Intrusion Detection Using Multi-objective Evolutionary Convolutional Neural Network for Internet of Things in Fog Computing

Yi Chen[1], Qiuzhen Lin[1]*, Wenhong Wei[2], Junkai Ji[1], Ka-Chun Wong[3], Carlos A. Coello Coello[4]

[1]*College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, PR. China*

[2]*School of Computer Science and Technology, Dongguan University of Technology, Dongguan, PR. China*

[3]*Department of Computer Science, City University of Hong Kong, Hong Kong, PR China*

[4]*CINVESTAV-IPN, Department of Computer Science, Mexico, D.F. 07360, Mexico*

Abstract:

Our world is moving fast towards the era of the Internet of Things (IoT), which connects all kinds of devices to digital services and brings significant convenience to our lives. With the rapid increase in the number of devices connected to the IoT, there may exist more network vulnerabilities, resulting in more network attacks. Under this dynamic IoT environment, an effective intrusion detection system (IDS) is urgently needed to detect attacks with low-latency and high accuracy. A number of promising IDSs have been proposed based on deep learning (DL) techniques, but they need to do parameter tuning under different environments, which is very time-consuming. To alleviate this problem, this paper proposes a multi-objective evolutionary convolutional neural network for intrusion detection system, called MECNN, which is run on the fog nodes of Fog computing on IoT. In this approach, convolutional neural network (CNN) is used as the classifier to detect intrusions and the multi-objective evolutionary algorithm based on decomposition (MOEA/D) algorithm is modified to evolve the CNN model, which greatly simplifies the parameter tuning process of DL. To be specific, a novel encoding scheme is first proposed to transform the topological architecture of CNN into a chromosome of MOEA/D and then the two conflicting objectives, i.e., detection performance and model complexity of the CNN model, are simultaneously optimized by MOEA/D, which can obtain a number of IDSs with various detection performance and model complexities. Then, the most suitable MECNN model can be deployed in different fog nodes of Fog computing, providing low-latency and high-accuracy intrusion detection for IoT. Finally, the experimental studies are conducted on two popular datasets (AWID and CIC-IDS2107), which have validated that our MECNN model can improve detection performance and robustness to better protect the IoT when compared to other state-of-the-art IDSs.

Keywords: Intrusion detection system, Internet of Things, Multi-objective optimization, Fog computing, Convolutional neural network, Neuroevolution.

1. Introduction

Internet of Things (IoT) technology is considered to be the fourth industrial revolution [1]. It is a system including interrelated computing devices, mechanical and digital machines, sensors, software, and other technologies, which can connect and exchange data with various devices and systems over the Internet without human-to-human or human-to-computer interaction [2]. With IoT, our daily life becomes more

convenient and well organized, which has been widely used in many application fields such as manufacturing, smart home, medical care, transportation, and so on [3]. As IoT plays an increasingly significant role in our daily life and production work, it is essential to ensure the security of the network and information protection in IoT applications [4]. Otherwise, it may lead to a significant disaster or even casualties.

Identifying unknown intrusions is a challenging task for IoT as it usually connects a wide range of devices with different computational powers, battery capacities, network protocols, and operating systems (OS) [5]. This heterogeneity poses a challenge to the deployment of security schemes. Most of the existing security approaches are passive security defense [6], including security gateway, firewall, code signature, and encryption technology, which cannot detect and respond proactively [7]. An intrusion detection system for IoT (called IoT-IDS) will capture and analyze IoT data traffic records, which then detects network attacks or abnormal behaviors. When the attack occurs, IoT-IDS will respond in time to intercept the attack. This method can proactively defend against attacks, which is suitable as a security precaution for IoT networks [8].

To improve the performance of IoT-IDS, machine learning (ML) approaches have been employed to design IDS, such as decision tree (DT) [9], naïve bayes (NB) [10], and support vector machine (SVM) [11]. The ML methods have shown promising performance to recognize the important features in IoT traffic, as they can proficiently identify network intrusions and attacks. However, when handling large-scale or high-dimensional traffic data, their detection performance will be degraded dramatically and their required time for detection will be increased significantly [12]. Therefore, the ML methods may not be so effective for the modern Internet environment due to the increasing traffic and bandwidth.

On the other hand, deep learning (DL) techniques seem more effective in handling the increasingly complex intrusion detection problems of IoT [13]. Some DL methods have been used to build IoT-IDS, such as auto-encoder (AE) [14], generative adversarial network (GAN) [12], and deep neural network (DNN) [15]. Nevertheless, the design of neural networks is a tricky process, as they usually consist of thousands and thousands of parameters, requiring a lot of computational resources and much time for training [16]. Moreover, their practical performance will be highly impacted by their architectures and the settings of hyperparameters [17]. In most references [18-20], the network topological architecture and its hyperparameters are manually set by using a trial-and-error method, which is a tedious and time-consuming task often requiring the participation of specialists in this field.

Recently, the Neuroevolution technique [21, 22] is suggested to automatically design a neural network, which can find suitable network topologies and weights for neural networks with evolutionary algorithm (EA) for solving various application problems. Some promising methods have been developed in the field of Neuroevolution, such as neuroevolution of augmenting topologies (NEAT) [23], evolutionary programming Net (EPNet) [24], and evolutionary algorithm that constructs recurrent neural networks (GNARL) [25]. However, they can only be used to optimize simple neural networks with few neuron units and hidden

networks. In [26], a Neuroevolution model is proposed to encode and evolve some parameters of the networks, but the network topologies in convolutional neural network (CNN) and recurrent neural network (RNN) are fixed, which are not encoded. In [17], different modules of CNN are first constructed and encoded. Then, EA is used to optimize the order of their combination to find a competitive CNN network. Also, in deep neuroevolution of augmenting topologies (DeepNEAT) [27], both the topological architecture and hyperparameters of the DNN are encoded and then optimized by EA, which can automatically construct the DNN model with satisfactory performance.

Moreover, the detection latency is also a challenge for intrusion detection in IoT. Most of the traditional IDSs are deployed on the cloud servers. Thus, the traffic packets from millions of IoT devices must be transmitted to the cloud servers for detecting anomalies via the network, resulting in high detection latency and network load, and consuming more battery of devices [4]. The recently proposed Fog computing [28] solves this problem well. The fog layer is deployed between the cloud layer and the edge layer, which is physically closer to the edge devices, and the number of fog layer nodes is much larger than the cloud layer nodes. The fog layer can help to reduce the workload of the cloud and network. More importantly, this layer can provide low-latency services for devices. Thus, the detection latency of fog-based IDS is significantly lower than that of cloud-based IDS, which can secure the IoT networks in a more timely manner.

To the best of our knowledge, most existing IoT-IDSs use the DL methods that have to manually set their parameters to find the suitable architectures gradually, requiring a huge training time for trial-and-error. Moreover, this manual method highly depends on the experiences of experts in the specific fields, which also cannot guarantee that their setup parameters are optimal. To alleviate the above problem, this paper proposes a multi-objective evolutionary convolutional neural network for intrusion detection system, called MECNN, which is used to detect network intrusions in IoT. A number of IDSs with low latencies and low-resource costs are suggested for IoT, which use CNNs as the classifier and can be deployed on the fog nodes of the Fog computing environment. To overcome the disadvantages of traditional DL methods that have to tune many parameters manually, this paper extends the idea of DeepNEAT [27] from DNN networks to CNN networks. The topological architecture of the CNN is transformed into a chromosome of multi-objective evolutionary algorithm (MOEA). Then, one competitive MOEA (MOEA/D [29]) is modified to optimize the detection performance (measured by classification error rate) and the model complexities (measured by the number of parameter) of CNN networks, simultaneously. In this way, our method can obtain a set of IoT-IDSs with various detection accuracies and model complexities through evolution, which can be deployed in different fog nodes according to their hardware configurations.

In summary, the main contributions of this paper are given as follows.

1) A novel multi-objective evolutionary convolutional neural network for IDS, called MECNN, is proposed for IoT under Fog computing, which evolves CNN by using MOEA. This approach can find suitable CNN models deployed on fog nodes of Fog computing to provide low-latency and

high-accuracy intrusion detection for IoT.

2) The model complexity of CNN (i.e., the number of parameters of CNN) and the detection performance (i.e., the classification error rate) are considered as two objectives of MOEA to be optimized. Then, MOEA/D [29] is modified to evolve the two objectives with a novel encoding scheme, which can finally obtain a set of IDSs with various detection accuracies and model complexities.
3) An accuracy emphasis (AE) mechanism is introduced to modify the population update method of MOEA/D, which can be more able to select new solutions with higher accuracy for the next evolution.

In addition, a series of extensive experiments are conducted on two popular datasets (AWID [30] and CIC-IDS2017 [31]) and the experimental results validate the advantages of the proposed MECNN in providing suitable IDSs with high detection accuracies and low model complexities for fog nodes in IoT when compared to other state-of-the-art methods.

The remainder of this paper is organized as follows. Section 2 gives an overview of the technologies used in IDSs. Section 3 describes the details of the proposed MECNN for intrusion detection in IoT networks. Section 4 presents the experimental configuration, while Section 5 provides the experimental results and discussions. Finally, conclusions and future work are presented in Section 6.

2. Background and Related Work

This paper aims to build a robust IoT-IDS with low computation costs for detecting attack activities to secure IoT networks. Some background and previous studies related to this paper are introduced below.

IoT plays a significant role in our daily life, but if the IoT networks do not provide sufficient security protection, they may suffer from a series of network attacks and lead to data leakage, property loss, and hardware damage, causing system downtime and even casualties [3]. Imagine the following scenarios: when you go out, burglars crack the smart lock of your home through the network to enter and steal things; when you are driving at high speed on the road, hackers remotely hack your car through the network to make you lose control of the steering wheel; when the urban traffic light system is maliciously tampered, the entire urban traffic condition will fall into chaos. These behaviors threaten our lives and property security. Therefore, it is essential to develop and deploy reliable IDSs for IoT. There have been a lot of research studies in the field of intrusion detection.

