Proteus: A Difficulty-aware Deep Learning Framework for Real-time Malicious Traffic Detection

Chupeng Cui[1,2], Qing Li[2], Guorui Xie[1,2], Ruoyu Li[1,2], Dan Zhao[2], Zhenhui Yuan[3], Yong Jiang[1,2]
[1]International Graduate School, Tsinghua University, Shenzhen, China
[2]Peng Cheng Laboratory, Shenzhen, China
[3]Department of Computer and Information Science, Northumbria University, Newcastle, United Kingdom

2024 IEEE 32nd International Conference on Network Protocols (ICNP) | 979-8-3503-5171-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICNP61940.2024.10858520

***Abstract*—Deep learning (DL) has been recently used for malicious traffic detection. However, DL models are often faced with a dilemma between model size and performance: larger models have better accuracy, but suffer from high detection latency, which severely impacts realtime traffic performance, while lightweight models have low detection latencies, but sacrifice accuracy. In this paper, we introduce Proteus, a swift and precise attack detection framework that adaptively adjusts DL models in real-time based on sample detection difficulty. To address diverse detection difficulties in traffic data, we devise a Double Dynamic Convolutional Neural Network (DDCN) with two pivotal modules: the Dynamic Feature Campaign (DFC) and the Tailor Module (TM). DFC enables the model to discern and accentuate the most influential features, while TM autonomously gauges sample difficulty, cropping the overall model. We further design an auxiliary detection module to streamline the detection, especially for network devices like routers lacking GPUs but equipped with multiple CPU cores. Experiments on different network devices show that Proteus completes the detection of each flow within 0.6ms, and achieves 99.34% detection accuracy, outperforming other solutions.**

***Index Terms*—Machine learning, malicious web traffic detection, low latency, security.**

I. Introduction

Given the drastic increase in cyber threats and malicious attacks across various domains such as IT, finance, energy, and health [1], malicious traffic detection has become crucial for protecting diverse networks including LAN, WAN, wireless and cloud networks. By 2023, the National Vulnerability Database (NVD)[1] reported 30,947 new network vulnerabilities, marking a 14.59% increase from 2022. Traditionally, rule-based detection methods [2] and shallow machine learning methods [3]–[6] have been favored for their interpretability and simplicity [7]. Yet, as new applications emerge and diversify network traffic, these methods often falter, and their performance can be constrained (refer to Section VI-D1).

Recently, deep learning (DL) has excelled in fields like computer vision [8], [9] and natural language processing [10], [11]. With the continuous emergence of new applications (e.g., smart housing and e-health [12]–[15]), the mixed traffic of various applications increases the complexity of traffic detection [16], and researchers in the field of network security are also trying to use deep learning methods to solve NID (Network Intrusion Detection) problems [17]–[23]. For example, the author of [22] proposed a DL based multi-level feature fusion model MFFusion, and demonstrates excellent performance on IoT23 Dataset [23]. Similarly, the authors of [20] designed a DNN structure using a series of fully connected layers and its performance surpasses numerous shallow machine learning methods on different datasets. Although deep learning provides higher accuracy than shallow ML methods and traditional rule-based methods, and shows great potential in NID, it brings computational and latency challenges [24], [25]. Common network devices such as routers are often targeted for NID. However, due to cost constraints, they are not equipped with GPUs. Since DL models require millions of FLOPs for computation, the absence of GPUs or other acceleration devices may introduce significant delays (hundreds or thousands of milliseconds), affecting real-time NID performance (such as data centers with throughput up to 100Gbps) [25].

Lightweight neural networks introduced in recent years [26], [27] employ efficient techniques [9], [28] to reduce the number of parameters and computational demands. These optimizations lead to faster inference time, making it possible to deploy such models on mobile and embedded devices. Notable examples include MobileNetV2 [29] and ShuffleNetV2 [30]. However, even these streamlined models can impose significant computational burdens on network devices like routers, which have more limited processing capabilities and are more sensitive to latency than typical mobile devices. Meanwhile, some studies have proposed multi-exit neural networks, such as BranchyNet [31] and HDKD [32], which significantly accelerate average inference speeds while maintaining classification accuracy by incorporating early exit branches. However, these methods are often designed for models with many layers (e.g., BERT [33] with 12 layers), which typically exhibit higher computational complexity and require acceleration devices like

Corresponding Authors: Qing Li (liq@pcl.ac.cn), Yong Jiang (jiangy@sz.tsinghua.edu.cn)

[1]https://nvd.nist.gov/

979-8-3503-5171-2/24/$31.00 ©2024 IEEE

GPUs. As a result, they may not be suitable for performing traffic detection tasks on network devices.

