COOL: Conservation of Output Links for Pruning Algorithms in Network Intrusion Detection

Thi-Nga Dao and HyungJune Lee, *Member, IEEE*

***Abstract*—To reduce network intrusion detection latency in a high volume of data traffic, on-device detection with neuron pruning has been widely adopted by eliminating ineffective connections from a densely connected neural network. However, neuron pruning has a serious problem called output separation in which some parts of neurons can easily be pruned in the middle and become isolated from the rest of the network. To this end, we introduce a solution called the conservation of output links (COOL) pruning method that iteratively preserves a set of effective connections to avoid neuron isolation. We first evaluate COOL on MNIST and CIFAR-10 data sets as well as programmable networking devices, such as P4-supported switches. The experimental results show that COOL outperforms existing methods in terms of both detection time and classification accuracy, especially in extremely sparse networks. Compared to three representative pruning methods, our COOL-based classification model performs at least 25% more accurately with the upper bound for the pruning probability. To further display the effectiveness of COOL-based intrusion detection, we formulate a novel detection time minimization problem by assigning suitable detection models for switches in Internet of Things (IoT) under performance requirements and resource limitations. The experimental results demonstrate that our COOL algorithm is particularly useful for delay-critical and high-traffic applications.**

***Index Terms*—Delay minimization, intrusion detection, output isolation, programmable data plane, weight pruning.**

I. Introduction

The Internet of Things (IoT) allows a massive and unprecedented amount of data to be exchanged [1], [2], [3], [4], making it an attractive environment for attackers and hackers [5], [6], and network attacks keep increasing in both number and volume, posing a dangerous threat to IoT. Therefore, it is crucial to ensure networks' security and privacy. A network intrusion detection system (NIDS) [7], which can early detect and respond to network threats, is a well-known and effective security solution. To deal with the increasing amount of exchanged data, there is a need for NIDS with low-detection delay and high-detection accuracy. Moreover, designing an efficient yet robust detection model feasible at low-end devices becomes important at the edge side.

In network security models [8], [9] traffic is required to transmit to external devices for management, which causes a long detection delay and slow response to network threats. To deal with the problem of large detection time, detection models are usually implemented on edge devices (e.g., switches) that are located close to IoT devices [10], [11], [12], [13]. However, these devices usually have limited computing and memory resources compared to fog or cloud devices. Therefore, for the edge layer, the classification model should have a lightweight yet effective architecture with low-model complexity.

Recently, neural networks (NNs) have emerged as an advanced machine learning (ML) technique with high-classification accuracy. However, the fact that NNs require a large number of floating-point operations and possibly complex activation functions raises two problems: 1) long detection delay and 2) difficulty executing on edge devices. Therefore, a simplification method, such as neuron pruning [12], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], memory reduction [25], [26], [27], and operations simplification [28], [29], [30], is needed to reduce the model complexity of the NN-based models.

To lessen the network complexity, a well-known solution called neuron pruning [12], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [31], [32] with three phases—learning a fully connected (FC) network, removing the weakest connections, and retraining a pruned network—is considered. However, neuron pruning may cause output isolation where at least one output unit has no link to the previous hidden layer after the second phase of connection removal. Since the isolated output neuron, which has no synaptic path from input features, cannot be updated during the third phase, the performance of the pruned model deteriorates significantly. The output isolation problem occurs more frequently when increasing the pruning rate, thus considerably lowering the classification performance. The only explanation offered by existing work is that the lower performance is due to the lack of remaining neurons or connections. To the best of our knowledge, no existing studies recognized the output isolation problem and our work is the first attempt to address this issue. One related problem reported in the literature is layer

Manuscript received 11 June 2023; revised 17 September 2023; accepted 27 September 2023. Date of publication 2 October 2023; date of current version 21 February 2024. This work was supported in part by the Vietnam National Foundation for Science and Technology Development (NAFOSTED) under Grant 102.02-2020.06, and in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant funded by the Korea Government (MSIT) through the Artificial Intelligence Convergence Innovation Human Resources Development (Ewha Womans University) under Grant RS-2022-00155966 and through the Artificial Intelligence Innovation Hub under Grant 2021-0-02068. *(Corresponding author: Thi-Nga Dao.)*

Thi-Nga Dao was with the Faculty of Radio-Electronic Engineering, Le Quy Don Technical University, Hanoi 1000, Vietnam. She is now with the Department of Computer Science and Engineering, Ewha Womans University, Seoul 03760, Republic of Korea (e-mail: daothinga.mta@gmail.com).

HyungJune Lee is with the Department of Computer Science and Engineering, Ewha Womans University, Seoul 03760, Republic of Korea.

Digital Object Identifier 10.1109/JIOT.2023.3321299

collapse in which connections between two consecutive layers are completely removed and all network parameters are untrainable [33]. Unlike layer collapse, only connections to separated output units are not learned in the output separation problem. Layer collapse is essentially an extreme case of output isolation with regard to the position of untrained parameters, and output isolation is a more common phenomenon than layer collapse.

To avoid the output isolation issue, we propose a method, called conservation of output links (COOL) pruning that ensures that all output neurons have at least one synaptic flow to the input layer in the pruned model. In the second phase, if the output isolation problem exists after removing connections with the lowest score, the pruning mask matrices are modified in a bottom-up manner starting from the last hidden layer to the input layer. This modification is terminated when no output neurons are quarantined. It should be noted that COOL can be used with any scoring metric since any weight pruning methods can suffer from the output isolation problem.

We introduce a novel optimization problem to minimize the detection delay of NIDS. Switches are assigned to classify network traffic by using one or more intrusion detection models based on performance requirements, such as classification accuracy and completion ratio, and available resource constraints. To maximize the amount of traffic to be classified, packets from switches that run out of resources can be forwarded to other resource-rich switches. However, since packet offloading can raise the high-transmission cost, we limit the amount of traffic that should be exchanged in the network for intrusion detection. We find optimal solutions for the optimization problem in two cases: 1) with and 2) without the COOL pruning. The experimental results show that the COOL-based detection model achieves a lower average intrusion detection time in IoT and allows many more parameter sets with a feasible solution.

We evaluate the COOL pruning algorithm and compare it with three other pruning algorithms. We conduct experiments on simulated programmable switches using a programming language called P4 [34] to collect the detection delay and classification metrics. The performance results show the effectiveness of our COOL pruning algorithm in terms of conserving accuracy while reducing detection time compared to existing methods. For example, when 60% of connections in fully dense layers are removed, the proposed method achieves a 20% reduction in detection delay with only a 1% drop in accuracy compared to the FC model.

The contributions of this work are listed below.

1) We analyze the output isolation problem in conventional weight pruning methods and derive the closed-form expression of the lower bound isolation likelihood that depends on network parameters and pruning rate.
2) The COOL pruning algorithm avoids the output isolation problem by preserving the strongest connection for each neuron and therefore produces higher accuracy, especially with an extremely sparse model.
3) We introduce the lightweight NIDS incorporating the COOL pruning method to reduce model complexity and detection delay.
4) Using the programmable data plane simulator, we evaluate the COOL-based intrusion detection model and compare it to other methods, which shows that the former produces comparable performance to the latter while considerably reducing detection delay.
5) To further evaluate the COOL method, we formulate a novel optimization problem that minimizes detection time under performance requirements and resource constraints, which shows the COOL scheme reduces the detection time and enables more traffic to be classified.

The remainder of this article is structured as follows. Section II reviews related studies in the literature, and Section III describes the network system as well as network assumption. Then, the intrusion detection model based on the COOL pruning algorithm is introduced in Section IV followed by the detection time minimization strategy in Section V. The performance is subsequently evaluated in Section VI to prove the effectiveness of our proposal. Finally, we draw conclusions regarding this work and suggest possible future studies in Section VII.

II. Related Works

In this section, we first summarize and analyze the drawbacks of previous pruning schemes. Then, we compare existing studies of on-network classification for network intrusion detection with our proposed architecture.

A. Pruning Algorithms

In traditional pruning algorithms, the scoring criterion is a salient factor in determining weak connections in the network. For example, Chandakkar et al. [14] and Han et al. [15] used the smallest absolute weight values as the scoring metric. In [17], information from all second-order derivatives of the error function was used to find pruned neurons. Meanwhile, Molchanov et al. [16] removed a feature map based on the loss change with the first-order derivative term. Yu et al. [18] introduced a neuron importance score propagation algorithm that measures the neuron score at a layer based on the score of neurons in the succeeding layer and connections between these two layers. Moreover, Luo and Wu [19] used an entropy-based criterion to determine the importance of filters in convolutional neural networks (CNNs). Hu et al. [20] computed the Average Percentage of Zeros (APoZs) of neurons for a given data set and eliminated neurons with high-APoZ values. In [21], random pruning at initialization was evaluated and compared with the FC network. In [22], neurons in a specific layer were given importance scores that were backpropagated from output to input and depended on the activation value of neurons in the preceding layer and weight between the current layer and preceding layer.

Conventional pruning methods sometimes face the critical problem of layer collapse [33] in which all connections of a specific layer are completely removed. This problem becomes more likely when the compression rate increases. Since layer collapse leads the model to be untrained, the pruning method should be designed to avoid this problem. For example, Tanaka et al. [33] found that larger layers tend to

TABLE I
COMPARISON BETWEEN COOL AND RELATED PRUNING WORK

References	Pruning criteria	Probabilistic/Deterministic	Data-aware/Data-agnostic	Output isolation
[12]	Weight rank	Probabilistic	Data-aware	Can exist
[14], [15]	Weight magnitude	Deterministic	Data-aware	Can exist
[16], [17]	Gradient of loss wrt weights	Deterministic	Data-aware	Can exist
[18]	Neuron score	Deterministic	Data-aware	Cannot exist
[19]	Entropy of filters	Deterministic	Data-aware	Cannot exist
[20]	APoZ	Deterministic	Data-aware	Cannot exist
[21]	Random	Deterministic	Data-agnostic	Can exist
[22]	Neuron score	Deterministic	Data-aware	Cannot exist
[23]	L1 norm of weight	Deterministic	Data-aware	Cannot exist
[24]	L1 norm of weight	Probabilistic	Data-aware	Can exist
COOL	Any weight scoring metric	Deterministic	Data-aware	Cannot exist

have lower scores than small layers due to the conservation of synaptic saliency. Therefore, they applied iterative pruning to avoid layer collapse by gradually reducing the size of larger layers or reducing the gap between scores of layers. Gupta et al. [35] and Lee and Yim [36] introduced a minimum threshold of weights or channels to be kept in each layer to guarantee connections between layers.