Many researchers employed ensemble techniques [32] in order to enhance the performance of IDS. In [33], an AdaBoost ensemble intrusion detection method is proposed to combine three ML techniques (DT, NB, and DNN). Then, a protocols-based method is used to analyze the properties of the attacks, which can generate new statistical flow features. This method can mitigate malicious events, particularly some botnet attacks against the domain name server (DNS), hyper text transfer protocol (HTTP), and message queuing telemetry transport (MQTT) protocols in IoT networks. In [34], a novel ensemble classifier, called RFAODE,

is suggested for IDS, which is built by using random forest (RF), average one-dependence estimator (AODE), and NB. The RF is introduced to improve the detection accuracy, while AODE is used to resolve the attribute dependency of NB. In [12], a fog-based unsupervised intrusion detection method, called FID-GAN, is proposed for cyber-physical systems using GANs, which can detect unknown attacks and conquer the challenge of acquiring labels. This FID-GAN can be placed in fog architecture with the computing resources closer to the edge devices, which can provide a low-latency detection service. In [6], a stacked de-noising AE (SDAE) and SVM are combined to propose a hierarchical intrusion detection model, called SDAE-SVM. This model uses SDAE to reduce the dimensions of traffic samples and then uses SVM for classification. However, the authors do not provide any detail on how the IDSs can be deployed on the IoT network. In [35], a lightweight IDS using supervised SVM is also suggested for IoT to detect network attacks. The authors point out that the use of a single attribute (packet arrival rate) is more efficient in detecting DDoS attacks than that of complex attributes. In [36], a novel representation learning method is proposed to describe the unknown attacks better. Then, three novel AE-based IDS models are developed to learn the latent representation from the raw data, which significantly improves the performance of the supervised learning methods for detecting unknown IoT attacks. In [13], a forensics-based DL model, called Deep-IFS, is suggested to detect intrude behaviors in industrial IoT traffic, which is composed of local gated recurrent (GRU) layer, multi-head attention (MHA) layer, and full-connected layer. Here, GRU is used to obtain local dependencies, while MHA is used to learn global representation. In [18], a bidirectional Long-Short-Term Memory (BiDLSTM) based IDS is proposed to reduce the high false alarm rate and improve the classification effect in the case of multi-class classification scenario.

In the above approaches with ML or DL techniques to build IDS, some research studies [37, 38] point out that it is equally important to perform feature selection on traffic data, which attempts to use MOEAs to improve the performance of ML for establishing more reliable IDSs. In [39], the traditional IDS is improved as smart, evolutionary, and multi-objective IoT-IDS. A modified multi-objective particle swarm optimization, called MOPSO-Lévy, is used for feature selection on the dataset and an ML method, i.e., K-nearest neighbor (KNN), is adopted to classify the traffic data. In [40], an IDS method called I-NSGA-III is designed, which uses an improved non-dominant sorting genetic algorithm-III (I-NSGA-III) for feature selection and a neural network growing hierarchical self-organizing map with probabilistic relabeling (GHSOM-pr) [41] for classification. In this approach, a novel niche preservation procedure is introduced, which consists of a bias-selection process to select the individual with the fewer selected features and higher objective values. In [42], an IDS using a modified multi-objective immune algorithm is suggested to reduce the feature dimension, which aims to find out the optimal feature subset. In this approach, due to the use of the immune algorithm, the convergence speed can be accelerated, and the reference vectors are used to guarantee the diversity in high-dimensional objective space.

To the best of our knowledge, the above existing IDSs usually use MOEAs for feature selection on

datasets, but rarely use MOEAs for optimizing their DL or ML models. To fill this research gap, this paper proposes a multi-objective evolutionary convolutional neural network to build IDSs for IoT. However, the manual setting of the parameters of CNNs is troublesome, which is not always optimal. As motivated by the idea of DeepNEAT, the topological architectures and hypermeters of CNN are transformed into the chromosome of MOEAs. Then, the model complexity and detection performance of CNN are regarded as the two objectives to be optimized by MOEAs. These two objectives conflict with each other and can be modeled as a multi-objective optimization problem. An increased model complexity with more parameters used in CNN usually leads to an increased detection accuracy, which also indicates that more computing resources and memory should be required for CNN. A simple model complexity with a small number of parameters will also result in a reduction in detection accuracy. In this paper, a competitive MOEA (MOEA/D [29]) is modified to optimize these two objectives simultaneously. Finally, a set of IDSs for IoT with different detection accuracies and model complexities can be attained and deployed on different nodes of Fog computing.

3. The Proposed Algorithm

In this section, the structure of CNN and how it works is first briefly described. Moreover, the encoding scheme used in the proposed method is described. Next, the general framework of MECNN is described. Finally, the Fog computing architecture is presented to show how the proposed method can be deployed on Fog computing and how it responds to intrusion behaviors.

3.1. Convolutional Neural Network

This subsection aims to introduce some key concepts of the CNN models to better understand the motivations of this paper and how the CNN models are encoded. CNNs are first proposed by LeCun *et al.* in [43, 44], which are now the most widely used algorithm in the field of DL. The main idea of CNN is to combine a feature extraction module and a classifier module. CNNs are good at classifying different types of documents or information, such as text, speech, or images.

The CNN model includes five types of layers: input layer, convolutional layers, pooling layers, full-connected layers, and output layer [45]. When these layers are stacked in a specific order, a CNN model is built. A typical CNN model is shown in Fig. 1. Some of these layers are described in detail.

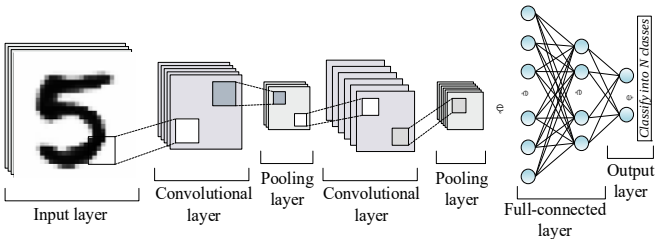


Fig. 1 A typical structure of the CNN model.

• Convolutional layer: a CNN model usually includes one or more convolutional layers. Convolution is

a special linear operation for feature extraction. Each convolutional layer contains an array of numbers called the convolutional kernels, also known as the filters. The kernel size can be set freely, which is defined as a parameter of the CNN network architecture. The kernel convolves the input data, called the tensor, in a certain order and then outputs data called the feature map. The feature map will be used as the input for the following layer. This process will continue until there are no more convolutional layers. Simply, the convolutional layers are responsible for extracting significant features from the raw data and using the features as input to the next layer.

• Pooling layer: the pooling layers perform the down-sampling operation to reduce the dimensions of the input data, which replace features in a region with some calculated features by specific methods, such as average pooling, max pooling, and overlapping pooling. Max pooling is the most common method, which divides the input data into several rectangular areas and uses the maximum value in the area to represent this area. The pooling operations are usually employed after the convolution operation.

• Full-connected layer: the full-connected layer, occasionally called dense layer, generally includes several layers, each of which has numerous neurons. Each neuron is connected to all the neurons of the previous layer, and they will receive input from neurons of the last layer by means of the specified weights. The model adjusts the weight between neurons by each epoch to continuously improve the classification effect of the model. The parameters associated with the full-connected layer include the number of full-connected layers, the number of neurons in a full-connected layer, and the activation function.

In addition to the parameters mentioned above directly related to the network topological architecture, there are some other parameters, often called hyperparameters, which are related to the training process of CNNs:

• Learning rule: also called optimizer, is a mathematical logic method repeatedly applied in the network, which can improve the neural network performance. The rule updates the weight and bias levels of the network when handling a task.

• Leaning rate: the learning rule determines the method of updating the weights and bias levels, and the learning rate determines how many weights are updated in each epoch.

• Batch size: batch size is the number of samples selected in an iteration, which affects the degree of optimization and the training speed of the model.

• Dropout rate: dropout rate is a recently proposed method [46], which is presented to prevent overfitting problems by randomly dropping some neurons during the training process.

• Activation function: the activation function performs a nonlinear transformation to the input, making the neural network enable learning and performing more complex tasks.

3.2. Encoding Scheme of MECNN

As described above, the design of a CNN model is complicated and involves plenty of parameters.

However, the search for optimal topological architectures and hyperparameters of CNN is still a challenging task, as it requires professional knowledge and practical experience. For example, theoretically, deepening the network depths of the full-connected layer and enlarging the number of neurons will generally improve the performance of the model. However, when the number of depths in CNN reaches a certain level, the performance of the model will not be improved significantly and even be decreased. Especially with the continual increase of the depths, this may lead to over-fitting, resulting in vanishing gradient, increasing more training time, and occupying more computing resources. This high computational cost makes the model unsuitable for deployment on the fog nodes that typically have limited resources. When the numbers of layers and neurons are too few, the classification effect of the model may be poor, which may not effectively detect the network intrusion activities.

Since it is complex to build an appropriate CNN model, a competitive MOEA (MOEA/D [29]) is modified to optimize and build the CNN model automatically. These related parameters are transformed into a chromosome of MOEAs, which are then optimized by MOEA/D. For convenience, in our encoding scheme, these parameters are divided into three groups according to their different roles: convolution group, full-connected group, and hyperparameters group.

The convolution group includes the number of convolutional layers, the number of kernels, the kernel size, whether to perform pool operations (indicated by 0 or 1), and the pooling size.

The full-connected group includes the number of full-connected layers, the number of neurons of each full-connected layer, and the dropout rate.

The hypermeters group includes the batch size, the learning rule, and the learning rate.

For the activation function, it is not considered as an optimization objective and the ReLu activation is used in the convolutional layers and the intermediate of the full-connected layer. For the output layer of the full-connected layer, the Sigmoid activation function is used for the binary-class scenario, while the Softmax activation function is used for the multi-class scenario.

In addition, real numbers are used in the model encoding of our method and a real-number sequence with 13 genes is used as a chromosome to represent a CNN model. The first 11 numbers represent a CNN entity model, and the last two numbers indicate the two objective values of the model. The definition of the chromosome is shown in Fig. 2. The range of available values for each gene and the meaning of the symbols used in this paper are expressed as follows:

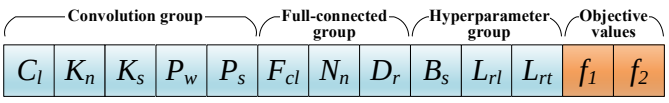


Fig. 2 Definition of the chromosome of MECNN.

