ik} \right)^2 + \frac{\lambda}{2} \|w\|^2 \tag{8}$$

Optimization algorithm:

$$\Delta w_k = -\eta \frac{\partial L_k(w)}{\partial w} = -\eta \left(\frac{2}{n_k} \sum_{i=1}^{n_k} \left(w^T x_{ik} - y_{ik} \right) x_{ik} + \lambda w \right) \tag{9}$$

Communication protocols:

$$w(t+1) = w(t) + \frac{1}{K} \sum_{k=1}^{K} \Delta w_k \tag{10}$$

Here, η is the learning rate, λ is the regularization factor, and t is the number of iterations. This equation indicates that each data owner calculates the gradient based on his/her data Δw_k and then sends it to the central server, which averages all the gradients, updates the model parameters w and then distributes the updated ones to all data owners. This process is repeated until convergence.

3. Method

3.1. Modeling of Convolutional Recurrent Neural Networks

Convolutional recurrent neural network (CRNN) is a state-of-the-art deep learning architecture that integrates the features of CNN and RNN. The model is designed to utilize both spatial and temporal information to improve the processing capability of high-dimensional, nonlinear and dynamically changing data. In terms of model structure, the CRNN first efficiently extracts features from the input data through its convolutional layer. Next, these features are fed into the recurrent layer for serialization, which utilizes the memory capability of the recurrent layer, enabling the model to perform more accurate time-series data classification and regression.

For network intrusion detection systems, the CRNN model has good application value. The model has the ability to analyze network traffic data in depth and can accurately distinguish between normal and abnormal traffic, thus showing high effectiveness in network attack detection and prevention. Since the CRNN model is capable of handling high-dimensional, nonlinear and dynamically changing network traffic data, it contributes to improving the accuracy and efficiency of network intrusion detection.

In order to fully utilize the respective advantages of CNN and RNN, this paper designs a network intrusion detection model that integrates the two, which is referred to as the CNN-RNN model, as shown in Figure 1. The construction and application of the model can be divided into three main steps: data preprocessing, feature extraction and time series analysis, and final classification and regression.

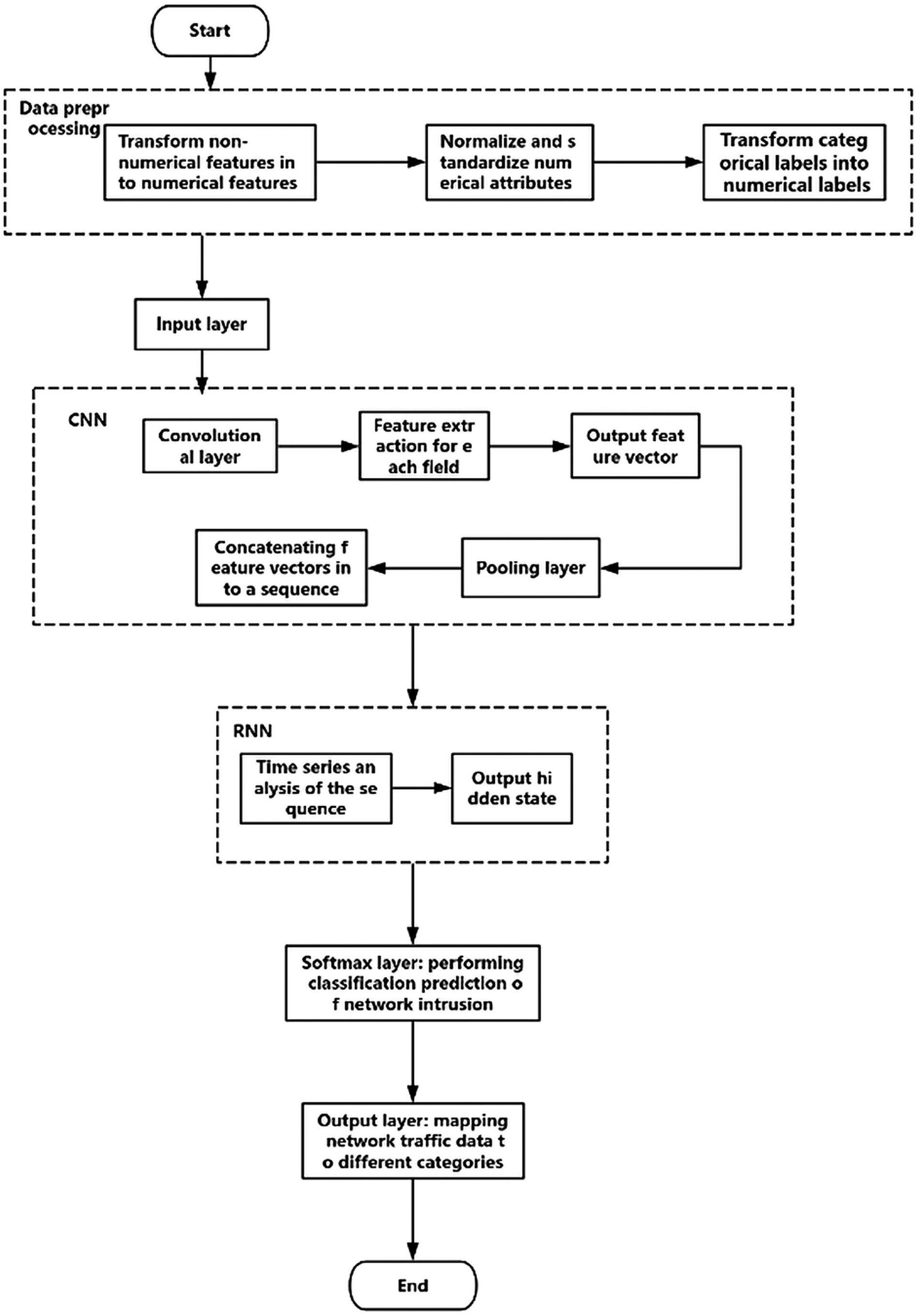

Start
Data prepr ocessing
Transform non-numerical features in to numerical features
Normalize and s tandardize num erical attributes
Transform categ orical labels into numerical labels
Input layer
CNN
Convolution al layer
Feature extr action for e ach field
Output feat ure vector
Pooling layer
Concatenating f eature vectors in to a sequence
RNN
Time series an alysis of the se quence
Output hi dden state
Softmax layer: performing classification prediction o f network intrusion
Output layer: mapping network traffic data t o different categories
End

Figure 1. Framework diagram of convolutional recurrent neural network modeling.

First, a series of preprocessing operations are performed on the dataset to be processed. These include:

- non-numerical attributes, such as protocol type, service type, and connection status, *etc.*, are converted into numerical features by the label encoding method;
- numerical attributes, such as duration, source byte, and target byte, *etc.*, are normalized or standardized to eliminate differences in magnitude;
- Categorical labels, such as "Normal", "DoS" and "U2R", *etc.*, are converted into numerical labels (*e.g.*, 0, 1, 2, *etc.*) for subsequent classification and regression analysis.

Then, we enter the feature extraction and time series analysis phase of the model. In this step, each network traffic record is considered as a timing signal and is divided into multiple equal-length sub-segments. These sub-segments are used as inputs to the CNN model for feature extraction. After CNN processing, each sub-segment generates a feature vector. All these feature vectors are stitched together into a sequence and used as inputs to the RNN model. The RNN model is responsible for the temporal analysis of this feature sequence and generates a hidden state.

Feature selection and preprocessing are critical for model performance. In this study, the feature selection process includes the following steps:

1. Feature screening: By conducting an initial analysis of network traffic data, features strongly correlated with intrusion detection are selected. Examples include protocol type, service type, source IP and destination IP, and packet length.
2. Feature encoding: For non-numerical features (such as protocol type, service type), label encoding is used to convert them into numerical features. This step enhances the model's ability to handle non-numerical data.
3. Feature normalization: Numerical features (such as packet length, duration) are normalized to eliminate differences in dimensions among features, thereby improving the model's training stability and convergence speed.

Finally, this hidden state is sent to the output layer for final classification and regression. In this way, the whole CNN-RNN model can effectively handle high-dimensional, nonlinear and dynamically changing network traffic data with high detection accuracy and efficiency.

There are two key components involved in the working process of the CNN model: the convolutional layer and the pooling layer. First, the convolutional layer is responsible for extracting local features from network traffic data, such as protocol types, source ports, and destination ports. By using multiple convolution kernels to perform convolution operations on the input vectors, the resulting multiple feature maps not only enhance the model's expressive capability, but also improve its generalization performance.

In a convolutional neural network, the output of the convolutional layer is obtained by matrix multiplying the input feature map with the convolutional kernel, summing and adding a bias term, this process is performed on each output feature map element.

Let the convolution kernel be h, with its length k, and the convolution step s, as shown in equation (11):

$$y_t = \sum_{j=0}^{k-1} h_j x_{j \times s + j},\ i = 0, 1, \ldots, \left[\frac{n-k}{s}\right] \tag{11}$$

where y_t is the i-th convolutional output, and [ ] denotes the under-image rounding.

The pooling layer has two main purposes in CNN modeling. First, it preserves the core features in the network traffic data through dimensionality reduction process, which is achieved in this paper through the mean pooling algorithm. Second, during the dimensionality reduction process, the method helps to remove noise and redundant information from the data, which in turn reduces the computational complexity of the model and mitigates the risk of overfitting.

To compute the output of the average pooling layer, the output value of each pooled region is obtained by computing the arithmetic average of the values of all the elements in the region, plus a bias term. Let the pooling layer window size be p, as shown in equation (12):

$$z_j = \frac{1}{p}\sum_{j=0}^{p-1} y_{i\times p+j'},\ i = 0, 1, ..., \left[\frac{n-k}{sp}\right] \tag{12}$$

where z_i is the i-th pooled output.