In our work, we consider a related problem, output separation, which has no synaptic flow from an output neuron to the input layer. Although the pruned network can still be trained, the output isolation issue significantly reduces performance since the parameters of one or more output units cannot be updated during training. Note that methods in [33], [35], and [36] focused on mitigating layer collapse, not output separation. Hence, this article analyzes the output separation problem and provides a possible solution to the output isolation problem.

As shown in Table I, we compare pruning methods based on four different metrics: 1) the method's pruning criteria; 2) whether the method is probabilistic or deterministic; 3) whether the method is data aware or data agnostic; and 4) the method's output isolation probability. Unlike other pruning methods, the COOL algorithm can be used with any weight-scoring metric. Weights are removed with a probability in a probabilistic pruning algorithm, while weights with the lowest scores are completely pruned in a deterministic pruning algorithm. If weight scores are computed on a training data set, a pruning algorithm is data-aware; otherwise, it is called data-agnostic. We also summarize pruning algorithms' output isolation probability in the last column of Table I.

B. On-Device Classification for Network Intrusion Detection

Researchers have investigated on-network classification using programmable switches for multiple applications [37], [38], [39]. On-network classification is especially useful for latency-critical applications thanks to the early response to detected events. It has been proved that various ML techniques can be executed in the data plane. In [37], four different supervised and unsupervised methods consisting of decision tree (DT), support vector machine (SVM), Naive Bayes (NB), and K-means were implemented in both software and hardware (NetFPGA). Specifically, a parser block acted as a features extraction module and a match-action pipeline served as a traffic classifier. The authors showed classification performance at a full line rate on real-world data sets.

As embedding network security into programmable switches enables packet processing at the line rate, it has received considerable attention. Xavier et al. [39] developed a simple and quite accurate DT-based framework that can be deployed into the programmable switches and validated the framework for an intrusion detection task using the BMv2 emulator and Netronome SmartNIC board. The performance results show high accuracy (above 95%) with little performance drop, even for a large number of flows. Other on-network detection studies have focused on detecting a specific common attack [40], [41], [42], [43]. For example, Musumeci et al. [40] developed a distributed denial-of-service attack detection framework assisted by ML techniques, such as SVM and random forest (RF). This framework includes two main phases: 1) feature extractor and 2) ML classifier. The P4 switch extracts necessary packet features and sends them to an external device for traffic classification. Unlike [40], we implement both feature extraction and intrusion detection on a programmable data plane. Therefore, there is no delay caused by packet transmission from the switch to the packet classifier.

To determine volumetric DDoS attacks, Ding et al. [41] and Turkovic et al. [42] estimated flow cardinality based on a count-min sketch, which is a probabilistic and memory-efficient data structure in which a packet arrives at a programmable switch and then a flow key is extracted and fed to multiple hash functions to compute register indices where counters are stored. Friday et al. [43] introduced a P4-based DDoS detection and mitigation scheme that can measure the bandwidth used by various applications and monitor the traffic rate of TCP connections, executing the tracking data rate in the data plane via a Bloom filter. Our work differs from [40], [41], [43], and [42] in the following two ways.

1) We consider the problem of classifying incoming traffic into various classes, including normal and a specific attack type. This is a more general problem that can help operators to gain insights into intrusion behaviors.
2) We implement a more complex ML technique (i.e., NN) in a P4 switch to further improve classification performance.

With the aim of building a lightweight intrusion detection model, Lei et al. [44] combined a magnitude pruning

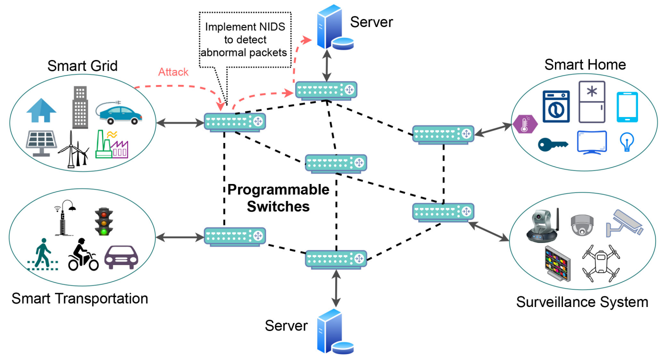


Fig. 1. Overview of NIDS in IoT.

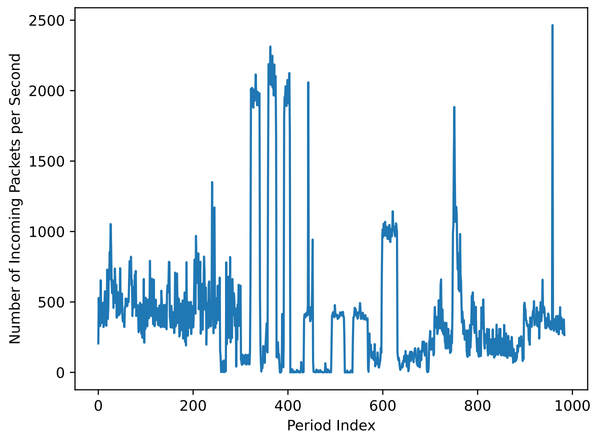


Fig. 2. Change in traffic rate in the IoT Korea data set.

algorithm and a deep NN consisting of 11 ReLU hidden layers. Although they claimed that this reduced model complexity while achieving strong classification performance, the evaluated architecture was still heavy and difficult to deploy in the data plane. To cope with this issue, we propose a more lightweight COOL pruning-based detection architecture with only one hidden layer that is suitable for programmable switches. Moreover, the COOL pruning method can overcome a problem raised by general pruning techniques (i.e., output separation).

III. Network System

IoT networks consist of IoT devices, servers, and switches that connect IoT devices and servers, as presented in Fig. 1. Data traffic generated by IoT devices from various applications can be transmitted to servers via switches. Participating switches are connected using an arbitrary network topology, such as a bus, star, tree, ring, or mesh as shown in Fig. 1. These switches are in charge of data traffic forwarding as well as intrusion detection to ensure that only legal data are forwarded.

It is assumed that IoT devices generate data periodically or whenever they detect an event, such as a fire, a gas leak, or smoke release. Hence, the data rate may vary considerably over time. As an example, Fig. 2 demonstrates the changes in data rate in the IoT Korea data set [45]. Specifically, we measure the number of incoming packets per second at a switch during nearly 1000 1-s periods. Generally, there is a significant change in data rate in the time span. For example, there are around 500 incoming packets per second in the beginning, and this number can increase up to 2500 or drop to nearly 0 throughout the experimental duration.

Since networking devices' incoming data rate varies over time, a switch's detection model should be adapted accordingly. Generally, model complexity is inversely proportional to model performance. Therefore, when the data rate is low, the switch can implement a complex detection model to achieve high-classification performance. In contrast, with a high-traffic rate, a simpler prediction model with low complexity should be utilized to maximize the amount of data traffic to be examined.

To quickly detect and respond to network threats, we execute the traffic classification model on a programmable data plane. More specifically, with data plane programmability, we can easily add customized packet processing functions to edge devices, thus significantly reducing detection delay. There are multiple commercial programming edge devices, including NetFPGA SUME, developed by Digilent [46] and Intel Tofino2 [47]. When a data packet arrives at the input port, the switch predicts the data label. According to [45], there are five different traffic labels: 1) normal; 2) reconnaissance; 3) man-in-the-middle; 4) denial-of-service; and 5) Botnet. Based on the prediction output, the switch then takes an appropriate action for this packet (e.g., forwarding the packet, dropping the packet, or adding an alarm field in the packet header).

In our work, each programmable switch decides the detection model independently. Multiple factors should be taken into account: the incoming data rate, available computing resources, and accuracy and detection time of each detection model. Note that there is a tradeoff between the traffic classification model's accuracy and detection time. Specifically, if using a model with high complexity, we can achieve high accuracy with a sacrifice in detection time and vice versa. Therefore, participating switches should carefully consider these factors.

In the following section, we introduce a lightweight detection and classification model based on the COOL pruning method. Then, in Section V, we propose an optimization problem for minimizing detection latency to select a suitable model given constraints on computing resources and classification performance.

IV. NIDS Based on the COOL Pruning Method

We aim to develop a timely and lightweight network intrusion detection architecture by incorporating the detection model with a pruning method. In this section, we first introduce the output isolation problem inherent in the weight pruning method. Then, the pruning algorithm based on the COOL is proposed to address the output isolation problem. Finally, we present the traffic classification model on the programming switch with the support of the COOL pruning algorithm.

The training procedure of traditional pruning methods requires three phases for the training procedure: 1) learning

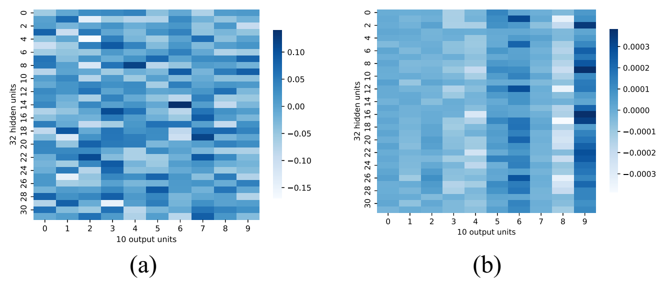


Fig. 3. Scores for links between the last hidden layer and output layer based on (a) weight and (b) gradient metrics.

the FC model; 2) pruning unnecessary connections; and 3) retraining the pruned network. We define x, h, and y as the vectors for the input, hidden, and output units, respectively. In the first phase, parameters, including weights and biases, are trained by minimizing the entropy-based loss function. In the next phase, the trained weights with the lowest scores are removed from the network in a layerwise manner. We define the pruning rate p_{prune} $(0 \leq p_{\text{prune}} \leq 1)$ as the ratio of the number of removed connections to the total number of connections of the FC layer. Assuming that connections between two consecutive layers are pruned with the same ratio p_{prune}, then the pruning rate of the whole network is also p_{prune}. To present the connection state of a layer, a binary mask matrix is used in which ones imply remaining links and zeros indicate pruned flows. In the final phase, the remaining connections of the pruned network are retrained, which is called fine-tuning. The entropy-based loss function is still used for retraining. A weight matrix is multiplied by a corresponding binary mask matrix so that the pruned weights are not trained in this phase.

However, some output neurons may not have any connections to the previous layer, which is an inherent limitation of the weight pruning method called output isolation. Since isolated output neurons have the same value for all samples, the classification performance decreases significantly, especially with a high-p_{prune} value and few hidden units in the previous layer. To gain insight into this issue, we first derive the lower bound value for the probability of the output isolation problem p_{oi} and then propose an algorithm that conserves connections for output units to overcome this problem.