• C_l : the number of convolutional layers and $C_l \in [1,2,3]$.

• K_n : the number of kernels in each convolutional layer and $K_n \in [2,4,8,16,32,64,128,256]$.

• Ks : the number of kernel sizes in each convolutional layer and $K_s \in [2,3,4,5,6,7,8,9]$.

• P_w: whether to perform pooling operation and $P_w \in [0,1]$ (0 indicates not to perform pooling operation while 1 is yes).

• P_s : the pooling size and $P_s \in [2,3,4,5,6,7,8,9]$.

• F_{cl} : the number of full-connected layers and $F_{cl} \in [1,2,3]$.

• N_n : the number of neurons in each full-connected layer and $N_n \in [8,16,32,64,128,256,512,1024]$.

• D_r : the dropout rate and $D_r \in [0,0.25,0.5]$.

• B_s : the batch size and $B_s \in [32,64,128,256]$.

• L_{rl} : the learning rule and three learning rules used in our method, i.e., $L_{rl} \in [0,1,2]$, where the values of 0, 1 and 2, indicate to use Adam, stochastic gradient descent (SGD), and Adamax, respectively.

• L_{rt} : the learning rate and $L_{rt} \in [0.5,0.1,0.05,0.01,0.001,0.0001]$.

• f_1 : the first objective to be optimized, which is the model complexity of CNN as measured by the number of parameters in the CNN model.

• f_2 : the second objective to be optimized, which is the detection performance of CNN as measured by the detection error rate of CNN.

The summary of the ranges of these parameters is given in Table 1.

Table 1
Range of parameters of CNN

Parameters	**Range**
Number of convolutional layers, C_l	1,2,3
Number of kernels, K_n	2,4,8,16,32,64,128,256
Kernel size, Ks	2,3,4,5,6,7,8,9
Weather to pooling, P_w	0,1
Pooling size, P_s	2,3,4,5,6,7,8,9
Dropout rate, D_r	0,0.25,0.5
Number of full-connected layer, F_{cl}	1,2,3
Number of neurons, N_n	8,16,32,64,128,256,512,1024
Batch size, B_s	32,64,128,256
Learning rule, L_{rl}	Adam, SGD, Adamax
Learning rate, L_{rt}	0.5,0.1,0.05,0.01,0.001,0.0001

3.3. General Framework of MECNN

As explained above, the first objective f_1 is the model complexity as measured by the number of parameters of CNN and the second objective f_2 is the detection performance as measured by the detection error rate of CNN. Those two objectives are generally conflicting with each other. A decrease in f_1 generally leads to an increase in f_2, while a reduction in f_2 may lead to an increase in f_1. In this paper, the state-of-the-art MOEA is modified to optimize these two objectives simultaneously. For the decomposition approach of MOEA/D, a popular decomposition method, i.e., Tchebycheff [47], is used, as defined by

$$g^{te}(x \mid \lambda, z^*) = \max_{1 \le i \le m} \{\lambda_j \mid f_i(x) - Z^*\}, \quad (1)$$

where m is the number of objectives, $\lambda = (\lambda_1, ...\lambda_m)^T$ is an m-dimensional weight vector with $\sum_{i=1}^{m} \lambda_i = 1 (\lambda_i \ge 0)$, and $Z^* = \{z_1^*, ..., z_m^*\}$ is an ideal point with $Z^* = \{\min f_i(x) \mid x \in \Omega\}$ for $i = 1, ..., m$.

The pseudo-code of MECNN is given in **Algorithm 1**. In line 1, the neighborhood index $B(i)$ of T closest subproblems is initialized. In line 2, the initial population P with N individuals is generated. In line 3, for each individual in P, a CNN model is built according to its genotype, the training set is used to train the model, and then the testing set is adopted to evaluate its two objective values. In line 4, the ideal point Z^* is initialized by using $Z^* = \{\min f_i(x) \mid x \in \Omega\}$. Next, the algorithm enters the main evolutionary loop. In line 7, two indexes k and l are randomly selected from $B(j)$, and two individuals x^k and x^l are then selected using the two indexes from the population P. Then, crossover and mutation operators are run on x^k and x^l to generate two new individuals y_1 and y_2. Here, the two-point crossover method [48] is used for crossover as shown in Fig. 3 (a), while the bit-wise mutation method is used for mutation as shown in Fig. 3 (b). Please note that the mutation in our algorithm will be activated for each gene of the individual by a predefined mutation probability, which is randomly selected from the preset range as shown in Tabel 1 for the new parameter. In line 8, for each newly generated individual y_1 or y_2, the CNN model is built according to the genotype, which is to be trained by using the training set and evaluated by using the testing set to get the objective values. Then, their Tchebycheff values $g^{te}(y_1 \mid \lambda^j, z)$ and $g^{te}(y_2 \mid \lambda^j, z)$ are calculated, and the individual with a small Tchebycheff value is selected as the new solution y. In line 9, the ideal point is updated.

Algorithm 1: MECNN

Inputs:

N : number of Population
T : neighborhood size
$G_{\max}$: maximum generation
A uniform spread of N weight vectors: $\lambda^1, ..., \lambda^N$
Training sets and testing sets

Output:

The final population P

1. Identify the neighborhood $B(i)$ of T closest subproblems
2. Generate an initial population $P = \{x^1, ..., x^N\}$
3. For each individual in P, build a CNN model and evaluate objective values
4. Initialize the ideal point $Z^* = (z_1^*, ..., z_m^*)$
5. **for** i =1 to $G_{\max}$,do
6. **for** j=1 to N , do
7. **Reproduction:** randomly select two indexes k, l from $B(j)$, and then generate a new solution y_1, y_2 from x^k and x^l by Crossover and Mutation
8. **Evaluation:** build CNN model by y_1, y_2 and evaluate their objective values
9. **Update of** Z^*: for each $k = 1, ..., m$, if $z_k > f_k(y)$, set $Z_k = f_k(y)$
10. **Update of Neighboring solutions**
11. **end for**
12. **end for**
13. **Return**

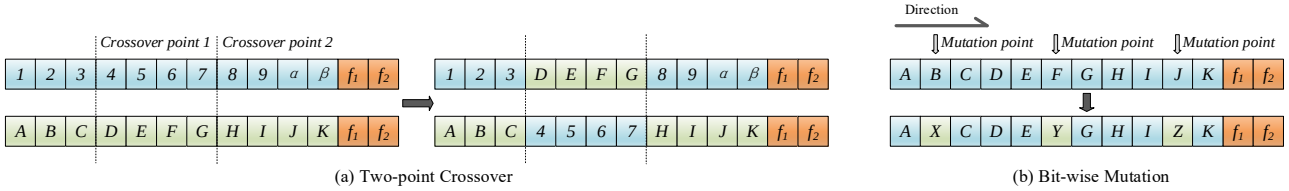


Fig. 3. Reproduction: (a) Crossover and (b) Mutation

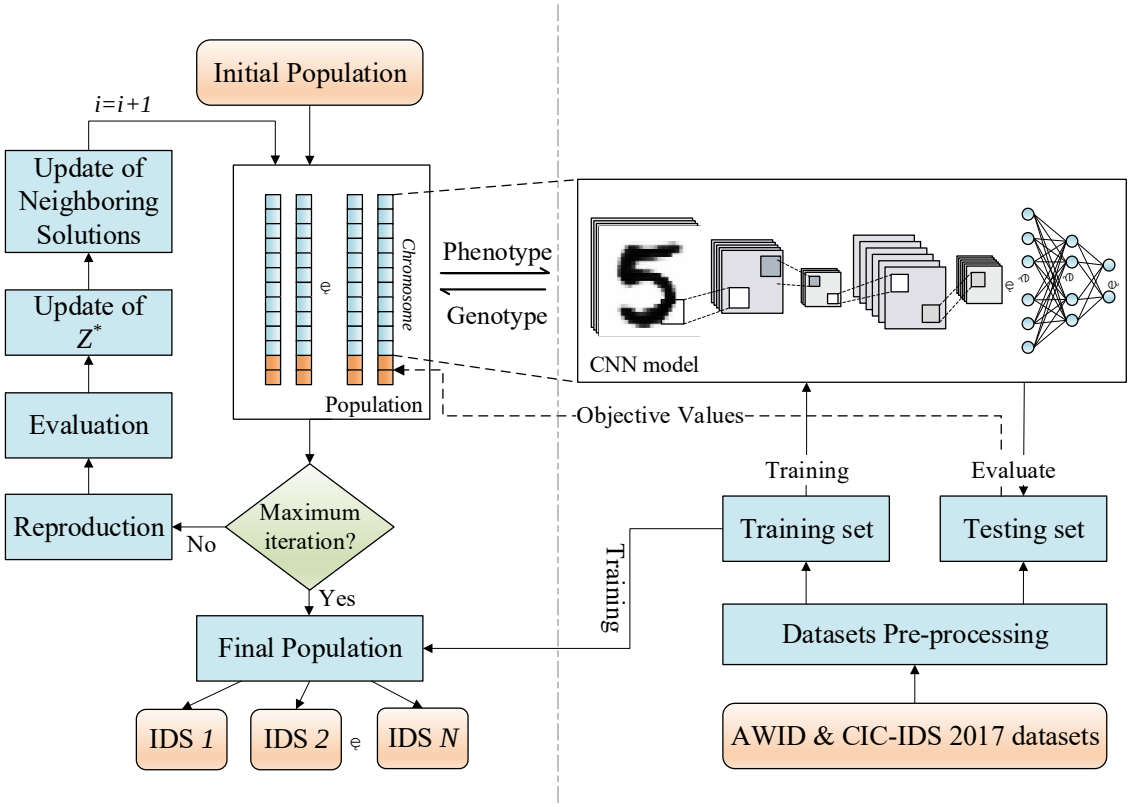
Finally, in line 10, the update of neighboring solutions is run, which is introduced in detail in **Algorithm 2**. Here, an accuracy emphasis (AE) mechanism is proposed in this process, which lets a new solution with a low detection error rate have more opportunities to update the population without sacrificing diversity [49]. In detail, the Tchebycheff values $g^{te}(y|\lambda^j,z)$ of y and $g^{te}(x^j|\lambda^j,z)$ of x^j are calculated in line 2 of **Algorithm 2**. Next, their Tchebycheff values are compared. If $g^{te}(y|\lambda^j,z)$ is smaller than $g^{te}(x^j|\lambda^j,z)$, their detection error rates are compared. Then if the detection error rate of y is still smaller than that of x^j, x^j is updated and replaced by y in line 5. Otherwise, if the detection error rate of y is greater than that of x^j, a random number is generated in [0, 1] to compare with the AE rate σ. If this random number is greater than σ, x^j is updated and replaced by y. Otherwise, the algorithm executes the next loop. The flowchart of MECNN is illustrated in Fig. 4, with the evolutionary process on the left side and the training and evaluation process on the right side.