In the model flow, the CNN is responsible for extracting local features from the packets to generate feature vectors, which are subsequently used as input sequences to the recurrent layer to enable the RNN to learn and capture the temporal relationships between packets. The recurrent layer mainly extracts global features such as connection duration, connection frequency and connection direction from the entire packet sequence. This not only enhances the model's ability to memorize the historical information of network traffic data, but also improves its prediction accuracy.

In recurrent neural networks, the output layer output for the current time step is obtained by processing the hidden state at that time step with a linear transformation and a nonlinear activation function. This process enables the output layer to predict the target value of the sequence data based on the hidden layer state, as shown in equations (13) and (14):

$$h_t = f(w_h\, h_{t-1} + w_Z Z_t + b_h) \tag{13}$$

$$o_t = g(w_0 h_t + b_0) \tag{14}$$

where the hidden state of the loop cell is h_t, the input is z_t, the output is o_t, the activation functions are f and g, the weight matrices are wh, w_z, w_0, the bias vectors are b_h, b_z, b_0, and $t = 1, 2, ..., T$ denotes the time step.

In the final stage of the model, the fixed-length vectors output from the loop layer are fed into a softmax output layer for final network intrusion classification prediction. The function of the softmax layer is to transform the output vectors into probability distributions, where each element represents the predicted probability of a specific class (*e.g.*, normal class or four different attack classes: DoS, Probe, R2L, U2R). Therefore, this output layer essentially accomplishes the mapping from network traffic data to different intrusion classes, reaching the overall goal of network intrusion detection.

Let the weight matrix of the output layer be V, the bias vector c and the number of categories m, as shown in equations (15) and (16):

$$y_t = V\, o_t + c \tag{15}$$

$$p(y_t = i) = \frac{\exp(y_i, i)}{\sum_{j=1}^{m}\exp(y_i, t)},\ i = 1, 2, ..., m \tag{16}$$

where y_t is the prediction vector of the output layer, and $p(y_t = i)$ is the probability that the t-th sample belongs to the i-th category of the output layer.

For the CRNN, this study selected the following key hyperparameters to optimize model performance:

1. Learning rate: 0.001. It controls the speed of model parameter updates, ensuring stable convergence during training.
2. Number of epochs: 70. This is the number of times the neural network works through the entire training dataset, balancing training time and model performance.
3. Batch size: 32. This is the number of samples used to update model parameters at each step.
4. Kernel size: 3×3. This is the size of the convolutional kernels in the convolutional layers, ensuring effective feature extraction.
5. Number of filters: 64. This is the number of convolutional kernels in the convolutional layers, enhancing the representational capacity of feature maps.
6. Pooling size: 2×2. This is the window size of the pooling layers, used for dimensionality reduction and feature extraction.
7. Number of units in hidden layers: 128. This is the number of hidden units in the recurrent layers, enhancing the model's memory capability.
8. Activation function: ReLU for convolutional layers and tanh for recurrent layers.
9. Loss function: Binary cross-entropy is used to measure the difference between predicted values and actual values.

10. Regularization: L2 regularization is used with a parameter value of 0.001, used to prevent overfitting.
11. Optimizer: FedProx is suitable for model parameter optimization in federated learning scenarios.

3.2. Random Forest Modeling

In order to improve the accuracy, generalization ability, and interpretability of the network intrusion detection system, and at the same time to reduce the risk of information leakage, this paper proposes a strategy that combines the random forest model with vertical federated learning. First, the random forest model has advantages in feature selection, which can automatically identify and select the most relevant features, reduce the number of features and improve the training efficiency. Second, a double randomness strategy is introduced, *i.e.*, randomly selecting samples and features when constructing each decision tree, which helps to reduce data noise and eliminate correlation between features and improves the accuracy and efficiency of attack detection. Again, the multi-classification strategy is used to achieve fine-grained identification and classification of different types of network attacks. Finally, the error rate and overfitting risk of a single model are reduced by integrating the prediction results of multiple decision trees.

The aim of this paper is to categorize the network traffic through the random forest model, for which the specific steps and methods are shown in Figure 2:

1. Data preparation. In this paper, we use a five-category dataset (normal, denial-of-service attack, probing attack, remote access attack, and user privilege elevation attack) that includes both normal and abnormal network traffic, where these five network traffic categories are used as target variables, and service types, packet lengths, error fragments, and emergency packets are used as feature variables and numerically converted to the category-type features by LabelEncoder.

   In the feature selection process, the importance of each feature is assessed using the random forest algorithm. By combining multiple decision trees, random forest can effectively reduce overfitting and provide an importance score for each feature in the decision-making process. Based on these scores, features that contribute the most to intrusion detection can be retained, thereby enhancing the model's detection accuracy and efficiency.
2. Data division. The dataset is divided into training and test sets according to the ratio of 8:2. This division strategy aims to avoid data duplication and mitigate the overfitting problem.
3. Model parameter setting. In this paper, we set the parameters of the random forest model, including the number of trees, the number of features and the splitting criterion. Among them, the number of trees is set to 100 to balance the model performance and computational cost; the number of features is set to 4 to reduce the model complexity while retaining the key information; the splitting criterion is chosen to be the Gini coefficient, which is a measure of data impurity.
4. Model training and evaluation. In this paper, we use the RandomForestClassifier of sklearn library to construct a random forest classification model and use the training set for model training. Its construction algorithm is shown in Algorithm 1.

For the RF model, this study selected the following key hyperparameters:

1. Number of trees: 100. The number of trees is set to 100 to balance model performance and computational cost.
2. Max depth: 10. Maximum depth of 10 is sets to prevent overfitting.
3. Min samples split: 2. The minimum number of samples required to split a node is set to 2.
4. Min samples leaf: 1. The minimum number of samples required to be at a leaf node is set to 1.
5. Max features: sqrt (square root of the number of features). The maximum number of features considered for splitting at each node is set to square root of the total number of features to increase model randomness.

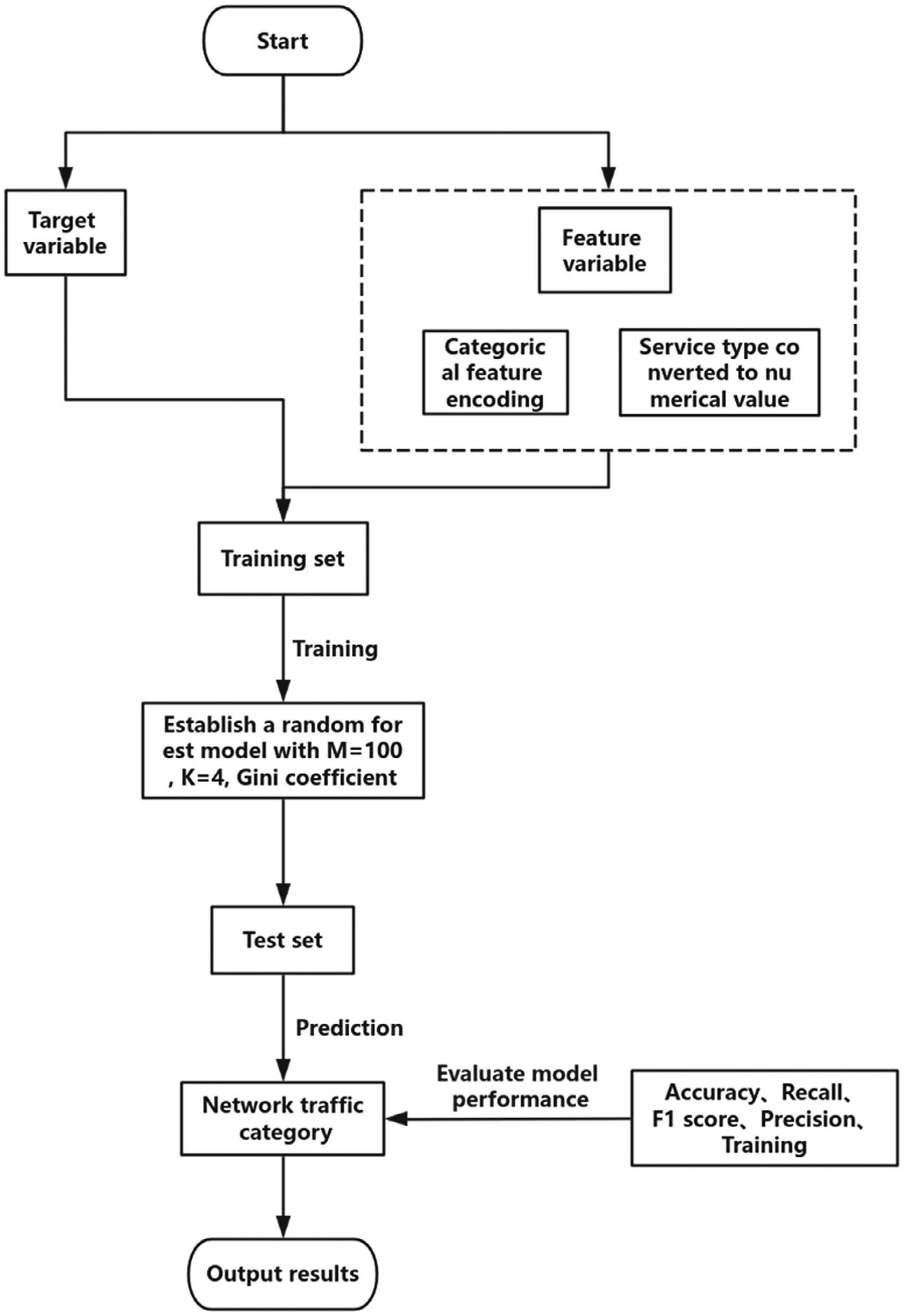

Start
Target variable
Feature variable
Categoric al feature encoding
Service type co nverted to nu merical value
Training set
Training
Establish a random for est model with M=100 , K=4, Gini coefficient
Test set
Prediction
Network traffic category
Evaluate model performance
Accuracy、Recall、F1 score、Precision、Training
Output results

Figure 2. Random forest model framework diagram.