It is assumed that scores of links between two consecutive layers are random numbers with a uniform distribution or there is no specific order for connection scores. To substantiate our assumption, we demonstrate scores of connections between the output layer and the preceding layer when training a dense NN to classify hand-written digits in the MNIST data set. Two different scoring metrics are used, weight-based and gradient-based, as shown in Fig. 3. The x-axis represents ten output units, while the y-axis accounts for 32 hidden features. In this example, connection scores with the weight-based metric are more uniformly distributed than those with the gradient-based metric, in which high scores focus on links to the outputs 6 and 9. Note that when scores are distributed uniformly in a specific range, the output isolation probability becomes the lowest and achieves the lower bound. For our example in MNIST, p_{oi} in Fig. 3(a) is lower than that in Fig. 3(b). Therefore, we assume that scores of connections between the output layer and the last hidden layer follow a uniform distribution and derive the lower probability of an output isolation problem.

We define n_k as the number of units in layer k with $0 \leq k \leq H+1$, where layer 0 denotes the input layer and layer $H+1$ is the output layer. Let n_{H+1} and n_H denote the number of output and hidden features in the second-to-last layer, respectively. Then, the number of connections between the hidden and output layers in the FC model is $n_H n_{H+1}$. In the FC model, each output neuron has n_H links to the previous layer. Since the events of pruning connections are independent, we compute the joint probability of removing all n_H links as the product of probabilities of trimming n_H links. Thus, the probability that the first link is deleted from the FC model after removing the paths with the lowest scores is $p_1 = ([n_H n_{H+1} p_{\text{prune}}]/n_H n_{H+1}) = (n'/n)$ where $n = n_H n_{H+1}$ and $n' = n p_{\text{prune}}$ for simplification. The probability that the second link is trimmed is $p_2 = (n' - 1/n - 1)$. Similarly, the probability that the link n_H is removed is $p_{n_H} = ([n' - n_H + 1]/n - n_H + 1)$. Finally, the joint probability that an arbitrary output neuron has no link to the preceding hidden layer is

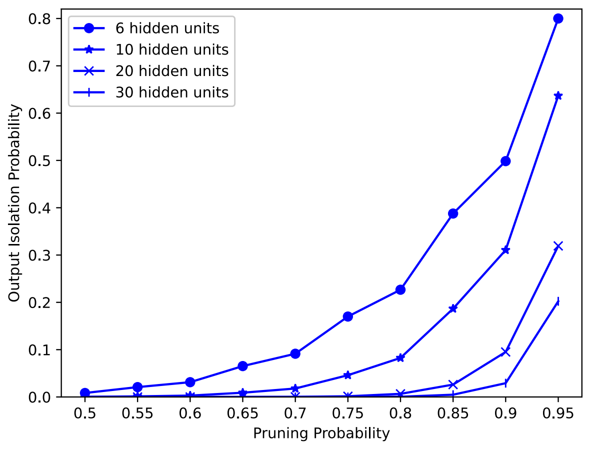
$$p_{oi} = \prod_{k=1}^{k=n_H} p_k = \frac{n'\left(n'-1\right)\left(n'-n_H+1\right)}{n(n-1)(n-n_H+1)}. \tag{1}$$

We remove connections between two consecutive layers with pruning rate p_{prune} only, and we iteratively compute the probability of pruning one connection among n_H connections to derive the isolation probability of an arbitrary output neuron.

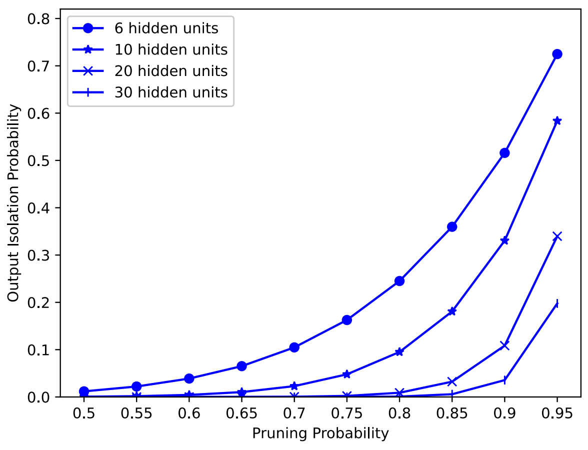
Fig. 4(a) shows the lower bound of p_{oi} when there are five attack labels. $n_{H+1} = 5$ is taken from the IoT data set [45]. The number of hidden neurons varies from 6 to 30, and p_{prune} is set between 0.5 and 0.95 with a step size of 0.05. As shown in Fig. 4, the output isolation probability increases as p_{prune} increases or n_H decreases. For example, with six hidden units, $p_{oi} = 0.778$ when $p_{\text{prune}} = 0.95$ compared to only 0.013 when $p_{\text{prune}} = 0.5$. As p_{prune} is less than 0.5, the p_{oi} value almost reaches 0, which means the output isolation problem rarely occurs. Since the p_{oi} value slowly increases with a low-pruning probability and then shows exponential growth when p_{prune} becomes high, the p_{oi} graph is quite similar to an exponential function. However, unlike an exponential function, which can approach infinity, the upper bound value for p_{oi} is 1.

We also evaluate the output isolation probability with ten traffic labels taken from the UNSW-15 data set [48]. As can be seen in Fig. 4(b), with $n_{H+1} = 10$, the p_{oi} value is slightly lower than that with $n_{H+1} = 5$, as there are more connections between the output and the previous layers (n) in the FC model when $n_{H+1} = 10$. Hence, with the same pruning probability, it is less likely that all links to an output neuron are removed in the pruned network.

To avoid the output isolation problem, we design the pruning method in a bottom-up manner as follows. First, the maximum score value $\max(S_{H+1})$ of connections between the second-to-last and the last layers is computed and the strongest connection for each output neuron is conserved by setting



(a)



(b)

Fig. 4. Lower bound for the output isolation probability when (a) $n_{H+1} = 5$ and (b) $n_{H+1} = 10$.

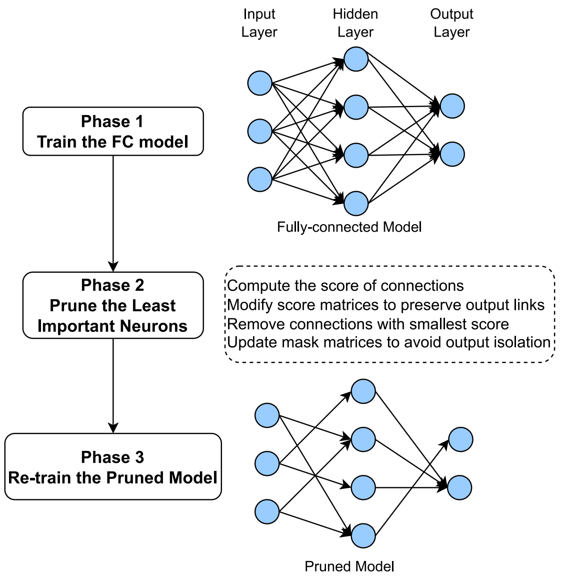

Input Layer
Hidden Layer
Output Layer
Phase 1
Train the FC model
Fully-connected Model
Phase 2
Prune the Least Important Neurons
Compute the score of connections
Modify score matrices to preserve output links
Remove connections with smallest score
Update mask matrices to avoid output isolation
Phase 3
Re-train the Pruned Model
Pruned Model

Fig. 5. Construction of pruned NN with conservation of output connections.

the score for this connection to $\max(S_{H+1})$. Similarly, for connections between the third-to-last and the second-to-last layers, we set the score of the strongest connection for each remaining hidden neuron in layer H to $\max(S_H)$. The procedure continues until we trim unimportant links between the input layer and the first hidden layer.

We define $p_{\max}$ as the maximum pruning probability to ensure that each output neuron has at least one connection to the previous layer. If $p_{\text{prune}} > p_{\max}$, it is impossible to avoid the output separation problem. Considering connections between layer k and $k+1$, the minimum number of remaining paths to conserve n_{k+1} units is n_{k+1} and the maximum pruning ratio at layer k is $1-(n_{k+1}/n_k n_{k+1}) = 1-(1/n_k)$. Then, $p_{\max} = \min_k(1-[1/n_k])$.

We use a binary mask matrix M_k with the same size as the weight matrix W_k to denote the pruning status of links from layer k to the succeeding layer. For example, M_1 in Fig. 5 is $M_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}^T$. To check whether an output neuron is

Algorithm 1 COOL Pruning Algorithm

```
    Input: Network architecture $n_k, 0 \le k \le H+1$ and pruning rate
    Phase 1: Train the FC model
1:  Derive the score matrices for connections $\{S_0, S_1, ..., S_H\}$
    Phase 2: Prune connections with the lowest scores
2:  $k = H$
3:  while $k \ge 0$ do
4:      Initialize updated score matrix $S'_k = S_k$
5:      $l_k$ : list of neurons in layer $k$ with link to output layer
6:      Compute $\max(S_k)$
7:      for $i \in l_k$ do                              ▷ Modify $S_k$
8:          $tmp = S_k[:, i]$                  ▷ Extract a column of $S_k$
9:          $index = \text{argmax}(tmp)$
10:         $tmp_{index} = max(S_k)$
11:         $S'_k[:, i] = tmp$                   ▷ Updated score matrix
12:     end for
13:     Based on updated score matrix $S'_k$, remove weights with lowest score in layer $k$ to obtain mask matrix $M_k$
14:     $k = k - 1$
15: end while
    Phase 3: Re-train pruned model with mask matrices $M_k$
    return The pruned model
```

isolated, matrix $\mathbf{M}$ with n_0 rows and n_{H+1} columns is defined as $\mathbf{M} = \prod_{k=0}^{H} M_k$. If the value at row i and column j is 0 ($\mathbf{M}_{i,j} = 0$), there is no synaptic flow from input feature i to output neuron j. We define an isolation vector I with shape $(1, n_{H+1})$ as follows:

$$I_j = \sum_{i=1}^{n_0} \mathbf{M}_{i,j}, \quad 1 \le j \le n_{H+1} \tag{2}$$

where the jth element of I indicates the isolation status of output unit j. Specifically, $I_j = 0$ implies that the output neuron j has no connection path to the input layer.