Algorithm 2: Update of Neighboring solutions

Input:

σ: accuracy emphasis rate

1. **for** each index j in $B(i)$, **do**
2. Compute the Tchebycheff value $g^{te}(y|\lambda^j,z)$ and $g^{te}(x^j|\lambda^j,z)$
3. **if** $g^{te}(y|\lambda^j,z) \le g^{te}(x^j|\lambda^j,z)$ **then**
4. **if** $f_2(j) \le y_2$ **then**
5. set $x^j = y$
6. **else if** $\sigma \le rand(0,1)$ **then**
7. set $x^j = y$
8. **end if**
9. **end if**
10. **end for**



(a) Evolutionary process. (b) Training and evaluation process.

Fig. 4 The flowchart of MECNN.

3.4. Fog Computing Architecture and IoT-IDS Deployment

This subsection first briefly describes the composition of the Fog computing and describes how our MECNN is trained and how it is deployed to provide intrusion detection for IoT. Fog computing is a paradigm developed by Cisco that transfers the data and services to the edge of the network in the cloud, which is seen as an extension of the concept of Cloud computing paradigm [28]. It is a highly virtual platform that provides compute, storage, and network services between end devices and traditional cloud servers. Fog computing has several unique features that distinguish it from other computing architectures, the most important of which is its closer distance to end-users, thus leading to lower latency service [50]. As shown in Fig. 5, the Fog computing-based IoT architecture consists of three layers: cloud layer, fog layer, and edge layer.

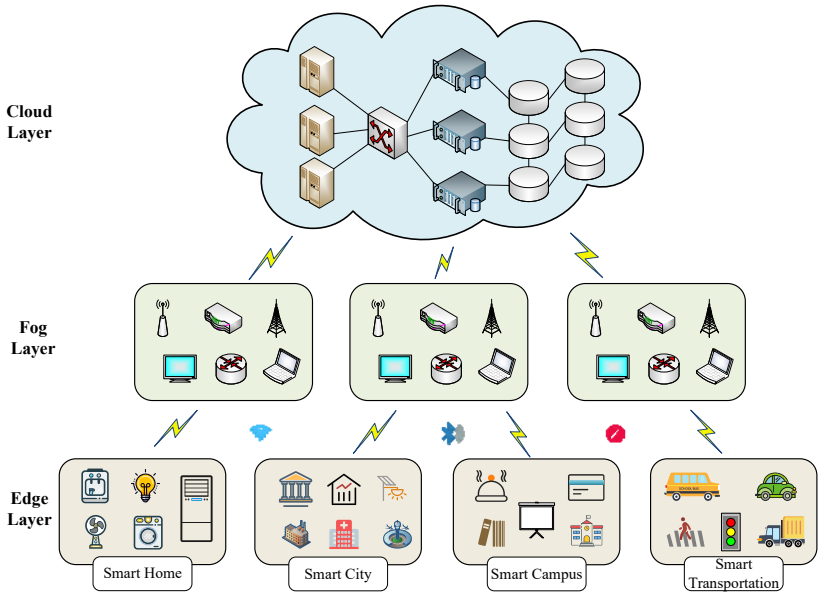


Fig.5 Fog computing-based IoT architecture

The cloud layer usually consists of high-performance servers, high-capacity storage platforms, and some network communication components, which have powerful computing resources. The fog layer is not made up of powerful servers but of less powerful and more distributed computers with various functions. Devices in this layer put more emphasis on quantity than on performance, extending computing resources from the cloud server to the edge of the network, and closer to the edge devices, so that the fog layer can provide low-latency-aware services for the edge devices. Finally, a variety of edge devices and edge nodes (i.e., smartphones, laptops, smart home devices, smart wearable devices, etc.) are connected to the corresponding fog nodes, forming the edge layer, bringing various conveniences to our lives.

If MECNN is deployed on the cloud node as other traditional methods to detect network anomalies, the network transmission load and bandwidth requirements will increase dramatically [12], which cannot meet the demand of low-latency. Therefore, in order to take full advantage of the features of the Fog computing platform, like other existing methods in [12] [51] and [52], the training process of MECNN that requires a lot of computing resources and does not have real-time requirements is conducted on the cloud nodes which are usually powerful and resourceful. Then, the trained MECNN models are deployed as virtual machines on fog nodes to monitor network traffic, which provide low-latency intrusion detection services for the Fog computing platform, as fog nodes are inherently closer to edge devices than cloud nodes.

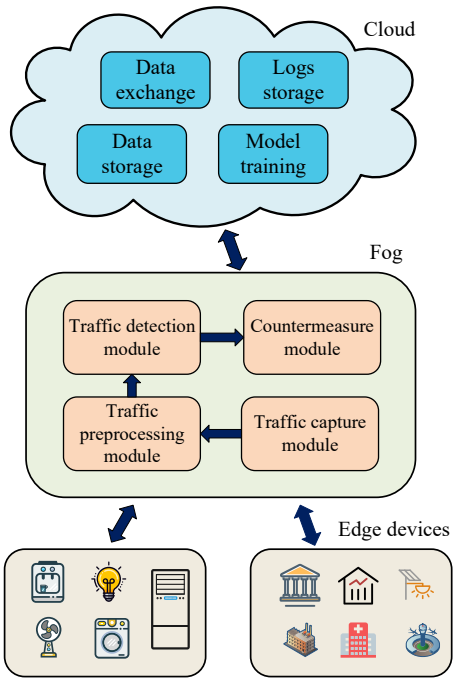


Fig. 6 Flowchart of MECNN for network intrusion detection

Fig.6 shows the process of how MECNN detects network intrusion in Fog computing. It can be seen that each fog node consists of four main components, i.e., traffic capture module, traffic preprocessing module, traffic detection module, and countermeasure model. Each node monitors the relevant area under its responsibility. The traffic capture module is responsible for capturing and receiving network traffic from the edge devices and then transforming them into batch samples for the next module. Next, the traffic preprocessing module is accountable for converting traffic into the standard format. Then the traffic detection module uses the MECNN model deployed on it to classify the traffic records, and it is worth noting that this process does not need to interact with the cloud, thus avoiding any latency. Finally, the countermeasure module is used to perform countermeasure actions (such as blocking, removing and alerting). Also, when the classified result is anomalous, this module will send the processing result to the cloud to store in logs storage.

4. Experimental Configuration

In this section, the information of the platform and environment used in the experiments are first briefly introduced. Next, the datasets used in the experiments and how to pre-process them are presented. Finally, some performance metrics are introduced to evaluate the performance of MECNN.

4.1. Experimental Setup

The overall experiments of this study were performed on a server with an Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.2GHz processor, RAM of 256 GB size, running 64-bit Ubuntu 16.04 operating system, and accelerated with Nvidia GeForce GTX 1080Ti with 11GB memory graphics card. The implementation of MECNN was carried out using Python 3.8.5 programing language, Keras Library [53], and TensorFlow API [54]. The fog node is a personal computer with an Intel(R) Core(TM) i5-4590 CPU @ 3.3GHz processor and

RAM of 4 GB size, running 64-bit Windows 10 operating system. Other parameters of MECNN are shown in Table 2.

Table 2
Parameters setting of MECNN

Parameter types	Value
Number of population N	30
Neighborhood size T	T=N/10
Probability of crossover	1
Probability of mutation	0.4
Number of generations	20
Epoch of the evolutionary process	8
Epoch of the final population	100
Accuracy emphasis rate	0.8

4.2. Datasets Description

Table 3
Summary of AWID and CIC-IDS2017 datasets characteristics

Dataset	No. of samples	No. of dimensions	No. of classes	Types of Attacks	Data Distribution			
				-	Classes	Train	Val	Test
AWID [30]	AWID-CLS-R-Trn: 1,339,406 AWID-CLS-R-Tst: 389,185	Category: 1 Feature:74	4 classes	Normal	Normal	941,617	235,404	346,285
				Flooding	Flooding	38,788	9,696	6,139
				Injection	Injection	52,304	13,075	16,682
				Impersonation	Impersonation	38,818	9,704	20,079
CIC-IDS2017 [31]	Original:2,830,743 Removed:2,867 Cleaned:2,827,876	Category: 1 Feature:77	15 classes, mapped to 7 classes	Benign	Normal	1,362,866	454,227	454,227
				Bot	Botnet	1133	412	411
				FTP-Patator, SSH-Patator	Brute Force	8,363	2,734	2,735
				DDoS, DoS GoldenEye, DoS Hulk, DoS Slow httptest, DoS slowloris, Heartbleed	Dos/DDos	227,472	76138	76,138
				Infiltration	Infiltration	25	6	5
				PortScan	PortScan	95,535	31,635	31,634
				Web Attack – Brute Force, Web Attack – Sql Injection, Web Attack – XSS	Web Attack	1,331	424	425

There are various public intrusion detection datasets that can be obtained on the Internet. Two of the most popular datasets are chosen to evaluate the performance of our MECNN, which are AWID [30] and CIC-IDS2017 [31].