Algorithm 1. Random forest algorithm.

```
Input: training set $D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$.
    The training set is an 80% random sampling of the dataset and contains the target and predictive variables.
    The number of trees is $M = 100$;
    the number of features per division is $K = 4$;

Output: random forest; $F = \{f_1, f_2, ..., f_M\}$

Algorithm steps:
Firstly, for: $i = 1, 2, ..., M$;
m samples are taken with replacement as subsets $D_i$ from the training set $D$;
The CART algorithm was used to generate a decision tree $f_i$ from $D_i$ in which only the optimal feature is selected
from among the randomly chosen features at each division;
Finally, return $F = \{f_1, f_2, ..., f_M\}$.
```

3.3. Integration of Federal Learning Frameworks

Federated learning is a distributed machine learning technique that aims to facilitate common modeling by multiple organizations or individuals while ensuring data privacy and compliance. In the process, the global model is distributed back to each participant to update their respective local models, while the data is always stored locally only and is not shared to other participants or uploaded to a central server. This mechanism significantly improves the privacy protection and security of data. Under this framework, this paper further embeds the CRNN model and the random forest model. The advantage is that it not only retains the original data privacy protection properties of federated learning, but also may produce performance enhancement due to the integrated application of multiple models. The flowchart of the whole model is shown in Figure 3.

In the federated learning framework, the communication protocol is responsible for coordinating and managing the exchange of parameters between different participating nodes. The steps are as follows:

1. Client update: Each client trains the model on local data and computes gradients or model parameter updates. These updates are securely transmitted to the central server through encryption to ensure data transfer security.

2. Parameter aggregation: The central server receives updates from each client and aggregates these updates using the federated averaging algorithm (FedAvg). This algorithm generates a global model update by computing the weighted average of the clients' parameter updates.

3. Model broadcast: The aggregated global model update is distributed to all clients so that they can continue with the local training. This process continues until the model converges.

The core of the model update aggregation process is the FedAvg algorithm, which is shown in Algorithm 2.

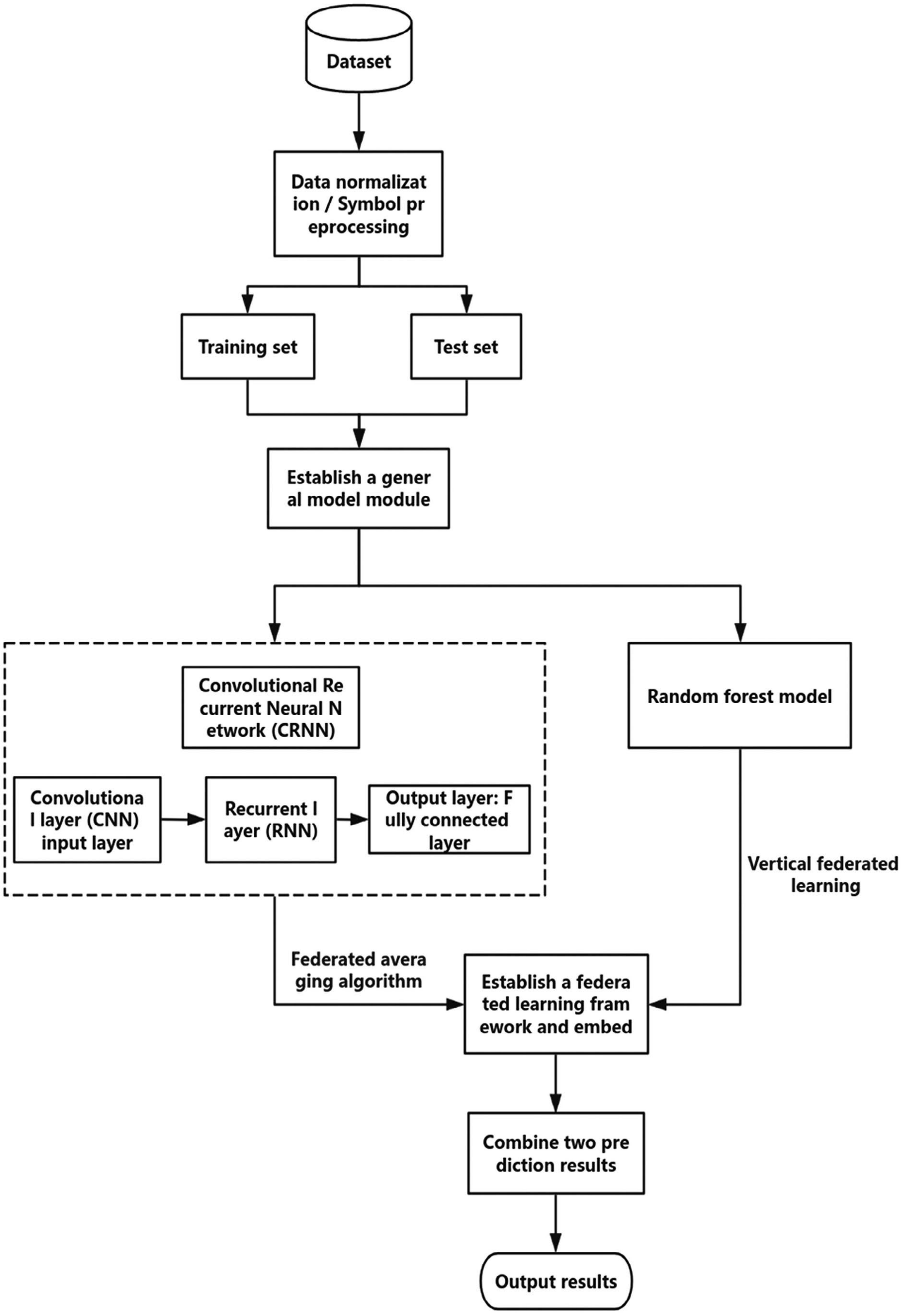

Dataset
Data normalizat ion / Symbol pr eprocessing
Training set
Test set
Establish a gener al model module
Convolutional Re current Neural N etwork (CRNN)
Convolutiona l layer (CNN) input layer
Recurrent l ayer (RNN)
Output layer: F ully connected layer
Random forest model
Vertical federated learning
Federated avera ging algorithm
Establish a federa ted learning fram ework and embed
Combine two pre diction results
Output results

Figure 3. Diagram of the fusion federated learning framework.

Algorithm 2. Federated averaging algorithm.

Assuming there are K clients participating, each client k has a local dataset D_k with size n_k. The global model update process includes the following steps:

1. Local computation: Each client trains using its local dataset D_k, calculating the gradient ΔW_k.
2. Weighted averaging: The central server performs a weighted average of the gradients uploaded by the clients, as shown in equation (17):

$$\Delta W = \frac{1}{N}\sum_{k=1}^{K} n_k \Delta W_k \tag{17}$$

   where $N = \sum_{k=1}^{K} n_k$ is the total sample size across all clients.

3. Model update: The server updates the global model parameters using the aggregated gradient, as shown in equation (18):

$$W_{t+1} = W_t - \eta \Delta W \tag{18}$$

   where η is the learning rate.

3.3.1. CRNN Model Embedded in a Federated Learning Framework

The combination of federated learning and CRNN provides an efficient and secure approach to distributed machine learning. In this framework, CRNN combines the strengths of CNNs and RNNs, where CNNs are responsible for automatically extracting valid features of sequential data, while RNNs focus on processing these features to capture their temporal dependencies, Figure 4. As a distributed machine learning strategy, federated learning allows multiple participants to jointly train a global model while protecting the privacy of their respective data. In the implementation, each participant performs initial training locally using its own raw data and then averages the model parameters from different clients by weighting them via a federated averaging algorithm to generate a more general and efficient global model.

In distributed machine learning, especially in the context of using CRNN models, data preparation has a crucial position as the first step. The process can be decomposed into the following core components: first, each participating client needs to complete local data collection. After data collection, two main preprocessing operations, normalization and encoding, follow. Normalization is responsible for standardizing data of different scales and ranges to enhance the efficiency of model training, which is especially important for input image data in CRNN models. Secondly, the encoding operation converts the non-numerical data into a numerical form and splits the data into training and test sets after preprocessing to evaluate the model performance. Then, through data loading and batch processing techniques, the data is organized into a format suitable for model training. Finally, these preprocessed and organized datasets are distributed to individual clients in preparation for the distributed model training that follows.

The initialization phase of the global model aims to provide a uniform starting point for training all participating clients. This process can be divided into the following steps: first, the basic architecture of the CRNN model needs to be specified, including its main constituent layers such as convolutional, cyclic, and fully connected layers, as well as the parameters required for these layers, such as convolutional kernel size and number of hidden units. Subsequently, the initialization of weights is performed on a central server W_{global}, ensuring that all clients start training from the same initial state. Next, a global model instance is created on the central server using these initialized weights and the predefined model architecture. Eventually, this initialized global model is distributed to each participating client as a starting point for their respective local model training.