Algorithm 1 presents our COOL pruning scheme to avoid the isolation of output neurons. The network structure is used as the input of Algorithm 1 with the number of input features n_0, the number of output neurons n_{H+1}, and the number of units in hidden layers. Our three-phased pruning scheme mostly differs from other pruning methods in the second phase. After training an FC model in the first phase, we obtain the score matrices for connections in the network $\{S_0, S_1, \ldots, S_H\}$. Matrix S_k represents the score for connections between layers k and $k+1$; the shape of S_k is the same as that of connection weights W_k between these two layers. For example, if layers k and $k+1$ have n_k and n_{k+1} units, respectively, then $S_k \in \mathbb{R}^{n_k \times n_{k+1}}$. Then, in the second phase, layer index k is initialized to the number of hidden layers H in line 2. We modify the original score matrix S_k from lines 7 to 11 and save it to the newly updated matrix S'_k. Specifically, each neuron in layer k should have at least one strong link with a weight equal to the maximum value of S_k, so that this link cannot be deleted. After obtaining S'_k, we apply the weight pruning scheme to layer k (line 13) and obtain the mask matrix M_k. Then, layer index k is reduced by 1 (line 14) and score matrix modification repeats until the input layer ($k = 0$). Finally, we fine-tune the pruned model using the mask matrix M_k in the last phase. The COOL algorithm is not dedicated to a specific scoring metric but can be used for any weight scoring metric to avoid the output isolation problem as shown in Fig. 5. Since the maximum number of iterations needed for the second phase of Algorithm 1 is the sum of the number of hidden units and output neurons in the network, the algorithm complexity is $O(\sum_{k=1}^{H+1} n_k)$ where n_k is the number of units in layer k.

Note that since the P4 language only supports integer and binary operations, we need to convert the trained network parameters into integer values. Assuming that χ bits are used to represent the fractional part of parameters, we now explain how to compute binary output neurons for a simple NN with one hidden layer in P4. W_1 and b_1 denote the trained weight and bias float values of the hidden layer, respectively. Then, the binary hidden vector h_{bin} is derived as follows:

$$W_{1,\text{bin}} = \lfloor W_1 \times 2^{\chi} \rceil \tag{3}$$
$$X_{\text{bin}} = \lfloor X \times 2^{\chi} \rceil \tag{4}$$
$$b_{1,\text{bin}} = \lfloor b_1 \times 2^{2\chi} \rceil \tag{5}$$
$$h_{\text{bin}} = \begin{cases} \left\lfloor \frac{W_{1,\text{bin}} X_{\text{bin}} + b_{1,\text{bin}}}{2^{\chi}} \right\rceil, & \text{if } W_{\text{bin}} X_{\text{bin}} + b_{\text{bin}} \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where the $\lfloor\rceil$ notation denotes the round operation to the nearest integer number. The binary output features y_{bin} can be computed similarly as follows:

$$y_{\text{bin}} = W_{2,\text{bin}} X_{\text{bin}} + b_{2,\text{bin}} \tag{7}$$

where $W_{2,\text{bin}}$ and $b_{2,\text{bin}}$ are the binary weight and bias parameters between the hidden and output layers.

When the pruned network is fine-tuned, the parameters are sent to programmable switches. Each switch computes the output values y that represent the probability of traffic classes for an incoming packet. Then, the packet is classified into the label argmax(y). Depending on the classified label, we can take different actions for this packet. For example, the packet can be forwarded normally or dropped, or an alarm field can be added to the packet header at the switch.

TABLE II
LIST OF MAIN NOTATION IN INTEGER LINEAR PROGRAMMING FORMULATION

Switch parameters	
S	Number of switches
$\mathbb{S}$	Set of switches
N	Total number of packets generated per second
N_i	#Generated packets sent to switch i per second
y_{ij}	#Packets classified by switch i using model j
y_{ij1}	#Non-offloaded packets classified by switch i w/ model j
y_{ij2}	#Offloaded packets classified by switch i w/ model j
x_{ik}	#Packets offloaded from switch k to switch i
n	#Packets classified by all switches
$t_{c,i}$	Classification time constraint for switch i
h_{ik}	Hop count from switch i to switch k
Classification model parameters	
C	Number of classification models
A_j	Classification accuracy of model j
T_j	Average classification time (second) of model j
a_{req}	Classification accuracy requirement
r_{req}	Completion ratio requirement
$t_{req,i}$	Classification time constraint at switch i
c_{req}	Transmission cost constraint

V. Detection Time Minimization Strategy

The main objective is to minimize the average detection time of packets classified by all switches in the network given the constraints of accuracy requirement, completion ratio, computing resource, and offloading cost. Table II presents the main notation of the optimization problem. We assume that the network has a set $\mathbb{S}$ of S switches and there are N_i packets per second purely forwarded to switch i. Switches are in charge of forwarding incoming packets to the destination and detecting any abnormal behaviors from incoming packets. Depending on the available computing resources, a switch may ask a neighboring switch to perform the classification task or accept to perform this task for nearby switches. We define x_{ik} as the number of packets offloaded from switch k to switch i. To reduce the offloading cost caused by packet transmission between switches and to reduce the detection time, we assume that a switch only asks neighboring switches to perform packet classification when this switch has insufficient computing resource. The hop count between switch i and k is denoted by h_{ik}. The relationship between h_{ik} and x_{ik} as follows:

$$x_{ik} = \begin{cases} 0, & \text{if } h_{ik} > 1 \\ \geq 0, & \text{otherwise.} \end{cases} \tag{8}$$

The total number of packets generated in the network per second is $N = \sum_{i=1}^{S} N_i$. Suppose that there are C possible classification models and each switch needs to choose the appropriate classification model for incoming packets. The classification accuracy and detection time for model j are denoted by A_j and T_j, respectively.

We define an integer assignment variable y_{ij1} as the number of packets that are purely forwarded to switch i and classified by model j at switch i. Meanwhile, the variable y_{ij2} is the number of packets that are offloaded to switch i from nearby switches and classified by model j at switch i. Then, the total number of packets classified by switch i using model j is denoted by $y_{ij} = y_{ij1} + y_{ij2}$.

The ILP can be formulated as follows:

$$\min \quad \sum_{i=1}^{S}\sum_{j=1}^{C}\Big(y_{ij1}T_j + y_{ij2}\big(T_j + T_{\text{hop}}\big)\Big) \tag{9}$$

subject to

$$\frac{1}{n}\sum_{i=1}^{S}\sum_{j=1}^{C}A_j y_{ij} \geq a_{\text{req}} \tag{10}$$

$$\frac{n}{N} \geq r_{\text{req}} \tag{11}$$

$$n = \sum_{i=1}^{S}\sum_{j=1}^{C} y_{ij} \tag{12}$$

$$\sum_{j=1}^{C} y_{ij}T_j \leq t_{\text{req},i} \quad \forall i \in \mathbb{S} \tag{13}$$

$$t_{\text{req},i} \leq 1 \quad \forall i \in \mathbb{S} \tag{14}$$

$$\sum_{i=1}^{S}\sum_{k=1}^{S} x_{ik} \leq c_{\text{req}}, \quad i \neq k \tag{15}$$

$$\sum_{j=1}^{C} y_{ij2} = \sum_{k=1}^{S} x_{ik} \quad \forall i \in \mathbb{S} \tag{16}$$

$$y_{ij} = y_{ij1} + y_{ij2} \quad \forall i \in \mathbb{S}\ \forall j \in \mathbb{C} \tag{17}$$

$$y_{ij} \leq \frac{1}{T_j} \quad \forall i \in \mathbb{S}\ \forall j \in \mathbb{C} \tag{18}$$

$$\sum_{j=1}^{C} y_{ij1} \leq N_i \quad \forall i \in \mathbb{S} \tag{19}$$

$$n \leq \sum_{i=1}^{S} N_i \tag{20}$$

$$0 \leq y_{ij1} \leq N_i \quad \forall i \in \mathbb{S}\ \forall j \in \mathbb{C} \tag{21}$$

$$0 \leq y_{ij2} \quad \forall i \in \mathbb{S}\ \forall j \in \mathbb{C} \tag{22}$$

$$0 \leq x_{ik} \leq N_k \quad \forall i \in \mathbb{S} \tag{23}$$

$$\sum_{i=1}^{S} x_{ik} \leq N_k. \tag{24}$$

The objective in (9) is to minimize the classification delay of packets by all switches in the network. Constraint (10) ensures that the average classification accuracy exceeds a given threshold value a_{req}, while (11) and (12) guarantee the completion ratio requirement is met. Meanwhile, (13) and (14) ensure that the total classification time switch i spends per second is less than value $t_{\text{req},i}$ with the maximum value of 1 s. Inequation (15) limits the total number of offloaded packets to reduce the transmission cost in the network. Finally, the remaining equations and constraints indicate the feasible range of variables under consideration.

VI. Experimental Results

A. Network Setup

To evaluate the COOL pruning algorithm, we consider three different data sets: 1) MNIST; 2) CIFAR-10; and 3) the IoT data set [45]. Since the COOL algorithm can be used with any scoring criterion, the weight magnitude is selected as a scoring metric, as its simplicity and effectiveness have been proven in the literature. Using the MNIST and CIFAR-10 data sets, we show the benefits of the proposed pruning algorithm compared to some existing pruning methods. Then, the COOL-based network intrusion detection model is evaluated using the IoT data set. The samples in the IoT data set were collected from a wireless network, including smart home devices (i.e., intelligent speakers and Wi-Fi cameras) and laptops as well as smartphones. The data distribution of this data set with five different traffic classes is shown in Table III. The normal and Botnet data are major classes containing 58.82% and 34.76% of the whole data set, respectively. The remaining three attack types are the minority labels. We randomly divide the whole data set into training and test sets. Network parameters are trained using the training data, while the performance of the evaluated model is measured on the test set only. The experiments are conducted on a desktop PC with an Intel Core i7 2.5-GHz CPU (with the Radeon R9 M370X 2048 MB and Intel Iris Pro 1536-MB GPU support) and 16-GB RAM, which is comparable to a normal or low-end switch specification.

TABLE III
Label Distribution of IoT Network Intrusion Data Set

	# of Packets	Percentage (%)
Normal	1,756,276	58.82
Reconnaissance	25,210	0.84
MitM	101,885	3.41
DoS	64,646	2.16
Botnet	1,037,977	34.76
Total	**2,985,994**	**100**

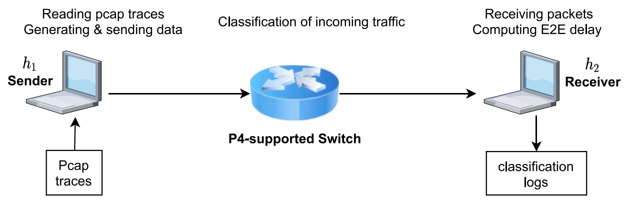


Fig. 6. Network topology used to collect E2E delay.