To cater to latency-sensitive and resource-constrained network devices, researchers have tailored a few approaches. Kitsune [24] designed a real-time attack detection system using a series of lightweight ANNs as autoencoders. Although Kitsune can be deployed on network devices, it can only perform binary classification tasks (i.e., milicious or not), whereas multi-class classifications of each known malicious type are often desirable as they can provide administrators with richer information to make better network management decisions. BCN [25] offers an automated solution for malicious detection using "branch convolution" and significantly reduces floating point operations (FLOPs). While BCN can perform multi-class classification tasks within 3ms, its inference accuracy is compromised compared to traditional DL models (refer to Section VI-D2).

In summary, existing DL-based NID methods face a trade-off: they can achieve high-precision detection with larger-scale models, but this comes at the cost of increased inference latency. Alternatively, they can reduce inference latency by employing a series of delicately designed lightweight models, but this often sacrifices detection performance. Thus, creating a DL solution that *ensures both high accuracy and low latency for network devices remains a challenge.*

MFFusion [22], BCN [25], and various NID methods typically detect all traffic by deploying a unique model. Yet, our experiments reveal that different network traffic poses varying detection difficulty (refer to Section VI-C). Certain traffic can be detected using DL methods with minimal computation, considered as "easy samples". Conversely, "difficult samples" require more computational resources to be properly detected. Notably, most attack traffic is categorized as easy samples after feature extraction, allowing efficient detection with smaller, faster models. This prompts the question: *can we tailor models according to traffic complexity, so as to find the sweet spot between accuracy and latency?* One rudimentary approach would be to deploy multiple models of various sizes on a device simultaneously. However, this not only consumes more resources, but also requires maintenance for each model. In addition, a suitable allocator is needed to allocate different types of traffic (simple and difficult samples) to different models, resulting in additional resource consumption and latency.

In this paper, we propose Proteus[2], an adaptive sample difficulty-aware framework for efficient and precise malicious network attack detection. Proteus incorporates real-time tailoring of model size based on network traffic needs while simultaneously adapting detection difficulty for different traffic flows. Specifically, inspired by dynamic neural networks [34], [35], we design a *Double Dynamic Convolutional Neural Network* (DDCN), which consists of a *Dynamic Feature Campaign Module* (DFC) and a *Tailor Module* (TM). DFC can automatically perceive the importance of sample features without human intervention and emphasizes more to those features that contribute more to the detection. TM senses the detection difficulty of samples and tailor the overall model accordingly in real-time.

The main contributions of this paper are as follows:

- We design a Dynamic Feature Campaign (DFC) module to improve the accuracy of NID tasks. DFC allows features from different channels to compete with each other, and the features that contribute significantly to the detection will win. Thus, DFC makes the network pay more attention to these features during the subsequent detection process.
- We propose a Tailor Module (TM) that adapts to samples' difficulty and crops the model in real time. TM consists of a Difficulty Perception Module (DPM) and a Dynamic Cropping Module (DCM). Based on the difficulty of the current sample, DPM outputs different model specifications according to the difficulty. As a result of DCM, the channels and weights of the current model are automatically cropped accordingly, completing the dynamic switching of model scales.
- Given the multi-core nature of network devices, we introduce an auxiliary detection module that utilizes a multi-core mechanism to ensure timely packet processing and feature extraction, thereby further improving detection efficiency. The module also makes use of buffering agents to prevent synchronization conflicts.

We tested Proteus on a virtual machine, Raspberry Pi, and a Wi-Fi router. Results indicate that: **1)** DDCN has 97 times fewer parameters, 433 times fewer floating-point operations, and 12 times faster inference than traditional DL methods [17], yet retains a 99.23% accuracy. **2)** Proteus consistently performs well in five diverse network scenarios. As a result, detection times on standard network devices are less than 0.6ms at a flow rate of 50Kfps.

II. Background

A. DL-based Network Traffic Classifier

Computer networks and systems are vulnerable to both discovered and unknown attacks, including DDoS, phishing, SQL injection and more [36]. Due to the increasing complexity of malicious traffic detection tasks, many DL based methods have been proposed to improve the performance. In [37], the authors propose a detection model that represents network traffic events using the time series of TCP/IP packets for attack detection. In [38], two types of neural networks, CNN and Long Short-Term Memory (LSTM), are used to identify malicious domain names. FS-Net [39] applies Recurrent Neural Network (RNN) to encrypted traffic detection, learning representative features from the raw flows for classification. [40] proposes a Transformer-based traffic representation model called ET-BERT, which pre-trains deep contextualized datagram level representation from large scale unlabeled data. [41] constructs a Graph Neural Network called TFE-GNN, integrating header information and payload for detection.