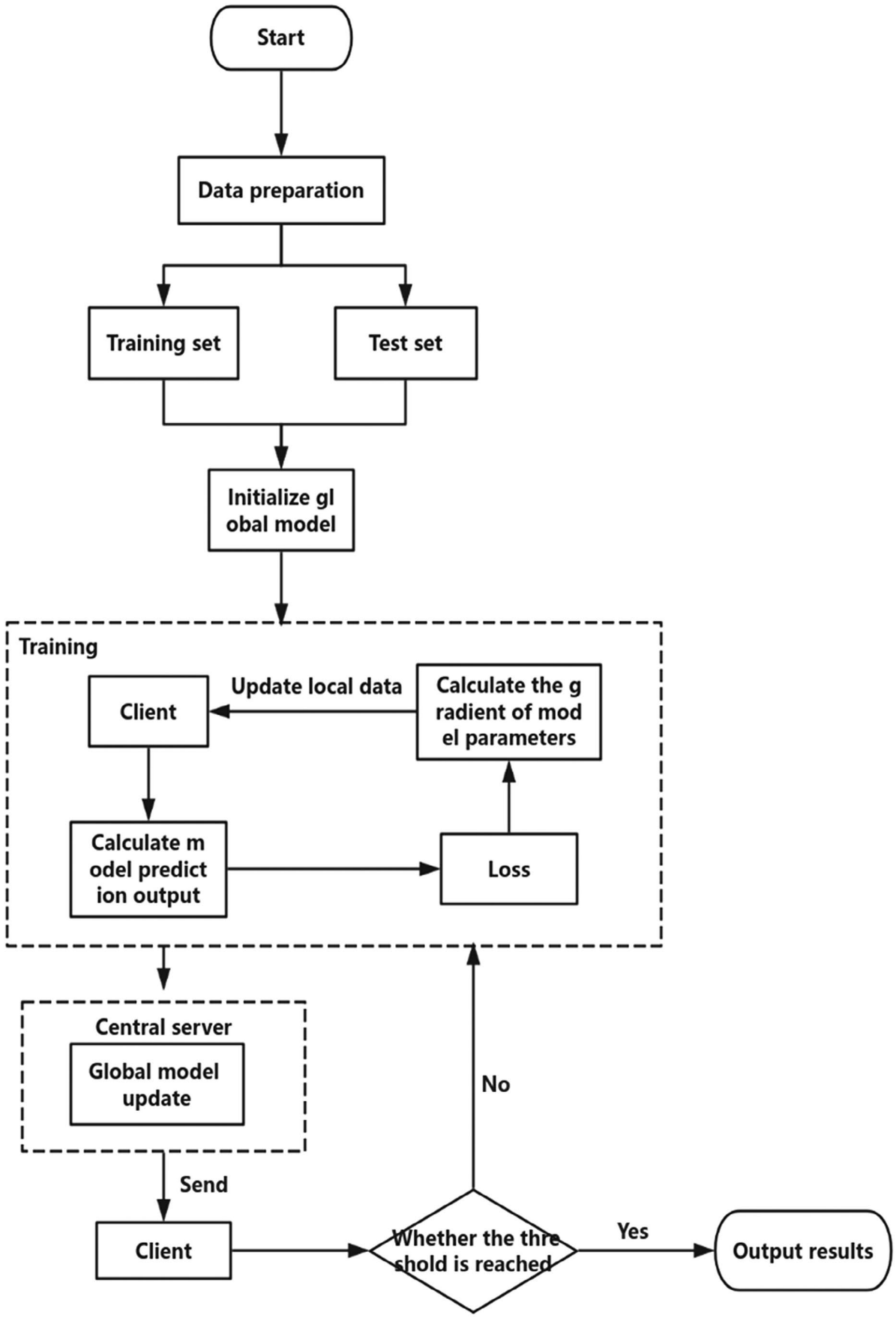


Figure 4. CRNN model embedded in a federated learning framework.

The purpose of local model training is to accurately adapt the models of each participating client to their respective local data. The process can be divided into the following steps: first, each client needs to load its own local dataset, which lays the foundation for subsequent model training. Second, forward propagation is performed to obtain the predicted output by utilizing the data and the current model parameters, and a loss function is used to compute the loss of the true labels. Then, backpropagation is performed based on this loss to compute the gradients of the model parameters. Finally, these gradients are used to update the client's local model parameters. As a result, each client trains an independent model that is highly compatible with its local dataset, *i.e.*, it generates a model that can be accurately adapted to the local data.

At the same time, each client gets the weights updated: ΔW_i, where i denotes the client. The loss is calculated as shown in equation (19):

$$\min_{W} F(W) = \min_{W} \sum_{k=1}^{K} \frac{N^{(k)}}{N} \cdot f^{(k)}(W) \tag{19}$$

where W are the global model parameters, $f^{(k)}(W)$ is the k-th client's loss function, $N^{(k)}$ is the sample size of the k-th client, and N is the total number of samples.

After each client has completed its local model training, the next step is the sharing of model parameters. At the core of this step, each client uploads the parameters of its local model, including weights and biases, to a central server. The advantage is that it allows individual clients to share their learning with other participants while protecting their data privacy. This sharing process is designed to facilitate the optimization of the global model while preserving the privacy of each participant's data.

The aggregated weights are updated using the federated averaging algorithm, where the central server collects the local model parameters (weights) sent by all clients and performs a weighted average of the model parameters from all clients, as shown in equation (20):

$$W_{global}^{new} = W_{global} + \frac{1}{N} \sum_{i=1}^{N} \Delta W_i \tag{20}$$

where N is the number of participants.

After the clients share the local model parameters, the central server is responsible for updating the global model weights using the aggregated weights to enhance performance and ensure learning from all participants' data, as shown in equation (21).

$$W_{global} = W_{global}^{new} \tag{21}$$

The central server updates the global model and distributes it to all clients, who receive it and update their local models to ensure consistency with the global model. Finally, the whole process is iterated several times to optimize the model performance. A convergence threshold is set as the criterion to stop training, and this threshold can be dynamically adjusted according to the model performance. Subsequently, the entire model training and updating process is repeated until the model's loss function reaches or exceeds the set convergence threshold, at which point training can be stopped.

3.3.2. Random Forest Models Embedded in a Federated Learning Framework

Random forest can improve the generalization ability and prediction performance of the model and can combine multiple decision tree models to complete the prediction. Secondly, by introducing the randomness, the degree of overfitting of the model to the training data can be reduced. In addition, data privacy is a key issue in federated learning, and the random forest model can protect the private features of participants to a certain extent by uploading only the local model parameters without sharing the original data, thus maintaining data privacy. Finally, the training process of random forests can be parallelized, which can fully utilize the computational resources of participants devices in federated learning and features distributed computing and scalability.

Therefore, when embedding the RF model into federated learning, it can achieve efficient and accurate learning of large-scale data dispersed across devices while protecting data privacy by utilizing the powerful classification and regression capabilities of random forest, Figure 5. This approach makes full use of the computational resources of each device and avoids the privacy risks associated with centralized data storage and transmission. At the same time, the random forest algorithm has a great advantage in dealing with complex and incomplete real-world data because of its ability to deal with high-dimensional data and missing value problems.

In this paper, model training is performed through the following five steps:

1. Data preparation phase: The dataset is loaded using the read_csv method of the pandas library; the total dataset is divided into three subsets, each containing 10,000 samples; these three subsets are assigned to the three participants A, B, and C, respectively.
2. Data preprocessing: The category of network traffic is selected as the target variable, and the service type, packet length, error fragment, emergency packet, *etc.* are selected as the feature variables; the train_test_split method of sklearn is used to further divide each subset into a training set and a test set, and the ratio of the test set is set to 0.2.
3. Federated learning framework construction: The FATE platform is used to implement vertical federated learning, so that the three participants can jointly train a random forest model without sharing the original data; a fate object is initialized and

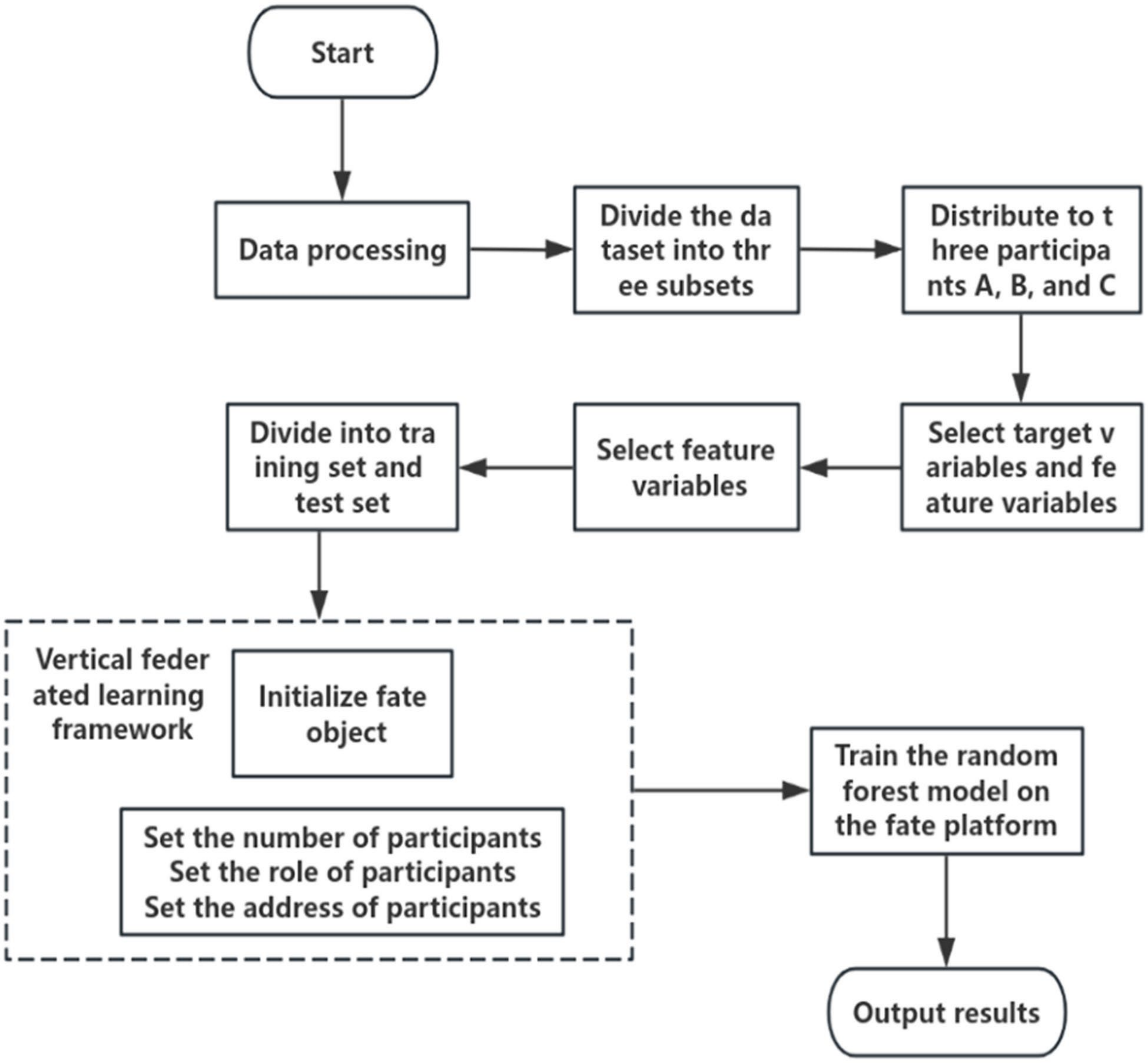

Start
Data processing
Divide the da taset into thr ee subsets
Distribute to t hree participa nts A, B, and C
Select target v ariables and fe ature variables
Select feature variables
Divide into tra ining set and test set
Vertical feder ated learning framework
Initialize fate object
Set the number of participants
Set the role of participants
Set the address of participants
Train the random forest model on the fate platform
Output results

Figure 5. Diagram of the vertical federated learning framework for the random forest model.

set according to the roles ("guest", "host", "arbiter") and addresses of the participants A, B, and C.