To estimate the detection models' detection delay, we consider a network with two hosts and one programmable switch that connects these hosts, as shown in Fig. 6. More specifically, the host h_1 extracts data traffic from pcap traces of the IoT data set and then sends data packets to the host h_2 via the switch. The switch is in charge of monitoring incoming data by classifying the data packets into one of five different traffic classes. A variety of classification models are implemented in the switch using the P4 programming language. It is assumed that we forward all packets from h_1 to h_2 to measure the average end-to-end (E2E) delay of detection models under consideration. In fact, the switch takes different actions for incoming packets: forwarding, dropping, and adding an alarm header to the packet. The network emulator Mininet [49] is used to define the network topology, including the number of hosts, switches, and network parameters (e.g., bandwidth and link delay). The Python-based Library Scapy is used for packet generation and transmission at h_1 as well as a reception at h_2.

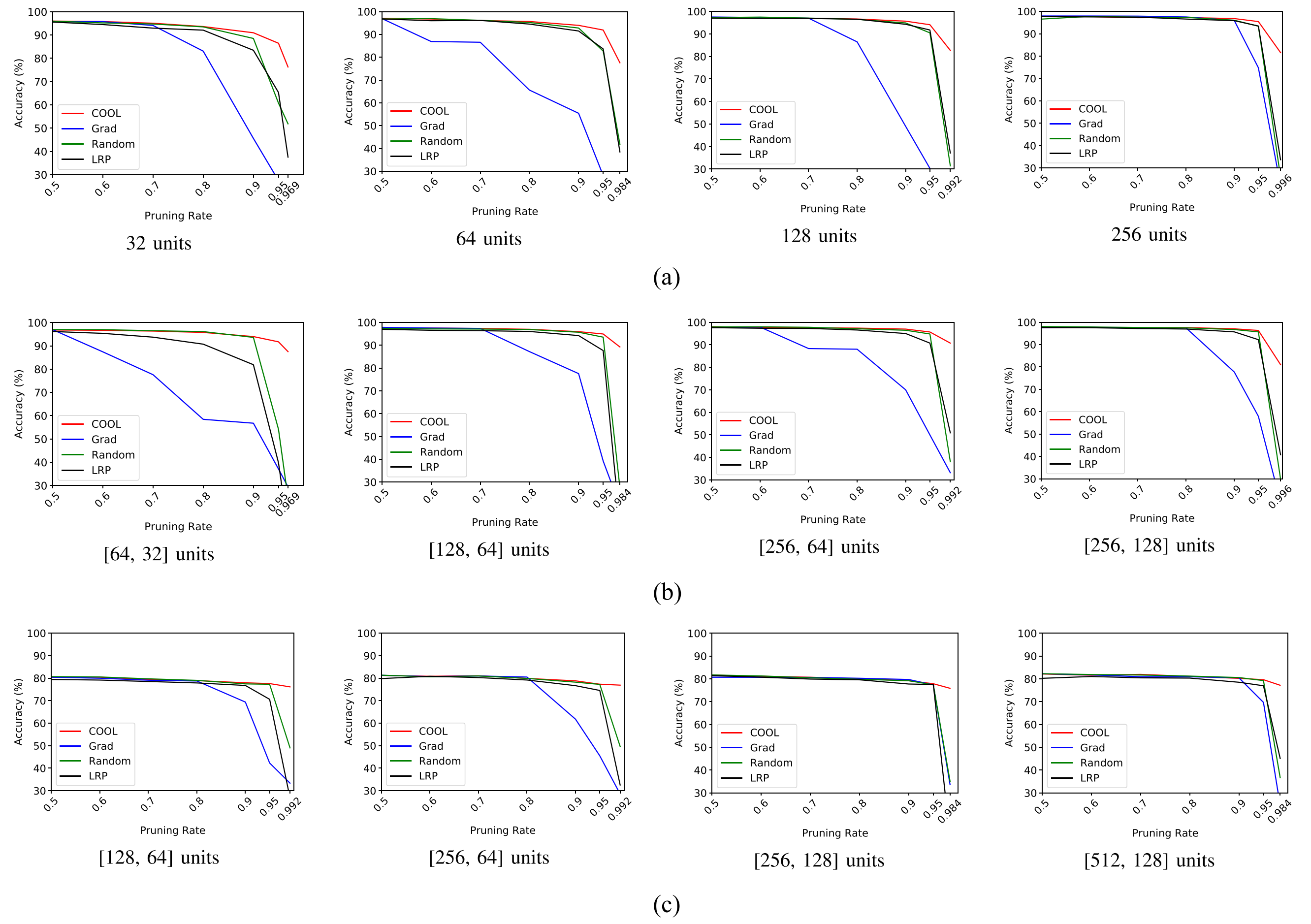


Fig. 7. Comparison of pruning algorithms' performance on MNIST and CIFAR-10. (a) NN with one hidden layer on the MNIST data set. (b) NN with two hidden layers on the MNIST data set. (c) CNN with two dense layers on the CIFAR-10 data set.

To evaluate the proposed architecture, we consider different performance metrics, including classification accuracy (percentage of correct predictions), false positive (FP) rate (rate of normal labels misclassified as attack labels), false negative (FN) rate (rate of attack labels misclassified as normal labels), and false interattack (FI) rate (rate of an attack type misclassified as another attack type).

B. Comparison of Pruning Methods' Performance on MNIST and CIFAR-10 Data Sets

We selected three well-known and up-to-date methods—gradient-based pruning [14], random pruning [21], and LRP [22] pruning—for the performance comparison using the MNIST and CIFAR-10 data sets. Fig. 7 shows the classification accuracy of NN-based classifiers with the pruning algorithms under consideration. Pruning methods are evaluated with different levels of model complexity by changing the number of hidden units and pruning rates. For our experiments, we apply pruning algorithms to dense hidden layers, and the number of hidden units is written below each subfigure. For object classification in the CIFAR-10 data set, we build a CNN with three convolutional layers of 128, 256, and 256 3×3 filters followed by two fully dense hidden layers. Dropout with a connection drop rate of 0.3 is used for better generalization. In all pruning methods, accuracy tends to degrade significantly when using a high-pruning ratio. Among the four considered pruning methods, LRP and our method can avoid output isolation. More specifically, LRP removes unnecessary hidden neurons/channels from the network. By ensuring that there is at least one neuron in hidden layers, LRP can avoid output isolation. However, our method outperforms LRP, especially when the pruning rate approaches the upper bound $p_{\max}$, due to the advantage of weight pruning over neuron pruning in terms of the number of remaining neurons. Both weight and neuron pruning methods preserve the same number of connections; however, more hidden neurons are generally conserved in weight pruning. Therefore, more informative features can be propagated through the network in weight pruning. Meanwhile, since gradient and random pruning methods suffer from output separation, they exhibit lower performance than LRP and our algorithm. The performance gap is clearly shown with a sparser model.

With the random and gradient pruning algorithms, one or more output units can be isolated. When the output separation problem arises, classification performance drops drastically, as shown in Fig. 7. To gain insight into the performance drop, Table IV compares the number of isolated output units between pruning methods with various network architectures

TABLE IV
COMPARISON OF THE NUMBER OF ISOLATED OUTPUTS BETWEEN PRUNING ALGORITHMS IN THE MNIST DATA SET

# of Hidden Units	Gradient	Random	LRP	OSeA
[32]	{2, 6, 7, 9}	{0, 0, 3, 2}	N	N
[64]	{4, 5, 8, 9}	{0, 0, 0, 5}	N	N
[128]	{2, 6, 8, 10}	{0, 0, 0, 3}	N	N
[256]	{0, 1, 3, 8}	{ 0, 0, 0, 4}	N	N
[64, 32]	{6, 7, 9, 9}	{0, 0, 1, 5}	N	N
[128, 64]	{2, 3, 5, 9}	{0, 0, 0, 4}	N	N
[256, 64]	{2, 4, 6, 7}	{0, 0, 1, 3}	N	N
[256, 128]	{1, 3, 5, 8 }	{0, 0, 0, 5}	N	N

when the pruning rate is set to $\{0.8, 0.9, 0.95, p_{\max}\}$. N denotes no output separation with all considered pruning rates. Otherwise, we provide the number of isolated output units for corresponding pruning rates from 0.8 to $p_{\max}$. For example, when 64 and 32 hidden units are used at the first and second hidden layers, there are two separate output units when p_{prune} is 0.95 and 0.969 in random pruning. There are also some isolated output units in traditional gradient and random pruning schemes. Since more connections are kept, the number of isolated units tends to decrease when a more complex classification model is used.

C. Evaluation of Pruning-Based Model on Intrusion Detection

First, we present learning curves of the COOL-based detection model on both training and validation sets with a pruning rate of 0.8 in Fig. 8. The number of hidden units is set to 10. There are two phases of parameter learning: 1) training the FC model and 2) retraining the COOL-based model after trimming unimportant connections. In the first phase, all network parameters are learned from the scratch by minimizing the loss function. Then, we eliminate 80% of the least significant connections and retrain the remaining parameters. Thus, in both phases, the loss curve gradually decreases while the accuracy improves over epochs. We stop network training when there is no improvement in classification accuracy on the validation set for the last 20 epochs.

After parameter training, the performance of the COOL-based model is considerably lower than that of the FC model because the number of connections in the COOL-based model is only one-fifth of that in the FC architecture, which greatly affects traffic classification performance. Therefore, the classification accuracy of the pruned model (around 90%) is roughly 4.3% lower than that of the FC model (around 94.3%).

Tables V and VI show the performance of the proposed architecture with various parameters consisting of the number of hidden units n_1, pruning rate p_{prune}, and the number of bits χ representing the input and model parameters. Specifically, the number of hidden units varies from 5 to 15, the pruning rate list is $\{0.2, 0.4, 0.6, 0.8, 0.9\}$, the χ value is set to $\{6, 10, 16\}$ bits. Bold numbers indicate the best-measured performance. As can be seen in Table V, the performance improves as n_1 increases and p_{prune} decreases. However, the performance improvement is more evident when n_1 changes from 5 to 10 (e.g., classification accuracy gains nearly 1% with $p_{\text{prune}} = 0.2$) rather than 10 to 15 (e.g., accuracy enhances around 0.5% with $p_{\text{prune}} = 0.2$). Since more hidden units result in higher model complexity, $n_1 = 10$ is selected as the default value to reflect the tradeoff between model complexity and classification accuracy.