(1) AWID dataset

Aegean WIFI Intrusion Dataset (AWID) is a freely accessible dataset that relies on IEEE 802.11 networks. The developer collected WIFI traffic records in a packet-based format to develop this dataset, representing the real-world wireless network which is a critical feature for modern IoT environments. The AWID dataset comprises two equal sets, i.e., AWID-CLS and AWID-ATK, and their difference is only the labeling method. The former is labeled as four classes: Normal, Flooding, Impersonation, and Injection, and the latter is labeled with more details, which has sixteen classes. Each of the two sets consists of a full subset (AWID-CLS-F and AWID-ATK-F) and a reduced one (AWID-CLS-R and AWID-ATK-R). The reduced subsets are more suitable for the experiments because of their smaller classes. Thus, in this paper, the reduced set (AWID-CLS-R-Trn and AWID-CLS-R-Tst) are used for experiments. There are 1,339,406 traffic records in AWID-CLS-R-Trn and 389,185 traffic records in AWID-CLS-R-Tst.

(2) CIC-IDS2017 dataset

The Canadian Institute for Cybersecurity intrusion detection system dataset 2017 (CIC-IDS2017) has 2,830,743 traffic records and contains benign and the most up-to-date common attacks, which resembles the true real-world IoT network traffic. According to the author [31] of CIC-IDS2017, the dataset spanned over eight different files stored in comma-separated value (CSV) files containing five days of normal and attack traffic records of the Canadian Institute of Cybersecurity. This dataset comprises 78 dimensions of features and one class feature, including benign class and other fourteen classes of attacks.

Compared with the datasets generated by simulation software, those two datasets, AWID and CIC-IDS2017, have higher reliability and authenticity, thus they can reflect the real IoT network traffic to a certain extent. Moreover, those datasets are also adopted by other state-of-the-art IoT-IDSs to evaluate the performance of the model. Therefore, these two datasets are well suitable for evaluating the performance of the proposed MECNN. The details of those two datasets and their corresponding data distribution are shown in Table 3.

4.3. Dataset Pre-processing

Next, the pre-processing operations are conducted on those two datasets. Firstly, there is some illegal information in CIC-IDS2017, such as null and non-numeric characters, which may influence the training process, so they should be removed. As shown in Table 3, a total of 2,867 illegal records were eliminated. Secondly, there is a total of fifteen types of classes in CIC-IDS2017, but some of them are remarkably similar. Inspired by [55], similar classes are combined into one class in this paper, as shown in Table 3. The one-hot encoding operation was conducted on these two datasets to convert the categories into numeric representations to facilitate neural network recognition. Moreover, each dataset was divided into three groups (training set, testing set, and validation set). The AWID dataset has already been divided into the training set (AWID-CLS-R-Trn) and the testing set (AWID-CLS-R-Tst), but there has no validation set, so AWID-CLS-R-Trn is further divided in experiments (80% of data are divided into the training set and 20% of data are divided into the validation set), and AWID-CLS-R-Tst is still regarded as testing set. For the

CIC-IDS2017 dataset, 60% of data are regarded as the training set, 20% of data are used for the testing set, and the rest 20% are used as the validation set.

4.4. Evaluation Metrics

To evaluate the performance of the proposed method, evaluation metrics, such as *False Negative* (*FN*), *False Positive* (*FP*), *True Positive* (*TP*), and *True Negative* (*TN*), are used in performance evaluation. Those metrics are computed through the confusion matrix. Furthermore, four extra evaluation measures, such as *Accuracy*, *Precision*, *Recall*, and *F1-measure*, which can be obtained according to Eqs. (2), (3), (4), and (5), respectively, are also used to evaluate the performance of the compared approaches in this paper.

$$Accuracy(ACC) = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

$$Precision(PRC) = \frac{TP}{TP + FP} \tag{3}$$

$$Recall(RCL) = \frac{TP}{TP + FN} \tag{4}$$

$$F1-measure(F1) = 2 \frac{Precision \cdot Recall}{Precision + Recall} \tag{5}$$

5. Experimental Results and Discussions

5.1. Experimental Results of MECNN

In this subsection, the changes of objective values during the evolution of individuals (network models) in MECNN are first described, which can show how our model can dynamically evolve the CNN with promising performance. Then, the best-performing individuals obtained by MECNN on the AWID and CIC-IDS2017datasets are introduced and discussed.

5.1.1. Population Analysis

This subsection analyzes the changes of individuals in the population before and after the evolution of MECNN on the AWID and CIC-IDS2017 datasets.

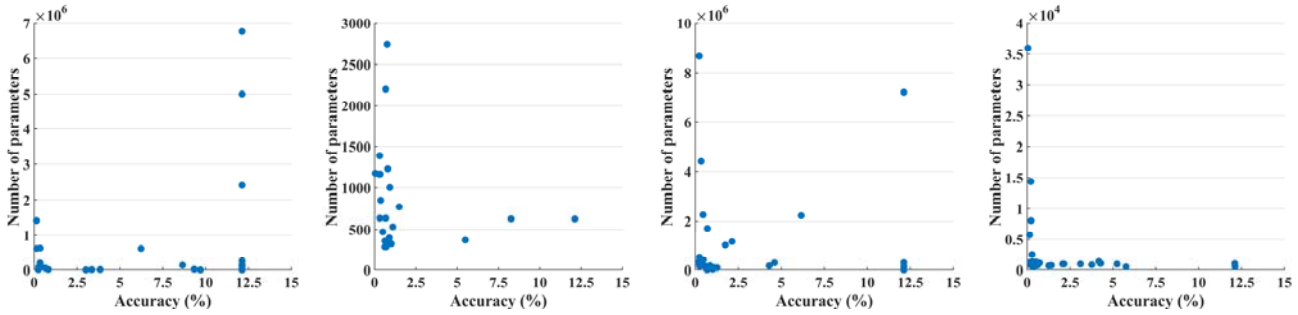


Fig. 7 The distributions of all the individuals in the initial and final generations on the AWID dataset.

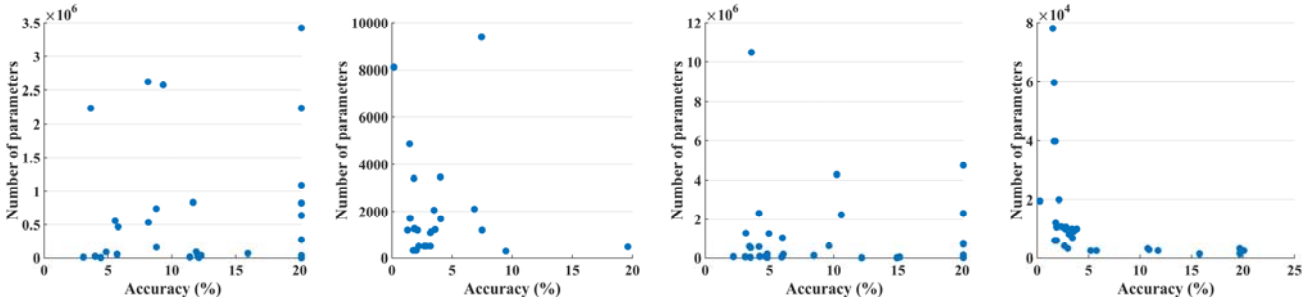


Fig. 8 The distributions of all the individuals in the initial and final generations on the CIC-IDS2017 dataset.

Table 4
The extreme value of each objective obtained in the initial and the final generations on the AWID dataset

		Initial generation		Final generation	
	Individual	Parameters	Error rate (%)	Parameters	Error rate (%)
Two-class	Least parameters	636	12.13	284	0.62
	Lowest error rate	1,415,176	0.14	1,178	0.04
	Average	619,080	7.40	771	1.47
Four-class	Least parameters	3,534	12.13	454	0.38
	Lowest error rate	297,038	0.18	35,968	0.04
	Average	1,031,585	5.19	2,938	2.17

Table 5
The extreme value of each objective obtained in the initial and the final generations on the CIC-IDS2017 dataset

		Initial generation		Final generation	
	Individual	parameters	Error rate (%)	parameters	Error rate (%)
Two-class	Least parameters	1,100	20.14	460	20.14
	Lowest error rate	9,574	3.07	8,118	0.16
	Average	634,292	12.55	2,013	4.92
Seven-class	Least parameters	1,045	14.92	1,317	11.58
	Lowest error rate	61,119	2.19	19,523	0.27
	Average	1,119,336	9.49	13,353	5.56

For the AWID dataset, Figs. 7(a) and 7(b) show the distributions of individuals in the initial and final generations on the binary-class scenario. Fig. 7(c) and 7(d) show the distributions of individuals in the initial and final generations on the four-class scenario. In Figs. 7(a) and 7(c), it can be observed that the distributions of individuals are irregular, and most individuals have high model complexity and high detection error rate, which is mainly because the individuals in the initial population are generated randomly. Since optimizing the CNN model is a practical problem, the real Pareto-optimal curve cannot be obtained in advance. However, in Figs. 7(b) and 7(d), after running 20 generations, it is apparent that the model complexities and the detection error rates of all the individuals are reduced significantly, and the curve is basically converged. The details of the extreme value of each objective obtained in the initial and final generations and the average change of individuals are collected in Table 4. For the binary-class scenario, in the initial generation, the average number of parameters in the individuals is 619,080, and the average

detection error rate is 7.40%. Whereas in the final generation, the average value of these objectives has been reduced significantly, i.e., the average number of parameters is 771, and the average detection error rate is 1.47%. For the four-class scenario, in the initial population, the average number of parameters of the individuals is 1,031,585, and the average detection error rate is 5.19%. After evolution, in the last generation, the average number of parameters is 2,938, and the average detection error rate is 2.17%. No matter whether it is a binary-scenario or four-class scenario, after the evolution of MECNN, the number of parameters and the detection error rates in the population have been reduced significantly. That is to say, after evolution, both the two objective values in individuals become smaller, which means that the corresponding CNN models are simpler, but the detection effect becomes better.