4. Model parameter setting and training: A random forest model is created on the FATE platform with the relevant hyperparameters set, including the number of trees (100), depth (10) and feature selection ratio (0.8); the number of training rounds is set to 10, and the evaluation indexes include "accuracy", "recall" and "F1 value".
5. Model evaluation and visualization: Performance evaluation is performed after completing the training of the model on the FATE platform; the visualization tools are used to show the learning curve and confusion matrix of the model.

Through the above steps, this paper successfully trains an RF model based on vertical federated learning framework with superior performance, which can not only handle high dimensional data and improve accuracy, but also protect data privacy and improve overall efficiency.

4. Results and Discussions

4.1. Experimental Environment and Assessment Indicators

4.1.1. Experimental Environment

The experimental training and testing in this paper were conducted under the Windows 11 operating system environment. The hardware configuration used includes two Intel Xeon Platinum 8380 processors, each with 40 cores and 80 threads, a main frequency of 2.3 GHz, a maximum RPM of 3.4 GHz, equipped with 60 MB of L3 cache and LGA4189 slot type, and the power consumption of each processor was 270 W. Python was chosen as the main development language, and PyCharm and Jupyter Notebook were chosen as the integrated development environment. For the deep learning library, PyTorch was chosen in this paper, as a widely used library, it is not only suitable for performing large-scale numerical computation, but also easy to build network models, which is very suitable for realizing the algorithmic models in this paper.

4.1.2. Assessment of Indicators

The experiment uses the following experimental evaluation metrics:

1. Accuracy is calculated as shown in equation (22):

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{22}$$

where TP (True Positives) denotes true positive cases, FP (False Positives) denotes false positive cases, FN (False Negatives) denotes false negative cases, and TN (True Negatives) denotes true negative cases; this formula calculates the classification accuracy of the model on the overall sample.

2. Precision is calculated as shown in equation (23):

$$Pre = \frac{TP}{TP + FP} \tag{23}$$

This formula quantifies the accuracy of the model when it predicts a positive case.

3. Recall is calculated as shown in equation (24):

$$Recall = \frac{TP}{TP + FN} \tag{24}$$

The formula measures the effectiveness of the model in identifying actual positive samples.

4. The F1 score (F1-score) is calculated as shown in equation (25):

$$F1\_score = 2\frac{Precision \cdot Recall}{Precision + Recall} \tag{25}$$

In this case, the relative contributions of *Precision* and *Recall* to the $F1$ score are equal. The $F1$ score takes values between 0 and 1, where 1 indicates the best performance and 0 the worst performance. This makes the formula an effective tool for evaluating the balance between model precision and recall ability.

5. Definition and calculation of AUC (Area Under Curve): the area under the ROC curve (Receiver Operating Characteris-

tic Curve) is an important indicator of the model's classification ability. Its value represents the ability of the model to distinguish between positive and negative samples: the larger the AUC value, the better the classification effect of the model. Its calculation formula is shown in equation (26):

$$AUC = \int_0^1 TPR(FPR)dFPR \tag{26}$$

where *TPR* represents the true case rate at the point on the ROC curve and *dFPR* represents the differential of the false positive case rate at the point on the ROC curve.

4.2. Dataset

The experiments in this paper use five datasets, the details of which are shown in Table 1. Each of the five datasets includes a training and a test set, and the amount of data in the training and test sets correspond to each target feature class.

4.3. Algorithmic Implementation of This Study

4.3.1. Parameters Required for the Experiment

As shown in Table 2, in this experiment, in order to configure the CRNN model, the study selects five key hyperparameters: the learning rate, the number of iterations, the loss function, the regularization, and the optimizer. Among them, the learning rate, as an important parameter to control the update rate of the model parameters, needs to be precisely adjusted based on the size and characteristics of the dataset. In the experimental setup, the learning rate was set to 0.001 to ensure that the model learns stably and effectively during the training process. Binary cross-entropy was chosen for the loss function, which helps to accurately calculate the error between the predicted and real values of the model. In order to suppress the overfitting risk of the model, L2 regularization was used, and

Table 1. Information table for the experimental dataset.

hallmark form	**NSL-KDD**		**KDDCup99**		**UNSW-NB15**		**CIC-IDS2017**		**CSE-CIC-IDS2018**	
	training set	**test set**	**training set**	**test set**	**training set**	**test set**	**training set**	**test set**	**training set**	**test set**
DoS	45927	7458	391458	229853	37000	4089	37000	4089	37000	4089
Probe	11656	2424	4107	4166	11656	2421	11656	2421	11656	2421
R2L	995	2754	1126	16189	995	2754	995	2754	995	2754
U2R	52	200	52	228	52	200	52	200	52	200
normal	67343	9711	97278	60593	56000	37000	56000	37000	56000	37000
aggregate quantitatively	125973	22544	494021	311029	105953	82332	105953	82332	105953	82332

Table 2. Information sheet on required parameters.

Hyperparameter name	Hyperparameter Meaning	set up
learning rate	Controlling the speed of model parameter updates	0.001
Number of iterations	Number of times the neural network worked on the entire training dataset	70
loss function	Differences between model predictions and true values	binary crossentropy
regularization	Preventing model overfitting	L2
optimizer	For optimizing model parameters	FedProx

its parameter value was set to 0.001, which can make the model parameters smoother. Finally, considering the complexity of the model structure in this paper, the non-uniform distribution characteristics of the dataset, and the training requirements of the model under the federated learning framework, the study chooses FedProx as the optimizer to optimize the model parameters and improve its overall performance.

4.3.2. Ablation Experiments

The paper evaluates the model's performance through ablation experiments, which involve incrementally increasing the model's complexity and observing changes in its output. The primary purpose of ablation experiments is to gain a deeper understanding of the model's working mechanisms while uncovering its strengths and potential weaknesses. As shown in Table 3, the results of the ablation experiments under different model configurations provide crucial insights for the model's evaluation and optimization.

As shown in Table 3, the ablation experiment results indicate that CNN and RNN exhibit consistent performance in terms of accuracy (ACC), precision (P), recall (R), and F1-score (F1), but their performance is significantly lower than that of CRNN and RF models. Specifically, CRNN outperforms CNN and RNN across all performance metrics. This advantage mainly stems from CRNN's ability to combine the feature extraction capabilities of CNN with the sequential modeling capabilities of RNN, enabling it to more effectively capture spatio-temporal information, thus excelling in network intrusion detection tasks.

Table 3. Table of ablation experiments' results.

Architecture	P	R	F1	ACC	AUC
convolutional neural network	0.765	0.766	0.766	0.787	0.909
recurrent neural network	0.765	0.766	0.766	0.787	0.909
convolutional recurrent neural network	0.999	0.998	0.998	0.998	0.999
random forest	0.980	0.970	0.980	0.980	0.990
The federated learning framework	0.999	0.999	0.999	0.999	0.999

By integrating CRNN and RF models into the federated learning (FL) framework, a comprehensive model proposed in this paper is formed. Experimental results show that the proposed algorithm achieves further improvement in all metrics. Specifically, the proposed algorithm approaches near-perfect performance in accuracy, precision, recall, F1-score, and AUC, significantly outperforming the standalone use of CRNN or RF.

The superior performance of the proposed method can be attributed to the following factors:

CRNN model: By combining the feature extraction capabilities of CNN with the sequential modeling capabilities of RNN, it effectively handles high-dimensional, nonlinear, and dynamically changing data features, significantly enhancing detection accuracy and robustness.

RF model: Offers strong feature selection and noise reduction capabilities, further enhancing the model's generalization ability and robustness.

FL framework: Through distributed data processing, it reduces the need for data centralization and transmission, effectively protecting data security and privacy. Additionally, the FL framework can leverage the computational resources of multiple participants, accelerating the training process and improving model efficiency and performance.

However, despite the significant performance advantages of the proposed method, it has high computational complexity, long training times, and substantial hardware resource requirements. These factors may present challenges in practical applications, necessitating trade-offs and optimizations during actual deployment.