Regarding the detection delay, Fig. 9 shows the effects of the pruning rate with different values of n_1. Clearly, the detection delay increases when more hidden units and a low value

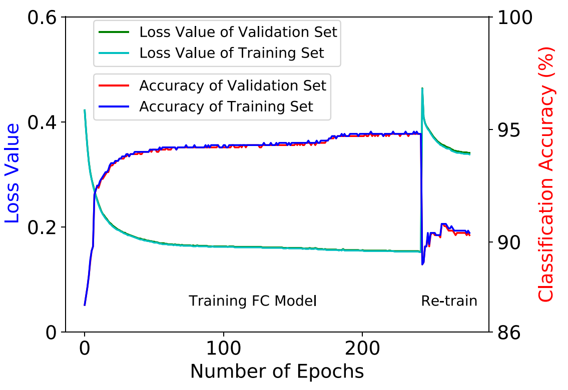


Fig. 8. Learning curves of the COOL-based intrusion detection model with $p_{\text{prune}} = 0.8$.

TABLE V
IMPACT OF n_1 AND p_{prune} ON PERFORMANCE OF THE PROPOSED COOL-BASED MODEL

n_1	p_{prune}	χ(bits)	Accuracy(%)	FP(%)	FN(%)	FI(%)
5	0.2	10	93.12	3.215	12.088	0.028
5	0.4	10	92.91	2.512	13.341	0.28
5	0.6	10	91.22	5.532	13.150	0.267
5	0.8	10	91.56	3.841	15.014	0.007
5	0.9	10	87.03	8.754	13.416	5.585
10	0.2	10	94.02	3.139	9.666	0.363
10	0.4	10	94.14	2.325	10.700	0.221
10	0.6	10	93.25	2.448	12.832	0.077
10	0.8	10	92.44	2.446	14.242	0.633
10	0.9	10	88.97	9.098	13.349	0.454
15	0.2	10	**94.51**	2.448	9.744	0.094
15	0.4	10	94.48	2.606	**9.662**	0.013
15	0.6	10	93.82	2.665	10.947	0.262
15	0.8	10	92.93	**1.655**	14.801	**0.003**
15	0.9	10	91.04	5.422	13.392	0.629

TABLE VI
IMPACT OF χ ON PERFORMANCE OF THE PROPOSED COOL-BASED MODEL

n_1	p_{prune}	χ(bits)	Accuracy(%)	FP(%)	FN(%)	FI(%)
10	0.2	6	82.73	21.216	8.124	3.498
10	0.4	6	83.64	18.608	12.114	1.038
10	0.6	6	83.75	19.147	8.733	3.380
10	0.8	6	81.41	22.774	9.159	3.442
10	0.9	6	72.67	34.326	**7.838**	9.501
10	0.2	10	94.02	3.139	9.666	0.363
10	0.4	10	94.14	2.325	10.700	0.221
10	0.6	10	93.25	2.448	12.832	0.077
10	0.8	10	92.44	2.446	14.242	0.633
10	0.9	10	88.97	9.098	13.349	0.454
10	0.2	16	**95.08**	1.523	9.650	0.136
10	0.4	16	94.51	**0.995**	11.909	**0.015**
10	0.6	16	94.15	1.936	11.335	0.118
10	0.8	16	93.00	1.618	14.593	0.097
10	0.9	16	92.03	3.055	14.929	0.072

TABLE VII
PERFORMANCE COMPARISON BETWEEN THE PROPOSED COOL-BASED MODEL AND EXISTING MODELS

Model	**Accuracy (%)**	**FP (%)**	**FN (%)**	**FI (%)**	**Detection delay (ms)**	**Memory (kB)**
COOL-based w/ $p_{prune} = 0.2$	94.02	3.139	9.666	0.363	1.174	3.125
COOL-based w/ $p_{prune} = 0.4$	94.14	2.325	10.700	0.221	1.163	2.344
COOL-based w/ $p_{prune} = 0.6$	93.25	2.448	12.832	0.077	1.023	1.562
COOL-based w/ $p_{prune} = 0.8$	92.44	2.446	14.242	0.633	0.86	0.781
COOL-based w/ $p_{prune} = 0.9$	88.97	9.098	13.349	0.454	0.754	0.391
NB-based model	45.71	84.87	**0.67**	9.93	**0.456**	0.469
SVM-based model	83.92	**0.11**	38.92	**0**	0.62	1.094
DT-based model	91.05	3.84	15.41	0.81	0.476	**0.062**
FC w/o hidden layer	88.68	4.49	22.13	0.40	0.785	1.094
FC w/ one hidden layer	**94.17**	1.93	10.69	0.21	1.245	3.906
FC w/ two hidden layers	**95.22**	1.72	9.09	0.037	1.959	7.344

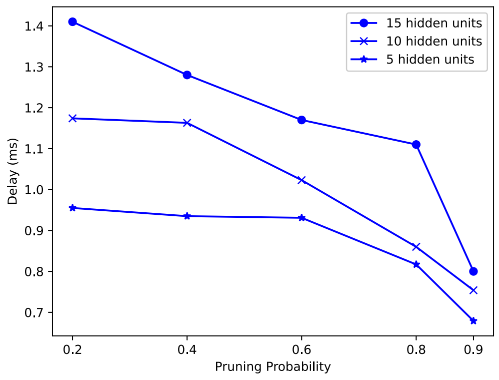


Fig. 9. Effects of pruning rates on detection delay.

of pruning rate are used, as more operations need to be executed. More specifically, the number of main operations (i.e., addition and multiplication) of the COOL-based model with p_{prune} is $2(n_0 n_1 + n_1 n_2)(1 - p_{\text{prune}})$ with one hidden layer. Since the detection time is proportional to the number of main operations, the delay increases as n_1 increases and p_{prune} decreases. Note that the relationship between detection time and the number of main operations is not linear because some operations exist regardless of the values of n_1 and p_{prune} in the COOL-based model. For example, after computing output features, we need to compare the values of output units to find the traffic label with the highest output value.

Table VI illustrates the impact of χ on the performance of the COOL-based architecture. There is a significant performance gain when χ changes from 6 to 10 bits compared to a much slower improvement when χ increases from 10 to 16 bits. The resulting improvement when using more bits to store model parameters can be clearly seen in all performance metrics. Note that when χ changes from 6 to 10 bits, the memory required to store parameters increases by a factor of 10/6 and performance gain can be achieved at a relatively high rate (e.g., accuracy improves by 16.3%, FP drops by 25.2%, and FI drops by 9% with $p_{\text{prune}} = 0.9$). Therefore, we select $\chi = 10$ bits as the default value for our other experiments.

Now, we examine the impacts of p_{prune} and n_1 on the average detection time of a packet for the proposed model and compare its performance results with those of existing models with different architectures (NB [50], SVM, DT [40], and FC [51]) in Table VII. The detection time is collected from a simulated programmable switch that runs on the desktop PC with Intel Core i7 2.5GHz CPU (with the Radeon R9 M370X 2048 MB and Intel Iris Pro 1536 MB GPU support) and 16 GB RAM. Note that the FC layer is a type of 1-D CNN layer when the kernel size is set to the number of input features. More advanced deep learning models like recurrent NNs and complex CNNs are not suitable to implement at programmable switches. The detection delay is recorded five times, and the average value is presented in this article. Among the verified models, the NB-based scheme has the lowest classification accuracy of 45.71% because it uses only the source and destination IP addresses as input features for detecting the attack label although a single host can generate multiple different attack classes. Therefore, the NB-based model has the lowest classification performance (i.e., accuracy, FP, and FI). Among the evaluated models, the DT-based scheme produces the second shortest delay (i.e., 0.456 ms) while attaining a relatively high accuracy of 91.05%. Our COOL-based models with p_{prune} less than 0.9 are more accurate than three other existing algorithms (SVM, DT, and FC w/o hidden layer) thanks to the nonlinear and sufficient connections from the input features to the output values. When $p_{\text{prune}} = 0.9$, since we remove 90% of connections compared to FC with one hidden layer, classification performance drops considerably. Despite achieving the highest classification accuracy (95.22%), the FC model with two hidden layers has the highest memory footprint for network parameters, including weights and biases. Meanwhile, the COOL-based model can reduce the memory footprint by around $p_{\text{prune}} * 100\%$ compared to the FC model with one hidden layer. The COOL-based model achieves a balance between performance metrics (i.e., classification accuracy and detection delay). Recall that we designed the COOL-based model such that less important connections are pruned from the FC model. Since fewer operations are required to classify the network traffic, the detection time can be reduced. However, we need to sacrifice classification performance, especially when p_{prune} becomes large. If p_{prune} is well selected, we can achieve a much shorter detection delay without significantly sacrificing classification performance. For example, when $p_{\text{prune}} = 0.6$, classification accuracy drops less than 1%, while detection delay improved around 20% from 1.245 to 1.023 (ms).

To show the multiclass performance of the proposed detection model, we present the confusion matrix of the architecture with two hidden layers of ten ReLU units in

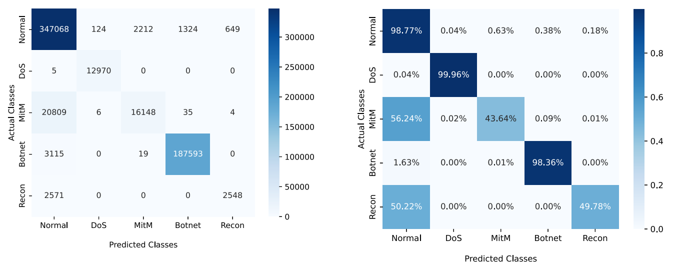


Fig. 10. Confusion matrix (left) and normalized confusion matrix (right) of the COOL-based detection model on the IoT data set.

Fig. 10. Three categories (normal, DoS, and MitM) achieve high accuracy, while performance for Botnet and reconnaissance is less than 50% because we used features that excel at recognizing normal, DoS, and MitM classes. Specifically, features consist of the linear sum, mean, and variance of packet length and time elapsed between two consecutive packets. The average packet length of DDoS attacks is usually higher than that of normal packets, while the average elapsed time of MitM is usually higher than that of normal traffic. To improve the accuracy of Botnet and reconnaissance categories, a new feature should be added. For example, traffic with requests to know available services and traffic volume from multiple sources to a given destination can be used to recognize reconnaissance and Botnet, respectively.