For the CIC-IDS2017 dataset, the comparison results are similar to that for the AWID dataset, Figs. 8(a) and 8(b) show the distributions of individuals in the initial and final generations on the binary-class scenario, while Figs. 8(c) and 8(d) show their distributions on the seven-class scenario. The individuals in Figs. 8(a) and 8(c) are also distributed irregularly, which have high complexities and high detection error rates, while in Figs. 8(b) and 8(d), nearly all individuals have significantly reduced their complexities and detection error rates. The details of the extreme value of each objective in the initial and final generations, and the average change of the individuals are shown in Table 5. For the binary-class scenario, it can be observed that the average number of parameters of the individuals in the initial population is 634,292, with an average detection error rate of 12.55%. However, in the final generation, the average number of parameters and the average detection error rate drastically drop to 2,013 and 4.92%, respectively. For the seven-class scenario, the average number of parameters and the average detection error rate are 1,119,336 and 9.49% in the initial population, i.e., both the two objective values are extremely large. Whereas in the final population after evolution, those two objective values drop significantly to 13,353 and 5.56%, respectively, which means that most individuals in the final population can build the corresponding CNN models with simpler model complexities and better detection performance.

After the evolution of MOEA, the complexities and detection performance of the CNN models are improved significantly. These experiments demonstrate that the proposed MECNN can indeed simplify the design and training of CNN models that require complex manual tuning of parameters to find the suitable architecture, which can evolve to obtain simple and efficient CNN models.

5.1.2. More Detailed Experimental Results

In this subsection, the best-performing individuals in the population are selected, and their detailed results on the binary-class and multi-class scenarios are discussed.

For the AWID dataset, the confusion matrix of our MECNN on the binary-class scenario and the four-class scenario are presented in Table 6 and Table 7, respectively. For the binary-class scenario, it can be clearly observed that although the difference in the number of Attack and Normal samples is large, our method still obtains satisfactory results with Precision, Recall, and F1-measure all above 99% and near to

100%. For the four-class scenario, it could be observed that detecting the Flooding class shows the worst performance with Precision of 99.17%, Recall of 99.92%, and F1-measure of 99.54%. However, despite the relatively weak performance on this class, other results are generally desirable.

Table 6
Confusion matrix of MECNN on the AWID dataset in case of binary-class scenario

		Predicted values			
	Classes	Attack	Normal	RCL (%)	F1 (%)
Actual values	Attack	162,354	31	99.98	99.83
	Normal	528	1,176,493	99.96	99.98
	PRC (%)	99.68	100.00	-	-

Table 7
Confusion matrix of MECNN on the AWID datasets in case of four-class scenario

		Predicted values						
	Classes	Flooding	Impersonation	Injection	Normal	PRC（%）	RCL (%)	F1 (%)
Actual values	Flooding	48,443	2	0	39	99.17	99.92	99.54
	Impersonation	5	48,468	2	47	99.97	99.89	99.93
	Injection	0	0	65,379	0	99.87	100.00	99.94
	Normal	400	13	81	1,176,527	99.99	99.96	99.98

Table 8
Confusion matrix of MECNN on the CIC-IDS2017 dataset in case of binary-class scenario

		Predicted values			
	Classes	Attack	Benign	RCL (%)	F1 (%)
Actual values	Attack	111,395	431	99.62	99.59
	Benign	496	453,827	99.89	99.90
	PRC (%)	99.56	99.97	-	-

Table 9
Confusion matrix of MECNN on the CIC-IDS2017 datasets in case of seven-class scenario

		Predicted values									
	Classes	Botnet	Brute Force	Dos/DDos	Infiltration	Normal	PortScan	Web Attack	PRC (%)	RCL (%)	F1 (%)
Actual values	Botnet	151	0	0	0	260	0	0	98.05	36.74	53.45
	BruteForce	0	2,717	4	0	11	3	0	99.60	99.34	99.47
	Dos/DDos	0	0	75,968	0	167	0	3	98.91	99.78	99.34
	Infiltration	0	0	0	1	4	0	0	100.00	20.00	33.33
	Normal	3	11	814	0	453,183	213	3	99.90	99.77	99.84
	PortScan	0	0	16	0	7	31,608	3	99.32	99.92	99.62
	WebAttack	0	0	3	0	4	0	418	97.89	98.35	98.12

For the CIC-IDS2017 dataset, the confusion matrixes of the binary-class scenario and the seven-class

scenario are shown in Table 8 and Table 9, respectively. For the binary-class scenario, the overall results are promising, with incredibly high values above 99% on three metrics (Precision, Recall, F1-measure). For the seven-class scenario, it is observed that detecting the Infiltration class shows the worst performance with Precision of 100%, Recall of 20%, and F1-measure of 33.33%, followed by the Botnet class with Precision of 98.05%, Recall of 36.74%, and F1-measure of 53.45%, and followed by Web Attack class with Precision of 97.89%, Recall of 98.35% and F1-measure of 98.12%. The reason for the poor effect of these classes is that the number of samples in these classes is very small, which causes a serious data imbalance problem. Although the proposed MECNN performs relatively poorly on these classes with small sample sizes, it still shows significantly improved performance on other cases, which can distinguish different types of network traffic well and secure the IoT network.

5.2. Comparative Analysis

In this subsection, to validate the reliability of the proposed MECNN, the best-performing individual is selected to compare with some conventional ML approaches, i.e., logistic regression (LR) [56], NB [10], AdaBoost [57] and SVM [11], and some recent state-of-the-art DL approaches, i.e., GRU [19], long short-term memory (LSTM) [58], BiDLSTM [18], artificial neural network (ANN) [59], AE-ANN [59], deep belief network-ANN (DBN-ANN) [59], SAE-DNN [20] and SDAE-SVM [6]. To ensure a fair comparison, these DL approaches are implemented according to the parameters and configurations mentioned in the corresponding references. In addition, the same pre-processing and division methods are used on the datasets to run the experiments and obtain the final experimental results.

Table 10 shows the comparative results of the AWID dataset. For the binary-class scenario, NB [10] performs worst regarding all metrics. On the other hand, in all ML approaches, AdaBoost is the best with Accuracy of 99.60%, Precision of 99.06%, Recall of 99.05%, and F1-measure of 99.05%, and its results are similar to that of the state-of-the-art DLs, which seems that it is a reliable approach. Concerning the recent DL approaches, SDAE-SVM [6] obtains the worst results with Accuracy of 95.05%, Precision of 87.79%, Recall of 89.61%, and F1-measure of 88.67%, which is similar to the results of LR [56] but is far inferior to AdaBoost [57]. Except for SDAE-SVM, the results of other DL methods all exceed 99% in all metrics. It's worth noting that, the results of the BiDLSTM [18] are similar to our method with Accuracy of 99.95%, Precision of 99.99%, Recall of 99.95%, and F1-measure of 99.97%. More importantly, our MECNN achieves the best performance with Accuracy of 99.96%, Precision of 99.99%, Recall of 99.95%, and F1-measure of 99.97%. For the four-class scenario, it can be noticed that the results are similar to that in the binary-class scenario, but the performance of most models is slightly degraded, which is because the multi-class scenario is more complicated and involves more classes to process. In addition, the Recall of LSTM [58] is somewhat higher than that of our MECNN (LSTM is 99.95%, while MECNN is 99.94%). However, MECNN is better than LSTM on the other three metrics, and F1-measure has considered Recall in the calculation process. Therefore, it can be said that MECNN performs best among all methods.

Table 10
The results of comparative experiments on the AWID dataset

	Binary-class scenario				Four-class scenario			
Method	**ACC (%)**	**PRC (%)**	**RCL (%)**	**F1 (%)**	**ACC (%)**	**PRC (%)**	**RCL (%)**	**F1(%)**
Machine Learning Approaches								
LR [56]	95.02	87.70	89.52	88.58	97.48	90.63	89.45	89.98
NB [10]	87.87	74.86	92.36	79.44	85.34	72.97	93.94	76.67
AdaBoost [57]	99.60	99.06	99.05	99.05	95.46	90.40	73.54	74.31
SVM [11]	94.87	87.25	89.50	88.33	96.56	87.12	86.65	86.82
Deep Learning Approaches								
GRU [19]	99.89	99.74	99.76	99.75	99.91	99.65	99.74	99.70
LSTM [58]	99.91	99.66	99.92	99.79	99.94	99.60	**99.95**	99.77
BiDLSTM [18]	99.95	**99.99**	**99.95**	**99.97**	99.95	99.74	99.92	99.83
ANN [59]	99.94	99.82	99.94	99.88	99.95	99.71	99.93	99.82
AE-ANN [59]	99.79	99.56	99.46	99.51	99.78	99.56	98.85	99.20
DBN-ANN [59]	99.94	99.80	99.93	99.86	99.93	99.67	99.87	99.77
SAE-DNN [20]	99.90	99.75	99.76	99.76	99.87	99.52	99.52	99.52
SDAE-SVM [6]	95.05	87.79	89.61	88.67	96.52	87.03	86.53	86.73
Proposed	**99.96**	**99.99**	**99.95**	**99.97**	**99.96**	**99.75**	99.94	**99.85**

Table 11 shows the comparative results on the CIC-IDS2017 dataset. For the binary-class scenario, it can be seen that NB [10] still obtains the worst results with Accuracy of 45.30%, Precision of 62.43%, Recall of 65.20%, and F1-measure of 45.07%, and SDAE-SVM [6] still performs worse than AdaBoost [57]. Except for NB and SDAE-SVM, the results of the other methods are generally satisfactory. Further, LSTM [58] shows better results with Accuracy of 99.76%, Precision of 99.15%, Recall of 99.6%, and F1-measure of 99.40%. Most importantly, our MECNN performs better than other methods regarding all measures. In addition, for the seven-class scenario, the performance of all approaches is deteriorated significantly, which is because the CIC-IDS2017 dataset has a large amount of data. However, the number of samples in some classes is exceedingly small, such as Botnet, Infiltration, etc., which causes a serious data imbalance problem. These four ML methods perform even worse, as they fail to deal with the high-dimensional and large-scale data. Although all these methods perform very competitively, MECNN still has the best results with Accuracy of 99.73%, Precision of 99.10%, Recall of 79.13%, and F1-measure of 83.31%.