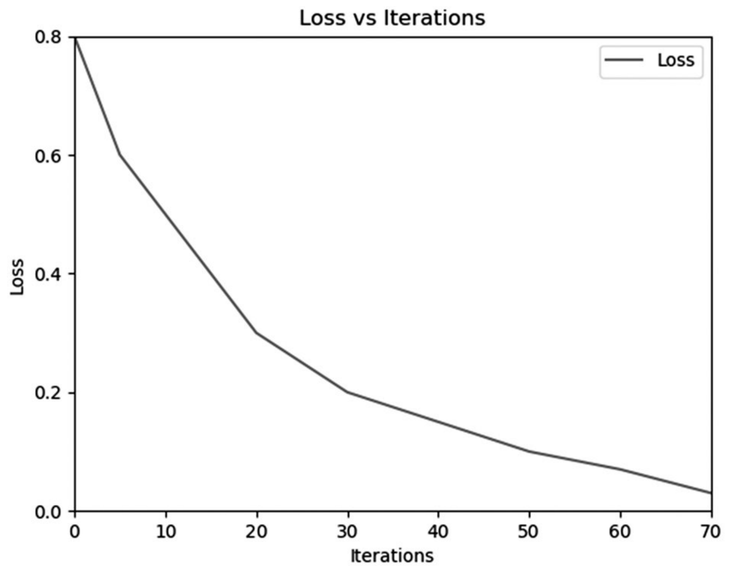
4.3.3. Comparison of Experimental Results with Different Hyperparameters

In this paper, the performance of the algorithm is evaluated for different number of iterations (from 0 to 70). As shown in Figure 6, the loss value (Loss) of the model gradually decreases while the accuracy (Accuracy) gradually increases as the number of iterations increases. This trend indicates that with more iterations, the model parameters are able to fit the training data more efficiently. The loss value is a key measure of the difference between the model's predicted results and the true labels, while the accuracy is an important metric for evaluating the correctness of the model's predictions. The reduction of the loss value and the improvement of the accuracy rate together indicate a significant enhancement of the performance of the algorithm in this paper. The reason for this effect is that as the iterations proceed, the model continuously adjusts its internal parameters to reduce the prediction error by learning the features and patterns in the training data. During each iteration, the algorithm optimizes the model parameters based on the feedback from the loss function, resulting in a decrease in the loss value and an increase in the accuracy. This continuous parameter optimization process allows the model to predict new samples more accurately, demonstrating the effectiveness of the algorithm in learning and generalization.

4.3.4. Analysis of Computational Complexity, Scalability, and Communication Overhead

The Convolutional Recurrent Neural Network (CRNN) proposed by the research institute combines the feature extraction capabilities of Convolutional Neural Networks (CNN) with the temporal modeling capabilities of Recurrent Neural Networks (RNN). This allows the CRNN to effectively handle high-dimensional, nonlinear, and dynamically changing data features. The computational complexity of each convolutional layer is $O(n \cdot k \cdot c)$, where n is the input data size, k is the kernel size, and c is the number of output channels. The computational complexity of the recurrent layer is $O(t \cdot h)$, where t is the number of time steps and h is the hidden layer size. Despite the high computational complexity of the model, optimization algorithms and hardware acceleration can effectively improve training speed and inference efficiency.

Through experiments, the research obtained data on the model's performance and communication overhead with different numbers of nodes, as shown in Table 4.



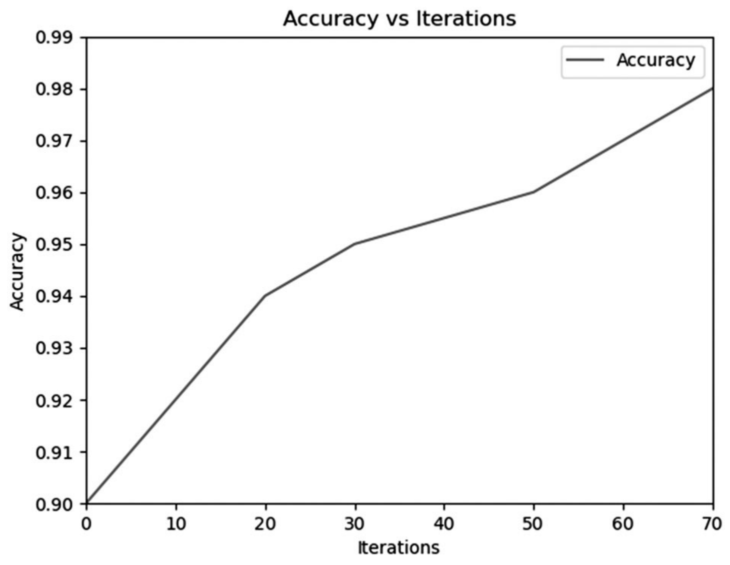


Figure 6. Plot of the number of iterations on the loss value and accuracy of this paper's algorithm.

Table 4. Model Performance and Communication Overhead with Different Numbers of Nodes.

Number of Nodes	Accuracy	Recall	F1 Score	AUC	Communication Overhead (GB)
1	0.85	0.82	0.83	0.87	1.00
2	0.88	0.85	0.86	0.89	1.50
5	0.90	0.87	0.88	0.91	2.56
10	0.91	0.89	0.90	0.92	3.85
20	0.94	0.93	0.94	0.95	5.77
50	0.97	0.96	0.97	0.98	9.65
100	0.999	0.999	0.999	0.999	15.00

From Table 4, it is evident that, as the number of nodes increases, the model's performance gradually improves. When the number of nodes reaches 100, the performance metrics (accuracy, recall, F1 score, and AUC) all reach 0.999. This indicates that increasing the number of nodes can significantly enhance the overall performance of the model. This improvement is attributed to the additional training data and computational resources provided by more nodes, which enhance the model's generalization ability and robustness. However, the increase in the number of nodes also leads to higher computational complexity, as each node needs to independently train the model, and the global model update requires aggregating more local model parameters.

The communication overhead significantly rises with the increase in the number of nodes. This increase in communication overhead can affect the efficiency of federated learning, particularly when the number of nodes is large. Nevertheless, by reasonably planning the communication frequency and data volume, it is possible to improve model performance while controlling communication overhead, ensuring the efficiency and scalability of federated learning.

Although the increase in the number of clients results in higher communication overhead and computational complexity, the model performance remains consistently high (with performance metrics reaching 0.999 when the number of nodes is 100). This demonstrates that increasing the number of clients can significantly enhance the overall performance of the model, mainly because more clients provide richer data samples, which help improve the model's training effectiveness.

Based on these results, it can be concluded that the proposed method is highly practical and effective in the context of federated learning. In practical applications, the number of nodes and communication frequency can be flexibly adjusted according to specific needs and available resources to achieve optimal model performance and system efficiency.

4.4. Comparison of this Study's Algorithm with Other Intrusion Detection Methods

4.4.1. Model Detection Performance Comparison

The algorithm in this paper was applied to the NSL-KDD dataset and compared with other models in terms of detection performance. The results of this comparison show the performance of the algorithm on this dataset, as shown in Table 5.

Table 5. Table of model performance comparison results.

mould	**P**	**R**	**F1**	**ACC**	**AUC**
FL-SEResNet	0.98	0.97	0.975	0.98	0.98
DBN	0.980	0.978	0.979	0.979	0.995
LSTM	0.984	0.982	0.983	0.983	0.996
NDAE	0.820	0.820	0.820	0.815	0.910
KNN	0.930	0.920	0.925	0.930	0.930
TL-NID	0.96	0.95	0.955	0.96	0.960
The algorithm in this study	0.999	0.999	0.999	0.999	0.999

The performance of this study's algorithm is compared with other commonly used machine learning and deep learning methods on network intrusion detection tasks. These comparison models include deep belief networks (DBN), long short-term memory networks (LSTM), asymmetric deep self-encoder (NDAE), and k-nearest neighbor algorithm (KNN), as well as an image classification model based on deep residual networks, feature-level attention mechanisms (FL-SEResNet), and a network intrusion detection model based on migration learning (TL-NID). The latter consists of a cascade of two convolutional neural networks.

As it can be seen from the results shown in Table 5, the algorithmic model proposed in this study outperforms the other models in all performance metrics, especially in the accuracy rate of 99.9%, which is 1.9% higher than the best comparison model, FL-SEResNet. This excellent performance demonstrates the effectiveness and superiority of this model for network intrusion detection tasks. This is mainly due to the feature extraction capability of this model on network traffic data. In addition, the recurrent neural network is able to capture long-term dependencies in time-series data, an ability that plays a crucial role in the network intrusion detection task.

Experiments on the NSL-KDD dataset show that compared with the classical BP neural network algorithm, the convolutional recurrent neural network model demonstrates advantages in terms of training time, detection accuracy and false alarm rate. Overall, the convolutional recurrent neural network model in this paper effectively integrates the advantages of deep learning techniques and classifiers, and can accurately classify network traffic data, thus demonstrating significant effectiveness and superiority in network intrusion detection tasks.

4.4.2. Comparison of the Results of this Study's Algorithm in Five Attack Categories and the Detection Performance of Other Models

The purpose of this experiment is to evaluate the performance of the proposed convolutional recurrent neural network intrusion detection model and compare it with a variety of commonly used machine learning models. Like in the previous experiment, these comparison models include DBN, LSTM, NDAE, KNN, FL-SEResNet, and TL-NID. To fully measure the detection performance of these models, five attack types are used and evaluated using five performance metrics described in 4.1.2.

Table 6 shows the detection results of this paper's algorithm on five attack categories and the performance comparison with other models.

Figure 7 shows the confusion matrices of each model across the five attack types.

As shown in Table 6 and Figure 7, the algorithm presented in this paper demonstrates excellent classification performance across five attack categories. These superior performances can be attributed to several key factors. Firstly, the algorithm utilizes multimodal feature fusion, combining the advantages of CNN and RNN. This allows it to capture both temporal and spatial features of network traffic. CNNs excel at extracting local features, while RNNs are proficient in handling sequential data. The combination enables the model to comprehensively understand and classify network attack behaviors. Secondly, the application of federated learning allows the model to be trained across multiple nodes without centralized data aggregation. This not only enhances data privacy and security but also improves the model's generalization capability by learning from distributed, heterogeneous data. Finally, the use of efficient optimization strategies, such as the Adam optimizer, ensures that the model converges more quickly to the global optimum, avoiding local minima. Additionally, adjusting the learning rate and employing regularization techniques further enhances the model's stability and robustness.