D. Optimization of Detection Time

If we assume that there are 30 switches in a network with a mesh topology and each switch is randomly connected to three other switches, when an event is generated, a 2-MB packet is sent to the server via networking switches. OR-Tools is used to solve the optimization problem in two cases: 1) with (w/) and 2) without (w/o) the COOL-based model. In the first case, each switch can choose appropriate classification models from the eleven models in Table VII. Meanwhile, the second case consists of only five possible choices: 1) NB-based; 2) SVM-based; 3) DT-based; 4) FC without a hidden layer; and 5) FC with one hidden layer. We consider these two cases to examine how the COOL-based model can improve the detection time of NIDS. Various parameters of the optimization problem are evaluated, including accuracy requirement a_{req}, transmission cost constraint c_{req}, completion ratio requirement r_{req}, and network speed. The one-hop delay T_{hop} is set to 1 ms.

Fig. 11 presents the average detection delay of classified packets in the network when the network speed changes from 1 to 3 GB/s (step size is 0.2), the completion ratio increases from 0.5 to 1 (step size is 0.1), the accuracy requirement is set to $\{90, 93, 94\}$, and the transmission cost constraint is 1000 packets. For each accuracy requirement value, there are a total of 66 combination sets between network speed and completion ratio. We find the optimal solution for the optimization problem and present the average detection delay in Fig. 11. Note that the white boxes represent cases with no feasible solution. As shown in Fig. 11, there are fewer sets with feasible solutions as a_{req} increases because fewer classification models can satisfy the higher accuracy requirement.

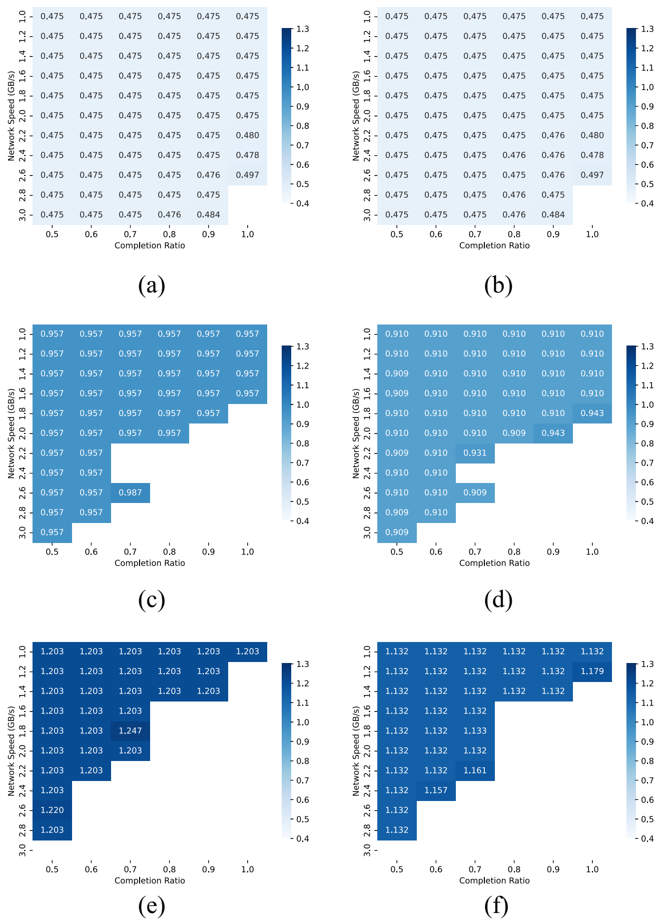


Fig. 11. Impact of accuracy requirement on average detection time. (a) w/o COOL, $a_{\text{req}} = 90\%$, (b) w/ COOL, $a_{\text{req}} = 90\%$, (c) w/o COOL, $a_{\text{req}} = 93\%$, (d) w/ COOL, $a_{\text{req}} = 93\%$, (e) w/o COOL, $a_{\text{req}} = 94\%$, (f) w/ COOL, $a_{\text{req}} = 94\%$.

For example, when $a_{\text{req}} = 90\%$, 64 out of 66 sets have feasible solutions compared to only 30 sets when $a_{\text{req}} = 94\%$ when the COOL-based models are not used. Moreover, generally, the number of feasible sets is inversely proportional to the network speed and completion ratio. For instance, with $a_{\text{req}} = 94\%$, the number of feasible sets increases from 0 to 5 when the network speed decreases from 3.0 to 1.0 GB/s. In addition, the COOL-based models can help to increase the number of feasible sets; for example, with $a_{\text{req}} = 94\%$, there are 33 feasible sets when using COOL-based models compared to 30 sets without COOL-based schemes. Fig. 11 suggests that it takes less time to classify incoming traffic as the network speed or completion ratio decreases. Since NIDS deals with fewer packets, the average detection time can be reduced.

For each completion ratio value c_{req}, there exists a specific network speed value called upper bound network speed value s_{upper}. If the network traffic has a higher speed than s_{upper}, there is no feasible solution. For instance, when accuracy requirement is set to 94%, $s_{\text{upper}} = 2.8$ GB/s with $c_{\text{req}} = 0.5$ and $s_{\text{upper}} = 1.4$ GB/s with $c_{\text{req}} = 1$. Similarly, for each network speed value, we observe an upper bound value of completion ratio c_{upper} such that there exists a feasible solution if the completion ratio is smaller than c_{upper}. For example, when $a_{\text{req}} = 94\%$, $c_{\text{upper}} = 0.5$ with network speed 2.8 GB/s

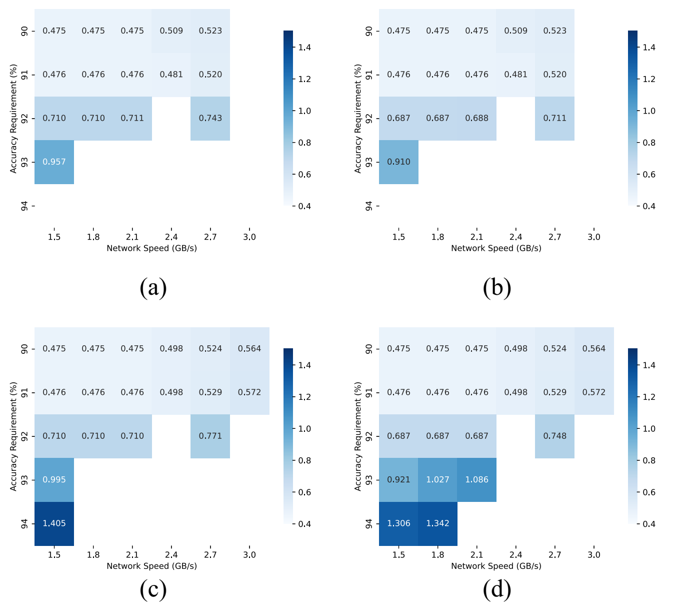


Fig. 12. Impact of transmission cost constraint on average detection time (ms). (a) w/o COOL, $c_{\text{req}} = 2000$ (b) w/ COOL, $c_{\text{req}} = 2000$ (c) w/o COOL, $c_{\text{req}} = 6000$ (d) w/ COOL, $c_{\text{req}} = 6000$.

compared to $c_{\text{upper}} = 0.7$ when the network speed drops to 2.0 GB/s.

When using the upper bound network speed value s_{upper}, we record an optimal solution for the optimization problem. Assuming that only the speed value is reduced and other parameters are kept constant, the stored solution can be used, and the average detection delay is almost similar to the case of s_{upper}. Therefore, Fig. 11 shows that the average delay when using the COOL-based model is around 0.475, 0.910, and 1.132 (ms) with a_{req} =90%, 93%, and 94%, respectively.

To further investigate the effects of the transmission cost constraint on the detection system, we measure the average detection delay when limiting the number of offloaded packets to 2000 and 6000, as shown in Fig. 12. The network speed changes from 1.5 to 3.0 GB/s with a step size of 0.3, while the accuracy requirement increases from 90 to 94% with a step size of 1. For this experiment, there are a total of 30 sets. If the transmission cost constraint increases, there are more cases with feasible solutions because neighboring switches can help to classify more network traffic. For example, there are 15 out of 30 feasible sets with $c_{\text{req}} = 2000$ compared to 21 feasible sets with $c_{\text{req}} = 6000$ if using the COOL scheme. However, if offloading packets to neighboring switches, it takes more time to classify all incoming packets.

Table VIII compares the average detection delay in two cases (i.e., with and without the proposed model) to show the effectiveness of the COOL-based scheme with accuracy requirement changes from 90% to 95%. The network speed is set to 1 GB/s, the completion ratio is 1, and the transmission cost constraint is 1000. When a_{req} is less than 91.5%, the average delay is similar in the two cases because the DT-based model with the second shortest delay of 0.476 ms is used to classify the incoming packets. Recall that the DT-based model achieves 91.5% accuracy. When the accuracy requirement is greater than or equal to 94.5%, there is no feasible solution in either case, as the highest accuracy that can be achieved by classification models on the IoT data set is 94.17% (FC with one hidden layer). The COOL-based models can shorten the average classification delay, especially with a high-accuracy requirement value. The improvement gain gradually increases with a_{req} because the COOL-based model can yield competitive classification accuracy with a shorter detection delay than the FC scheme. Therefore, when a_{req} is larger, the COOL-based algorithm is used more frequently, which helps to reduce the average detection time. For example, when a_{req} is 94%, the delay can be shortened by 6% from 1.204 to 1.132 (ms) using the COOL-based model.

TABLE VIII
DETECTION TIME WITH AND WITHOUT COOL

Accuracy constraint (%)	w/o COOL (ms)	w/ COOL (ms)
90.00	0.475	0.475
90.50	0.476	0.476
91.00	0.476	0.476
91.50	0.587	0.576
92.00	0.710	0.687
92.50	0.833	0.798
93.00	0.957	0.910
93.50	1.080	1.021
94.00	1.204	1.132
94.50	NaN	NaN
95.0	NaN	NaN

VII. CONCLUSION

In a pruning-based NIDS, synaptic connections to output neurons may be lost, especially in a highly sparse model, significantly reducing detection performance. To address this problem, we design the COOL algorithm to conserve the most significant connections for nonisolated neurons in the pruned model. Thanks to the COOL, the COOL algorithm outperforms other weight-pruning methods in terms of classification accuracy on various benchmark data sets and significantly reduces detection delay compared to fully dense networks. To further evaluate the COOL-based NIDS, we formulate an optimization problem with the objective of minimizing the average detection delay of traffic in the network under constraints on overall performance and available resources. Our experimental results show that more traffic can be managed with a shorter detection delay in the intrusion detection system with the support of the COOL algorithm. With the aim of further reducing detection delay, we plan to embed the COOL algorithm in the distributed detection architecture in future works. New features should be added to the classification model to improve multiclass performance. In addition, the proposed COOL-based NIDS should be evaluated in real-world environments to determine how it handles real traffic.