These two experiments conducted on the UNSW-NB15 and CIC-IDS2017 datasets have demonstrated the superiority of our proposed MECNN in terms of detection performance of anomalous packets when compared to other state-of-the-art intrusion detection methods.

Table 11
The results of comparative experiments on the CIC-IDS2017 dataset

	Binary-class scenario				Seven-class scenario			
Method	**ACC (%)**	**PRC (%)**	**RCL (%)**	**F1 (%)**	**ACC (%)**	**PRC (%)**	**RCL (%)**	**F1 (%)**
Machine Learning Approaches								
LR [56]	93.45	89.55	89.87	89.71	94.74	37.85	40.12	38.64
NB [10]	45.30	62.43	65.20	45.07	31.00	36.53	75.19	32.23
AdaBoost [57]	99.37	98.90	99.13	99.01	88.86	26.75	23.41	24.47
SVM [11]	93.85	90.02	90.78	90.39	95.87	53.41	44.46	45.90

				Deep Learning Approaches				
GRU [19]	98.93	97.67	99.02	98.33	98.53	77.59	74.23	75.66
LSTM [58]	99.76	99.15	99.66	99.40	98.89	89.22	74.79	77.05
BiDLSTM [18]	98.89	99.56	99.06	99.30	98.89	83.10	74.98	77.03
ANN [59]	98.81	96.40	97.64	97.01	98.77	83.48	74.46	76.91
AE-ANN [59]	98.24	93.30	98.11	95.66	98.13	54.20	53.31	54.74
DBN-ANN [59]	98.78	96.18	97.69	96.93	98.60	82.49	61.77	64.51
SAE-DNN [20]	98.35	96.85	98.05	97.43	97.66	68.85	59.96	60.65
SDAE-SVM [6]	93.42	89.53	89.79	89.66	95.38	52.01	44.43	45.25
Proposed	**99.84**	**99.73**	**99.75**	**99.74**	**99.73**	**99.10**	**79.13**	**83.31**

5.3. Performance Analysis

In this subsection, the difference in response time of MECNN on cloud node and on fog node is first described and then the usage of hardware of MECNN on fog node when handling intrusion detection is analyzed.

5.3.1. Response Time

Since the response time is a critical metric for any IoT network, the length of response time determines whether the IoT network can provide timely services to the users. Therefore, it is necessary to evaluate the difference in the response time of the proposed MECNN deployed on the fog node and cloud node. Fig. 9 shows the difference in response time using MECNN on different numbers of packets (from UNSW-NB15 dataset) under these two environments. It is worth noting that the fog-based model achieves a shorter response time of 30%-45% when compared to the cloud-based model. The shortened response time is due to the fact that the fog nodes are closer to the edge nodes in terms of distance than the cloud nodes, which naturally results in lower network latency. In addition, the longer response time of the cloud-based model is caused by communication delay. When handling network intrusions, the cloud nodes first need to receive traffic records from edge devices through fog nodes and then process them. As a result, the processing time of cloud-based model is always larger than that of the fog-based model, regardless of the size of the data. Therefore, as expected, it is more recommended to perform the MECNN training process, which requires a lot of computing resources, on the cloud nodes and then deploys the trained MECNN models on the fog nodes to detect network intrusions for IoT.

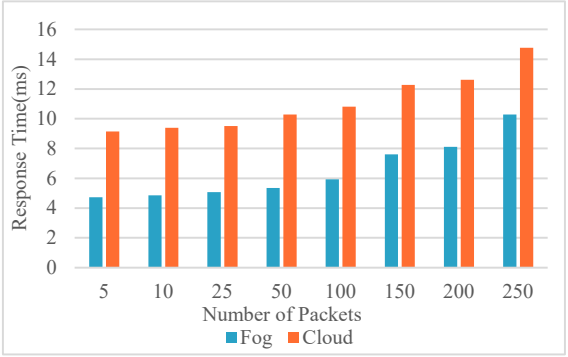


Fig. 9 Average response time for fog-based and cloud-based environments using MECNN.

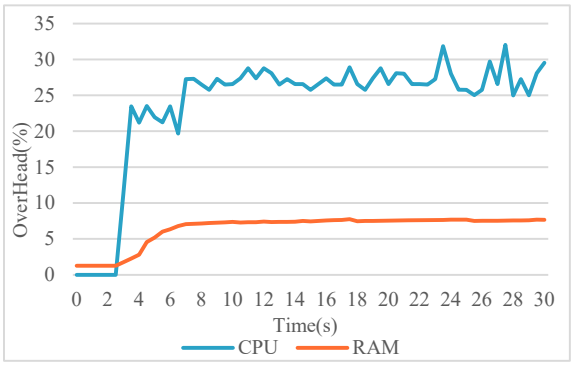


Fig. 10 CPU and memory overhead using MECNN on the fog node.

5.3.2. CPU and Memory Analysis

As described earlier, cloud nodes usually have powerful computing resources. However, the fog nodes do not have this ability. Therefore, it is an indispensable part to study the hardware load of the fog node when handling intrusions or anomalies. For this purpose, some experiments are conducted to study the resource load of the fog node. Fig. 10 shows the CPU and memory usage within 30 seconds of running MECNN on the fog node. The CPU overhead is about 0% at the beginning of the experiment, which is increased to about 22% and then fluctuated between 25% and 30% during the running of the experiment. For the memory overhead, the fog node initially only has 3% of memory usage at the beginning of the experiment. Then, the memory usage is increased to 7%-8% when the experiment is run. These experiments prove that although the fog node does not have as powerful computing power as the cloud nodes, it is still sufficient to meet the hardware requirements of running MECNN to detect network intrusions. Moreover, deploying MECNN on fog nodes rather than on cloud nodes can provide lower latency service.

5.4. Ablation Studies

In MECNN, an accuracy emphasis (AE) module is proposed, which modifies MOEA/D to retain a new solution with a lower detection error rate while preserving the diversity of the population. In this subsection, an ablation experiment is performed to evaluate the effectiveness of AE. This experiment is conducted on the two-class scenario on the CIC-IDS2017, and the same initial population is used for our algorithm with and without the AE module. The average number of parameters and the average error rates obtained by the initial and final populations are shown in Table 12. In the initial population, it can be observed that the average detection error rate is 12.55%. In the final population, the detection error rate is 4.92% when using the AE module, while it will increase to 5.34% without using the AE module, which means that the AE module has about 0.42% improvement to reduce the average detection error rates of the population, validating the effectiveness of the AE module.

Table 12
The ablation study about the AE module

-		Using AE module		Without AE module	
Initial generation		Final generation		Final generation	
Parameters	Error rate	Parameters	Eror rate	Parameters	Error rate
634,292	12.55	2,013	**4.92**	1,088	5.34

6. Conclusions and Future Work

This paper proposes a new multi-objective evolutionary convolutional neural network for intrusion detection system, called MECNN, which can be deployed on the fog nodes of Fog computing to detect network intrusions in the IoT environment. The CNN model is first employed to distinguish different types of IoT traffic records, and an encoding scheme is proposed to transform the architecture of CNN into a chromosome of MOEA/D, which is used to solve the difficulty of parameter tunning for DL. Then, the detection performance and the model complexity of CNN are considered as two objectives, which are optimized by a modified MOEA/D. Additionally, an accuracy emphasis (AE) module is embedded into MOEA/D to enhance its performance, which can retain better-performing individuals without sacrificing the population's diversity. After the evolution of MOEA/D, a set of MECNN models with different model complexities and detection performance can be obtained, and the most suitable MECNN model can be deployed on different fog nodes of fog computing based on their hardware configuration, to provide intrusion detection for IoT. When compared with other state-of-the-art IDSs also based on DL techniques, MECNN significantly simplifies the parameter tuning process for DL model and can provide low-latency and high-accuracy intrusion detection in IoT.

Although the proposed MECNN shows very promising performance, the parameters in the CNN model still need to be manually set in a wide range in advance. Thus, a fully automated method to build the CNN models will be studied in our future work. Moreover, the proposed MECNN shows slightly poor performance when the training data are severely unbalanced. Thus, some effective methods to deal with this imbalance data problem will be investigated in our future work.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under Grant 61876110, the Joint Funds of the National Natural Science Foundation of China under Key Program Grant U1713212, Shenzhen Technology Plan under Grant JCYJ20170817102218122, the Guangdong "Pearl River Talent Recruitment Program" under Grant 2019ZT08X603, and Shenzhen Science and Technology Innovation Commission under Grant R2020A045. This work was also in part support by the CONACyT project no. 1920, a 2018 SEP-Cinvestav grant (application no. 4), and the Basque Government through the BERC 2018-2021 program by the Spanish Ministry of Science.

Reference

[1] J. Rifkin, *The zero marginal cost society: The internet of things, the collaborative commons, and the eclipse of capitalism*: St. Martin's Press, 2014.

[2] C. Perera and A. V. Vasilakos, “A knowledge-based resource discovery for Internet of Things”, *Knowledge-Based Systems*, vol. 109, pp. 122–136, 2016.

[3] E. Anthi, L. Williams, M. Słowińska, G. Theodorakopoulos, and P. Burnap, “A supervised intrusion detection system for smart home IoT devices,” *IEEE Internet of Things Journal,* vol. 6, no. 5, pp. 9042-9053, 2019.

[4]A. Forestiero, “Metaheuristic algorithm for anomaly detection in Internet of Things leveraging on a neural-driven multiagent system”, *Knowledge-Based Systems*, vol. 228, p. 107241, 2021.

[5] M. Hassan, S. Huda, S. Sharmeen, J. Abawajy, and G. Fortino, “An adaptive trust boundary protection for IIoT networks using deep-learning feature extraction based semi-supervised model,” *IEEE Transactions on Industrial Informatics,* 2020.

[6] Z. Lv, L. Qiao, J. Li, and H. Song, “Deep learning enabled security issues in the Internet of Things,” *IEEE Internet of Things Journal*, 2020.

[7] P. Bajpai, A. K. Sood, and R. J. Enbody, “The art of mapping IoT devices in networks,” *Network Security,* vol. 2018, no. 4, pp. 8-15, 2018.