Table 6. Comparison of model results under five attack categories.

mould	Type of attack	P	R	F1	ACC	AUC
FL-SEResNet	DoS	0.980	0.970	0.975	0.98	0.98
	Probe	0.970	0.960	0.965	0.97	0.97
	U2R	0.960	0.950	0.955	0.96	0.96
	R2L	0.950	0.940	0.945	0.95	0.95
	Normal	0.980	0.980	0.980	0.98	0.98
TL-NID	DoS	0.960	0.950	0.955	0.960	0.960
	Probe	0.950	0.940	0.945	0.950	0.950
	U2R	0.940	0.930	0.935	0.940	0.940
	R2L	0.930	0.920	0.925	0.930	0.930
	Normal	0.960	0.960	0.960	0.960	0.960
DBN	DoS	0.981	0.982	0.982	0.982	0.996
	Probe	0.978	0.976	0.977	0.977	0.995
	U2R	0.969	0.968	0.969	0.969	0.992
	R2L	0.974	0.972	0.973	0.973	0.993
	Normal	0.969	0.968	0.969	0.969	0.992
LSTM	DoS	0.983	0.984	0.984	0.984	0.996
	Probe	0.980	0.978	0.979	0.979	0.995
	U2R	0.971	0.970	0.971	0.971	0.993
	R2L	0.976	0.974	0.975	0.975	0.994
	Normal	0.990	0.988	0.989	0.989	0.998
NDAE	DoS	0.990	0.960	0.970	0.970	0.980
	Probe	0.710	0.800	0.750	0.760	0.880
	U2R	0.220	0.290	0.250	0.250	0.640
	R2L	0.540	0.570	0.550	0.550	0.770
	Normal	0.930	0.970	0.950	0.950	0.980
tKNN	DoS	0.999	0.999	0.999	0.999	0.999
	Probe	0.998	0.998	0.998	0.998	0.998
	U2R	0.987	0.987	0.987	0.987	0.987
	R2L	0.996	0.996	0.996	0.996	0.996
	Normal	0.998	0.998	0.998	0.998	0.998
The algorithm in this study	DoS	0.985	0.986	0.986	0.986	0.997
	Probe	0.982	0.978	0.980	0.980	0.996
	U2R	0.971	0.970	0.971	0.971	0.993
	R2L	0.976	0.974	0.975	0.975	0.994
	Normal	0.992	0.988	0.990	0.990	0.999

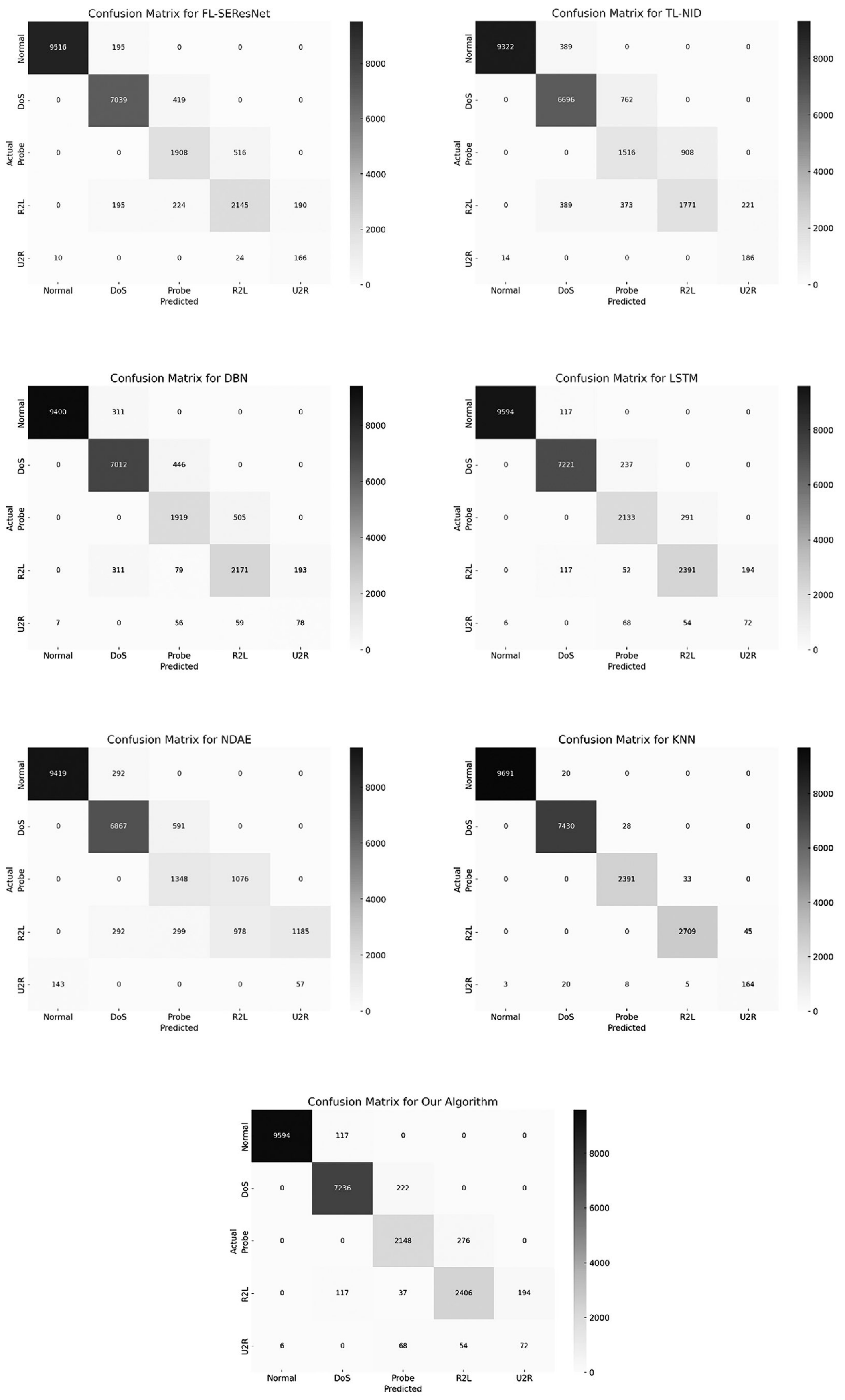


Figure 7. Confusion matrices of different models across five attack types.

When compared with other state-of-the-art techniques, the superiority of the proposed algorithm is also evident. The proposed algorithm performs better in handling small sample categories (*e.g.*, U2R) and effectively processes heterogeneous data, making it suitable for federated learning scenarios. In contrast, FL-SEResNet performs well in Normal and DoS categories but has weaker capabilities in handling heterogeneous data. TL-NID shows stable performance in most attack categories but struggles with complex attack patterns like R2L and U2R. DBN performs well in multi-class attack detection, especially in DoS and Probe categories, but has a long training time and poor adaptability to dynamically changing network attacks. LSTM excels in handling sequential data but has high computational complexity and long training times. NDAE performs adequately in Normal and DoS categories but poorly in Probe, U2R, and R2L categories, with a high false positive rate. KNN performs well across nearly all categories but has high computational complexity, resulting in long training and prediction times, especially with large datasets.

In practical network intrusion detection applications, the proposed algorithm has significant practical implications. Firstly, the federated learning framework enables training across multiple distributed nodes, adapting to large-scale distributed network environments, effectively addressing data privacy concerns, and enhancing system scalability. Secondly, by combining the strengths of CNN and RNN, the model excels in handling diverse and complex network attacks. Multimodal feature fusion and efficient optimization strategies allow it to maintain high precision and recall rates across various attack types. Additionally, the application of regularization techniques and optimization strategies further strengthens the model's robustness, reducing the risk of overfitting. Lastly, the algorithm demonstrates strong adaptability to evolving network threats. Through federated learning and multimodal feature fusion, the model can learn from heterogeneous data from different nodes, effectively capturing new attack patterns and adapting to dynamically changing network environments.

The proposed framework also shows potential for widespread applications in various fields such as IoT, cloud computing, and industrial control systems. For IoT, traditional centralized intrusion detection methods struggle to meet the needs of vast and widely distributed devices. The federated learning framework can train models in a distributed manner across IoT nodes, protecting data privacy while improving detection efficiency and accuracy. In cloud computing environments, centralized data storage and processing face significant security risks. The proposed algorithm can train and update models across multiple cloud nodes, enhancing the adaptability and robustness of the detection system. In industrial control systems, highly sensitive data and operational environments require extremely high security. The proposed algorithm, through federated learning and multimodal feature fusion, can effectively perform intrusion detection in distributed industrial control systems, ensuring system security. The high accuracy and recall rates further enhance the protective capabilities of industrial control systems.

5. Conclusion

In this paper, we proposed a novel network intrusion detection framework that combines convolutional recurrent neural networks and random forest models within a federated learning setting. The proposed approach effectively addresses the challenges of data privacy, computational efficiency, and model generalization in traditional network intrusion detection methods. By leveraging the strengths of CRNN and RF, the framework achieves high accuracy and robustness in detecting various types of network attacks.

The extensive experimental results on benchmark datasets demonstrate the superiority of the proposed method compared to state-of-the-art techniques, consistently outperforming them in terms of accuracy, precision, recall, F1 score, and AUC. The integration of federated learning enables collaborative model training while preserving data privacy, making the proposed approach suitable for real-world deployment.

The proposed framework has significant implications for the field of cybersecurity, offering a promising solution for secure and efficient network intrusion detection. It can be applied to various domains, such as IoT, cloud computing, and industrial control systems, to enhance their security posture and protect against evolving threats. However, there are still challenges and opportunities for future research. Further investigations can focus on improving the adaptability and scalability of the proposed method, addressing the dynamic nature of network traffic and the heterogeneity of participating clients. The incorporation of advanced techniques, such as attention mechanisms, adaptive learning rates, and differential privacy, could further enhance the performance and security of the framework.

In conclusion, the proposed network intrusion detection framework based on CRNN, RF, and federated learning represents a significant step forward in the field of cybersecurity. It offers a powerful and flexible solution for detecting and mitigating network attacks while preserving data privacy. The insights gained from this research can guide future efforts in developing more robust and efficient intrusion detection systems, ultimately contributing to a safer and more secure cyberspace.