REFERENCES

[1] J. Qi, P. Yang, G. Min, O. Amft, F. Dong, and L. Xu, "Advanced Internet of Things for personalised healthcare systems: A survey," *Pervasive Mobile Comput.*, vol. 41, pp. 132–149, Oct. 2017.

[2] D. Glaroudis, A. Iossifides, and P. Chatzimisios, "Survey, comparison and research challenges of IoT application protocols for smart farming," *Comput. Netw.*, vol. 168, Feb. 2020, Art. no. 107037.

[3] R. Li, T. Song, N. Capurso, J. Yu, J. Couture, and X. Cheng, "IoT applications on secure smart shopping system," *IEEE Internet Things J.*, vol. 4, no. 6, pp. 1945–1954, Dec. 2017.

[4] L. D. Xu, W. He, and S. Li, "Internet of Things in industries: A survey," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.

[5] K. Y. Najmi, M. A. AlZain, M. Masud, N. Z. Jhanjhi, J. Al-Amri, and M. Baz, "A survey on security threats and countermeasures in IoT to achieve users confidentiality and reliability," *Mater. Today Proc.*, vol. 81, pp. 377–382, May 2023.

[6] X. Liang and Y. Kim, "A survey on security attacks and solutions in the IoT network," in *Proc. IEEE 11th Annu. Comput. Commun. Workshop Conf. (CCWC)*, 2021, pp. 853–859.

[7] J. R. Vacca, *Computer and Information Security Handbook*. Amsterdam, The Netherlands: Newnes, 2012.

[8] F. Erlacher and F. Dressler, "On high-speed flow-based intrusion detection using snort-compatible signatures," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 1, pp. 495–506, Jan./Feb. 2022.

[9] M. F. Umer, M. Sher, and Y. Bi, "Flow-based intrusion detection: Techniques and challenges," *Comput. Security*, vol. 70, pp. 238–254, Sep. 2017.

[10] A. Ferdowsi and W. Saad, "Generative adversarial networks for distributed intrusion detection in the Internet of Things," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2019, pp. 1–6.

[11] R. Kumar, P. Kumar, R. Tripathi, G. P. Gupta, S. Garg, and M. M. Hassan, "A distributed intrusion detection system to detect DDoS attacks in blockchain-enabled IoT network," *J. Parallel Distrib. Comput.*, vol. 164, pp. 55–68, Jun. 2022.

[12] T.-N. Dao and H. J. Lee, "Stacked autoencoder-based probabilistic feature extraction for on-device network intrusion detection," *IEEE Internet Things J.*, vol. 9, no. 16, pp. 14438–14451, Aug. 2022.

[13] Y. Wang, W. Meng, W. Li, Z. Liu, Y. Liu, and H. Xue, "Adaptive machine learning-based alarm reduction via edge computing for distributed intrusion detection systems," *Concurrency Comput. Pract. Exp.*, vol. 31, no. 19, 2019, Art. no. e5101.

[14] P. S. Chandakkar, Y. Li, P. L. K. Ding, and B. Li, "Strategies for re-training a pruned neural network in an edge computing paradigm," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, 2017, pp. 244–247.

[15] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[16] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," 2016, *arXiv:1611.06440*.

[17] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems*, vol. 5, S. J. Hanson, J. D. Cowan, and C. L. Giles, Eds. Burlington, MA, USA: Morgan-Kaufmann, 1993, pp. 164–171.

[18] R. Yu et al., "NISP: Pruning networks using neuron importance score propagation," 2017, *arXiv:1711.05908*.

[19] J.-H. Luo and J. Wu, "An entropy-based pruning method for CNN compression," 2017, *arXiv:1706.05791*.

[20] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," 2016, *arXiv:1607.03250*.

[21] S. Liu et al., "The unreasonable effectiveness of random pruning: Return of the most naive baseline for sparse training," in *Proc. 10th Int. Conf. Learn. Represent.*, 2022, pp. 1–22.

[22] S.-K. Yeom et al., "Pruning by explaining: A novel criterion for deep neural network pruning," *Pattern Recognit.*, vol. 115, Jul. 2021, Art. no. 107899.

[23] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets," 2016, *arXiv:1608.08710*.

[24] H. Wang, Q. Zhang, Y. Wang, and H. Hu, "Structured probabilistic pruning for convolutional neural network acceleration," 2017, *arXiv:1709.06994*.

[25] S. Wiedemann, K.-R. Müller, and W. Samek, "Compact and computationally efficient representation of deep neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 3, pp. 772–785, Mar. 2020.

[26] K. Shirahata, Y. Tomita, and A. Ike, "Memory reduction method for deep neural network training," in *Proc. IEEE 26th Int. Workshop Mach. Learn. Signal Process. (MLSP)*, 2016, pp. 1–6.

[27] Y. Pisarchyk and J. Lee, "Efficient memory management for deep neural net inference," 2020, *arXiv:2001.03288*.

[28] M. Rusci, L. Cavigelli, and L. Benini, "Design automation for binarized neural networks: A quantum leap opportunity?" in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2018, pp. 1–5.

[29] T. Simons and D.-J. Lee, "A review of binarized neural networks," *Electronics*, vol. 8, no. 6, p. 661, 2019.

[30] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognit.*, vol. 105, Sep. 2020, Art. no. 107281.

[31] W. Li, X. Wang, H. Han, and J. Qiao, "A PLS-based pruning algorithm for simplified long–short term memory neural network in time series prediction," *Knowl.-Based Syst.*, vol. 254, Oct. 2022, Art. no. 109608.

[32] S. Srinivas, A. Kuzmin, M. Nagel, M. van Baalen, A. Skliar, and T. Blankevoort, "Cyclical pruning for sparse neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 2762–2771.

[33] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 6377–6389.

[34] P. Bosshart et al., "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[35] M. Gupta et al., "Is complexity required for neural network pruning? A case study on global magnitude pruning," 2022, *arXiv:2209.14624*.

[36] K. Lee and J. Yim, "Hyperparameter optimization with neural network pruning," 2022, *arXiv:2205.08695*.

[37] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? Toward in-network classification," in *Proc. 18th ACM Workshop Hot Topics Netw.*, 2019, pp. 25–33.

[38] M. P. J. Kuranage, K. Piamrat, and S. Hamma, "Network traffic classification using machine learning for software defined networks," in *Proc. Int. Conf. Mach. Learn. Netw.*, 2019, pp. 28–39.

[39] B. M. Xavier, R. S. Guimarães, G. Comarela, and M. Martinello, "Programmable switches for in-networking classification," in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.

[40] F. Musumeci, V. Ionata, F. Paolucci, F. Cugini, and M. Tornatore, "Machine-learning-assisted DDoS attack detection with P4 language," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2020, pp. 1–6.

[41] D. Ding, M. Savi, F. Pederzolli, M. Campanella, and D. Siracusa, "In-network volumetric DDoS victim identification using programmable commodity switches," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, pp. 1191–1202, Jun. 2021.

[42] B. Turkovic, J. Oostenbrink, and F. Kuipers, "Detecting heavy hitters in the data-plane," 2019, *arXiv:1902.06993*.

[43] K. Friday, E. Kfoury, E. Bou-Harb, and J. Crichigno, "Towards a unified in-network DDoS detection and mitigation strategy," in *Proc. 6th IEEE Conf. Netw. Softwarization (NetSoft)*, 2020, pp. 218–226.

[44] M. Lei, X. Li, B. Cai, Y. Li, L. Liu, and W. Kong, "P-DNN: An effective intrusion detection method based on pruning deep neural network," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2020, pp. 1–9.

[45] H. Kang, D. H. Ahn, G. M. Lee, J. Do Yoo, K. Ho Park, and H. K. Kim, Sep. 2019, "IoT network intrusion dataset," IEEE Dataport. [Online]. Available: https://dx.doi.org/10.21227/q70p-q449

[46] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi, "P4 edge node enabling stateful traffic engineering and cyber security," *J. Opt. Commun. Netw.*, vol. 11, no. 1, pp. A84–A95, 2019.

[47] A. Agrawal and C. Kim, "Intel Tofino2—A 12.9 Tbps P4-programmable Ethernet switch," in *Proc. IEEE Hot Chips 32 Symp. (HCS)*, 2020, pp. 1–32.

[48] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proc. Mil. Commun. Inf. Syst. Conf. (MilCIS)*, 2015, pp. 1–6.

[49] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. 9th ACM SIGCOMM Workshop Hot Topics Netw.*, 2010, pp. 1–6.

[50] G. K. Ndonda and R. Sadre, "A two-level intrusion detection system for industrial control system networks using P4," in *Proc. 5th Int. Symp. ICS SCADA Cyber Security Res. (ICS-CSR)*, 2018, pp. 1–10.

[51] J. Shun and H. A. Malki, "Network intrusion detection system using neural networks," in *Proc. 4th Int. Conf. Nat. Comput.*, vol. 5, 2008, pp. 242–246.

Thi-Nga Dao received the B.S. degree in electrical and communication engineering from Le Quy Don Technical University, Hanoi, Vietnam, in 2013, and the M.S. degree in computer engineering and the Ph.D. degree in computer engineering from the University of Ulsan, Ulsan, South Korea, in 2016 and 2019, respectively.

Since July 2019, she had been a Lecturer with the Faculty of Radio-Electronic Engineering, Le Quy Don Technical University. She is currently a Postdoctoral Fellow with the Computer Science and Engineering Department, Ewha Womans University, Seoul, South Korea. Her research interests include machine learning-based applications in network security, network intrusion detection and prevention systems, human mobility prediction, and mobile crowdsensing.

HyungJune Lee (Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, in 2001, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, USA, in 2006 and 2010, respectively.

He joined Broadcom, San Jose, CA, USA, as a Senior Staff Scientist for working on research and development of 60-GHz 802.11ad SoC MAC. Also, he worked for AT&T Labs, Atlanta, GA, USA, as a Principal Member of Technical Staff with the involvement of LTE overload estimation, LTE-WiFi interworking, and heterogeneous networks. He is currently a Professor with the Computer Science and Engineering Department, Ewha Womans University, Seoul. His current research interests include distributed learning and future wireless networks on the IoT, fog computing, VANET, and machine learning-driven network system design.