[8] L. Fang, Y. Li, Z. Liu, C. Yin, M. Li, and Z. J. Cao, “A Practical Model Based on Anomaly Detection for Protecting Medical IoT Control Services Against External Attacks,” *IEEE Transactions on Industrial Informatics*, 2020.

[9] M. Al-Omari, M. Rawashdeh, F. Qutaishat, A. H. Mohammad, and N. Ababneh, “An Intelligent Tree-Based Intrusion Detection Model for Cyber Security,” *Journal of Network and Systems Management,* vol. 29, no. 2, pp. 1-18, 2021.

[10] S. Alam, S. K. Sonbhadra, S. Agarwal, and P. Nagabhushan, “One-class support vector classifiers: A survey”, *Knowledge-Based Systems*, vol. 196, p. 105754, 2020.

[11] H. Hindy, E. Bayne, M. Bures, R. Atkinson, C. Tachtatzis, and X. Bellekens, “Machine Learning Based IoT Intrusion Detection System: An MQTT Case Study (MQTT-IoT-IDS2020 Dataset),” *arXiv preprint arXiv:2006.15340*, 2020.

[12] P. F. de Araujo-Filho, G. Kaddoum, D. R. Campelo, A. G. Santos, D. Macêdo, and C. Zanchettin, “Intrusion Detection for Cyber-Physical Systems using Generative Adversarial Networks in Fog Environment,” *IEEE Internet of Things Journal*, 2020.

[13] M. Abdel-Basset, V. Chang, H. Hawash, R. K. Chakrabortty, and M. Ryan, “Deep-IFS: Intrusion Detection Approach for IIoT Traffic in Fog Environment,” *IEEE Transactions on Industrial Informatics*, 2020.

[14] Y. Chen, N. Ashizawa, C. K. Yeo, N. Yanai, and S. Yean, “Multi-scale Self-Organizing Map assisted Deep Autoencoding Gaussian Mixture Model for unsupervised intrusion

detection", *Knowledge-Based Systems*, vol. 224, p. 107086, 2021.

[15] N. Ravi, and S. M. Shalinie, "Semisupervised-Learning-Based Security to Detect and Mitigate Intrusions in IoT Network," *IEEE Internet of Things Journal,* vol. 7, no. 11, pp. 11041-11052, 2020.

[16] J. Liang, E. Meyerson, B. Hodjat, D. Fink, K. Mutch, and R. Miikkulainen, "Evolutionary neural automl for deep learning." *Proceedings of the Genetic and Evolutionary Computation Conference,* pp. 401-409, 2019.

[17] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures." *Proceedings of the Genetic and Evolutionary Computation Conference,* pp. 497-504, 2017.

[18] Y. Imrana, Y. Xiang, L. Ali, and Z. Abdul-Rauf, "A bidirectional LSTM deep learning approach for intrusion detection", *Expert Systems with Applications*, vol. 185, p. 115524, 2021.

[19] M. V. Assis, L. F. Carvalho, J. Lloret, and M. L. Proença Jr, "A GRU deep learning system against attacks in software defined networks," *Journal of Network and Computer Applications,* vol. 177, pp. 102942, 2021.

[20] G. Muhammad, M. S. Hossain, and S. Garg, "Stacked Autoencoder-based Intrusion Detection System to Combat Financial Fraudulent," *IEEE Internet of Things Journal,* 2020.

[21] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary intelligence,* vol. 1, no. 1, pp. 47-62, 2008.

[22] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art", *Knowledge-Based Systems*, vol. 212, p. 106622, 2021.

[23] K. O. Stanley, and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation,* vol. 10, no. 2, pp. 99-127, 2002.

[24] X. Yao, and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE transactions on neural networks,* vol. 8, no. 3, pp. 694-713, 1997.

[25] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE transactions on Neural Networks,* vol. 5, no. 1, pp. 54-65, 1994.

[26] J. Koutník, J. Schmidhuber, and F. Gomez, "Evolving deep unsupervised convolutional networks for vision-based reinforcement learning," *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation,* pp. 541-548, 2014.

[27] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, and N. Duffy, "Evolving deep neural networks," *Artificial intelligence in the age of neural networks and brain computing,* pp. 293-312: Elsevier, 2019.

[28] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," *Proceedings of the first edition of the MCC workshop on Mobile cloud computing,* pp. 13-16, 2012.

[29] Q. Zhang, and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition,"

IEEE Transactions on evolutionary computation, vol. 11, no. 6, pp. 712-731, 2007.

[30] C. Kolias, G. Kambourakis, A. Stavrou, and S. Gritzalis, "Intrusion detection in 802.11 networks: empirical evaluation of threats and a public dataset," *IEEE Communications Surveys & Tutorials,* vol. 18, no. 1, pp. 184-208, 2015.

[31] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," *ICISSp,* vol, 1, pp. 108-116, 2018.

[32] Y. Ren, L. Zhang, and P. N. Suganthan, "Ensemble classification and regression-recent developments, applications and future directions," *IEEE Computational intelligence magazine,* vol. 11, no. 1, pp. 41-53, 2016.

[33] N. Moustafa, B. Turnbull, and K.-K. R. Choo, "An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things," *IEEE Internet of Things Journal,* vol. 6, no. 3, pp. 4815-4830, 2018.

[34] M. Jabbar, and R. Aluvalu, "RFAODE: A novel ensemble intrusion detection system," *Procedia computer science,* vol. 115, pp. 226-234, 2017.

[35] S. U. Jan, S. Ahmed, V. Shakhov, and I. Koo, "Toward a lightweight intrusion detection system for the internet of things," *IEEE Access,* vol. 7, pp. 42450-42471, 2019.

[36] L. Vu, Q. U. Nguyen, D. N. Nguyen, D. T. Hoang, and E. Dutkiewicz, "Learning Latent Representation for IoT Anomaly Detection," *IEEE Transactions on Cybernetics,* 2020.

[37] E. De La Hoz, E. De La Hoz, A. Ortiz, J. Ortega, and A. Martínez-Álvarez, "Feature selection by multi-objective optimisation: Application to network anomaly detection by hierarchical self-organising maps", *Knowledge-Based Systems*, vol. 71, pp. 322–338, 2014.

[38] V. Herrera-Semenets, L. Bustio-Martínez, R. Hernández-León, and J. Van Den Berg, "A multi-measure feature selection algorithm for efficacious intrusion detection", *Knowledge-Based Systems*, vol. 227, p. 107264, 2021.

[39] M. Habib, I. Aljarah, and H. Faris, "A Modified Multi-objective Particle Swarm Optimizer-Based Lévy Flight: An Approach Toward Intrusion Detection in Internet of Things," *Arabian Journal for Science and Engineering,* vol. 45, no. 8, pp. 6081-6108, 2020.

[40] Y. Zhu, J. Liang, J. Chen, and Z. Ming, "An improved NSGA-III algorithm for feature selection used in intrusion detection", *Knowledge-Based Systems*, vol. 116, pp. 74–85, 2017.

[41] Y.-H. Yang, H.-Z. Huang, Q.-N. Shen, Z.-H. Wu, and Y. Zhang, "Research on intrusion detection based on incremental GHSOM," *Chinese journal of computers,* vol. 37, no. 5, pp. 1216-1224, 2014.

[42] W. Wei, S. Chen, Q. Lin, J. Ji, and J. Chen, "A multi-objective immune algorithm for intrusion feature selection," *Applied Soft Computing,* vol. 95, pp. 106522, 2020.

[43] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998.

[44] Y. LeCun, and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks,* vol. 3361, no. 10, pp. 1995, 1995.

[45] X. Zhang, Y. Wang, N. Zhang, D. Xu, and B. Chen, "Research on Scene Classification Method of High-Resolution Remote Sensing Images Based on RFPNet," *Applied Sciences,* vol. 9, no. 10, pp. 2028, 2019.

[46] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[47] K. Miettinen, *Nonlinear multiobjective optimization*: Springer Science & Business Media, 2012.

[48] A. Bala, and A. K. Sharma, "A comparative study of modified crossover operators," *2015 Third International Conference on Image Information Processing (ICIIP),* pp. 281-284, 2015.

[49] E. K. Tang, P. N. Suganthan, and X. Yao, "An analysis of diversity measures," *Machine learning,* vol. 65, no. 1, pp. 247-271, 2006.

[50] Yi, Shanhe, et al. "Fog computing: Platform and applications." *2015 Third IEEE workshop on hot topics in web systems and technologies (HotWeb)*. IEEE, 2015.

[51] M. Abdel-Basset, H. Hawash, R. K. Chakrabortty, and M. J. Ryan, "Semi-Supervised Spatiotemporal Deep Learning for Intrusions Detection in IoT Networks", *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 12251–12265, 2021.

[52] C. A. De Souza, C. B. Westphall, R. B. Machado, J. B. M. Sobral, and G. D. S. Vieira, "Hybrid approach to intrusion detection in fog-based IoT environments", *Computer Networks*, vol. 180, p. 107417, 2020.

[53] A. Gulli, and S. Pal, *Deep learning with Keras*: Packt Publishing Ltd, 2017.

[54] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, and M. Isard, "Tensorflow: A system for large-scale machine learning," *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16),* pp. 265-283, 2016.

[55] R. Panigrahi, and S. Borah, "A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems," *International Journal of Engineering & Technology,* vol. 7, no. 3.24, pp. 479-482, 2018.

[56] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, *Applied logistic regression*: John Wiley & Sons, 2013.

[57] R. Rojas, "AdaBoost and the super bowl of classifiers a tutorial introduction to adaptive boosting," *Freie University, Berlin, Tech. Rep*, 2009.

[58] R. Vinayakumar, K. Soman, and P. Poornachandran, "A comparative analysis of deep learning approaches for network intrusion detection systems (N-IDSs): deep learning for N-IDSs," *International Journal of Digital Crime and Forensics (IJDCF),* vol. 11, no. 3, pp. 65-89, 2019.

[59] S. Gamage, and J. Samarabandu, "Deep learning methods in network intrusion detection: A survey

and an objective comparison," *Journal of Network and Computer Applications,* vol. 169, pp. 102767, 2020.