References

[1] R. Creemers *et al.*, "Translation: Cybersecurity Law of the People's Republic of China" in DigiChina, 2017. http://digichina.stanford.edu/work/translation-cybersecurity-law-of-the-peoples-republic-of-china-effective-june-1-2017/

[2] S. Jian *et al.*, "Overview of Network Intrusion Detection Technology", *Journal of Cyber Security*, vol. 5, no. 4, pp. 96–122, 2020. http://dx.doi.org/10.19363/J.cnki.cn10-1380/tn.2020.07.07

[3] C. Zheng *et al.*, "Intrusion Detection Based on Federated Learning and Deep Residual Networks", *Computer Applications*, vol. 43, no. S1, pp. 133–138, 2023. http://www.joca.cn/EN/10.11772/j.issn.1001–9081.2022081222

[4] B. Li *et al.*, "DeepFed: Federated Deep Learning for Intrusion Detection in Industrial Cyber–Physical Systems", *IEEE Trans. Industr. Inform.*, vol. 17, no. 8, pp. 5615–5624, 2021. https://doi.org/10.1109/tii.2020.3023430

[5] T. Zhang *et al.*, "Federated Learning for Internet of Things", in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, 2021, pp. 413–419. https://doi.org/10.48550/arXiv.2106.07976

[6] S. Agrawal *et al.*, "Federated Learning for Intrusion Detection System: Concepts, Challenges and Future Directions", *Comput. Commun.*, vol. 195, pp. 346–361, 2022. https://doi.org/10.1016/j.comcom.2022.09.012

[7] A. Belenguer *et al.*, "GöwFed: A Novel Federated Network Intrusion Detection System", *Journal of Network and Computer Applications*, vol. 217, no. 103653, p. 103653, 2023. https://doi.org/10.48550/arXiv.2210.16441

[8] O. Belarbi *et al.*, "Federated Deep Learning for Intrusion Detection in IoT Networks", in *Proceedings of the GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, 2023. https://doi.org/10.1109/GLOBECOM54140.2023.10437860

[9] X. Sun *et al.*, "A Hierarchical Federated Learning-based Intrusion Detection System for 5G Smart Grids", *Electronics (Basel)*, vol. 11, no. 16, pp. 2627, 2022. https://doi.org/10.3390/electronics11162627

[10] S. Hao *et al.*, "A Network Intrusion Detection Model Based on Efficient Federated Learning Algorithm", *Journal of Computer Applications*, vol. 43, no. 4, pp. 1169–1175, 2023.

[11] K. Li, "A Network Intrusion Detection Model Based on Efficient Federated Learning Algorithm", *Liaoning Normal University*, no. 12, pp. 1–59, 2022. https://link.cnki.net/doi/10.27212/d.cnki.glnsu.2022.000538

[12] H. Zhang *et al.*, "Summary of Intrusion Detection Models Based on Deep Learning", *Computer Engineering and Applications*, vol. 58, no. 6, pp. 17–28, 2022.

[13] R. Vinayakumar *et al.*, "Applying Convolutional Neural Network for Network Intrusion Detection", in *Proceedings of the 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 1222–1228, 2017. https://doi.org/10.1109/ICACCI.2017.8126009

[14] C. Yin *et al.*, "A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks", *IEEE Access*, vol. 5, pp. 21954–21961, 2017. https://doi.org/10.1109/access.2017.2762418

[15] X. Duan *et al.*, "Network Traffic Anomaly Detection Method Based on Multi-scale Residual Classifier", Computer Communications, vol. 198, pp. 206–216, 2023. https://doi.org/10.1016/j.comcom.2022.10.024

[16] H. Qian, "Machine Learning-based Smart Device Identification and Security Management for Internet of Things", *Southeast University*, no. 6, pp. 1–67, 2022. https://link.cnki.net/doi/10.27014/d.cnki.gdnau.2021.001412

[17] L. Tang *et al.*, "Review of Deep Learning for Short Text Sentiment Tendency Analysis", *Journal of Frontiers of Computer Science and Technology*, vol. 15, no. 5, pp. 794–811, 2021.

[18] S. Ma, "Research on Stereoscopic Video Quality Evaluation Method Based on Convolutional Neural Network", *Tianjin University*, no. 1, pp. 1–67, 2022. https://link.cnki.net/doi/10.27356/d.cnki.gtjdu.2019.002916

[19] J. Ma, "Research on Image Verification Code Recognition Algorithm Based on Deep Learning", *Shenyang Normal University*, no. 9, pp. 1–47, 2021. https://link.cnki.net/doi/10.27328/d.cnki.gshsc.2021.000149

[20] Z. Xie, "Research on Knowledge Base Q&A Technology Based on Deep Learning", *Central China Normal University*, no. 1, pp. 1–68, 2019.

[21] Y. Li *et al.*, "Network Anomaly Detection Based on TCM-KNN Algorithm", in *Proceedings of the 2nd ACM symposium on Information, Computer and Communications Security*, 2007. https://doi.org/10.1145/1229285.1229292

[22] Y. Imrana *et al.*, "A Bidirectional LSTM Deep Learning Approach for Intrusion Detection", *Expert Syst. Appl.*, vol. 185, no. 115524, p. 115524, 2021. https://doi.org/10.1016/j.eswa.2021.115524

[23] M. Masum and H. Shahriar, "TL-NID: Deep Neural Network with Transfer Learning for Network Intrusion Detection", in *Proceedings of the 15th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2020. https://doi.org/10.23919/ICITST51030.2020.9351317

[24] D. Liang and P. Pan, "Research on Intrusion Detection Based on Improved DBN-ELM", in *Proceedings of the 2019 International Conference on Communications, Information System and Computer Engineering (CISCE)*, 2019. https://doi.org/10.1109/CISCE.2019.00115

[25] N. Shone *et al.*, "A Deep Learning Approach to Network Intrusion Detection", *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41–50, 2018. https://doi.org/10.1109/tetci.2017.2772792

[26] B. Gan *et al.*, "A Convolutional Neural Network Intrusion Detection Method Based on Data Imbalance", *The Journal of Supercomputing*, vol. 78, no. 18, pp. 19401–19434, 2022. https://doi.org/10.1007/s11227-022-04633-x

[27] Z. Li *et al.*, "Intrusion Detection Using Convolutional Neural Networks for Representation Learning", in *Neural Information Processing, Cham: Springer International Publishing*, 2017, pp. 858–866. https://doi.org/10.1007/978-3-319-70139-4_87

[28] Q. Xu, "A Text Categorization Method for Smart Devices Based on Natural Language Processing", *Shanghai University of Applied Sciences*, no. 3, pp. 1–62, 2021. https://link.cnki.net/doi/10.27801/d.cnki.gshyy.2021.000150

[29] A. N. Sokolov *et al.*, "Traffic Modeling by Recurrent Neural Networks for Intrusion Detection in Industrial Control Systems", in *Proceedings of the 2019 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*, 2019. https://doi.org/10.1109/ICIEAM.2019.8742961

[30] N. Farnaaz and M. A. Jabbar, "Random Forest Modeling for Network Intrusion Detection System", *Procedia Computer Science*, vol. 89, pp. 213–217, 2016. https://doi.org/10.1016/j.procs.2016.06.047

[31] T. Markovic *et al.*, "Random Forest Based on Federated Learning for Intrusion Detection", in *AIAI 2022. IFIP Advances in Information and Communication Technology*, 2022, pp. 132–144. https://doi.org/10.1007/978-3-031-08333-4_11

[32] H. B. McMahan *et al.*, "Communication-efficient Learning of Deep Networks from Decentralized Data", in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017, pp. 1273–1282.

[33] L. Yang *et al.*, "Resource Management Algorithm of Indoor Visible Light Communication and WiFi Heterogeneous Network", in *Optoelectronic Devices and Integration X*, 2021. https://doi.org/10.1117/12.2600722

[34] R. Zhu, "Research on Cyberbullying Detection Technology Based on Deep Neural Network", *Nanjing University of Science and Technology*, no. 3, pp. 1–83, 2021. https://link.cnki.net/doi/10.27245/d.cnki.gnjsu.2021.000233

Received: May 2024
Revised: June 2024
Accepted: June 2024

Contact addresses:
Qianying Zou
Geely University of China
Chengdu
China
e-mail: zouqianying@guc.edu.cn

Yushi Li
Chengdu College of University of Electronic Science and Technology of China
Chengdu
China
e-mail: 1406458279@qq.com

Xinyue Jiang
Chengdu College of University of Electronic Science and Technology of China
Chengdu
China
e-mail: 1797717885@qq.com

Yuepeng Zan
Chengdu College of University of Electronic Science and Technology of China
Chengdu
China
e-mail: 2017829598@qq.com

Fengyu Liu*
Geely University of China
Chengdu
China
e-mail: liufengyu@guc.edu.cn
*Corresponding author

Qianying Zou received the B.S. degree in information engineering from Chengdu University of Technology, Chengdu, China, in 2003, and the M.S. degree in computer applications from Chengdu University of Technology, Chengdu, China, in 2006. Her major research interests include big data applications and artificial intelligence.

Yushi Li is currently pursuing his B.S. degree in computational science at the University of Electronic Science and Technology of China, Chengdu College, Chengdu, China. He is in his fourth year of study. His primary research focus lies in big data applications.

Xinyue Jiang is currently pursuing his B.S. degree in computational science at the University of Electronic Science and Technology of China, Chengdu College, Chengdu, China. His research interest is in computational science.

Yuepeng Zan is currently pursuing his B.S. degree in computational science at the University of Electronic Science and Technology of China, Chengdu College, Chengdu, China. His primary research focus lies in big data applications.

Fengyu Liu received the B.S. degree in software engineering from the University of Electronic Science and Technology of China, Chengdu College, Chengdu, China, in 2018. He is currently a teaching assistant. His primary research interests include computer vision and front-end/back-end design.