[2]a Greek sea god capable of taking on any shape he wished.

Although DL-based scheme achieve better performance, the above methods introduce higher latency (in seconds) when running on network devices without GPUs, which affects overall network performance. Designing high-performance and real-time detection solutions based on DL for network devices with limited performance and resources remains a challenge.

B. Dynamic Neural Network

In the field of NID, traditional neural networks mainly follow a static inference paradigm [25], [37], [42], maintaining the structures unchanged after training, which limits the expression ability [43]. In contrast, dynamic neural networks [44]–[46] adjust their structures and parameters based on the input during inference, offering improved efficiency and adaptability. CondConv [45] improves model accuracy to a certain extent by introducing multiple parallel convolution kernels and dynamic aggregation functions to adaptively adjust the model structure. Building on CondConv, Dynamic Convolution [46] further improved by ensuring that the combined weights of convolution kernels in the same layer equal 1. This optimization not only reduces the parameters of the model, but also improves accuracy.

In summary, dynamic neural networks enhance the performance by allocating the most suitable structure based on the individual characteristics of the samples. This adaptability inherently leads to higher computational efficiency. Our proposed Proteus not only improves detection performance based on the idea of dynamic neural networks, but also utilizes the advantages of ”dynamic” to reduce overall computational overhead, thereby reducing latency.

III. Proteus Overview

The overall framework of Proteus is shown in Figure 1, which consists of two main processes: the training process and the detection process.

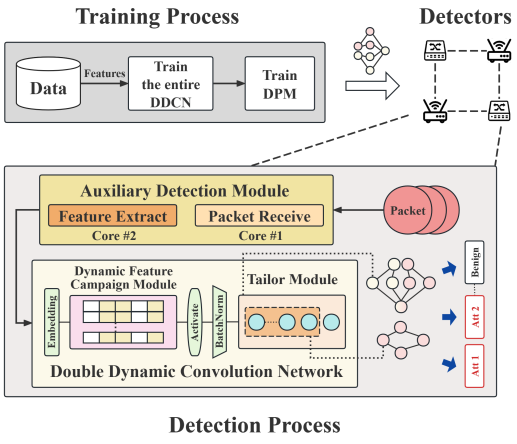


Fig. 1: The Proteus framework.

A. Training Process

In Proteus, we utilize a high-performance central controller with powerful computing power and the ability to store a large amount(e.g. 110G) of training data. Specifically, we extract flow-level features from data identified by 5-tuples and construct a dataset to train a Double Dynamic Convolutional Network (refer to Section IV-A). In order to further improve the accuracy of sample difficulty detection, after training the entire network, we separately trained the difficulty perception module (refer to Section IV-C1). The trained DDCN copy is sent to the corresponding network device for detection.

B. Detection Process

The detection process of Proteus takes place on the CPU of network devices. The Auxiliary Detection module (refer to Section 6) is designed to ensure timely feature extraction in order to avoid incomplete stream-level features. Specifically, we receive packets in core#1 and write them to the ring buffer, while core#2 extracts the desired features in each packet according to a predefined scheme. Finally, we send the collected features to the trained DDCN for detection. It should be noted that if the required features exist for a longer time than the threshold, they will be directly sent to the DDCN model for detection to avoid the flow queue running out of memory.

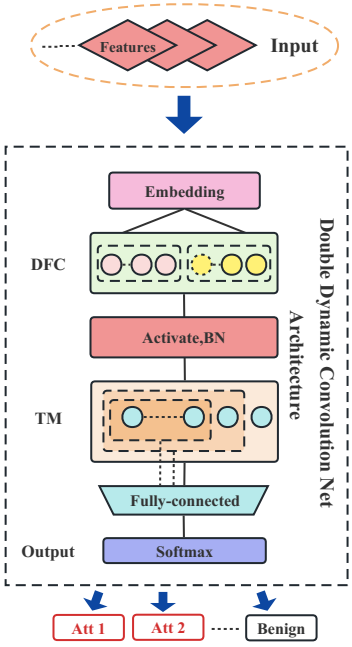


Fig. 2: The DDCN architecture.

IV. Double Dynamic Convolutional Neural Network

A. DDCN Architecture

DDCN adaptively senses the difficulty of sample detection and tailors the network structure in real-time, while focusing on features in different samples that contribute most to sample

detection. As such, it ensures low latency and achieving high accuracy. The architecture of the DDCN is shown in Figure 2. The input features of DDCN, denoted by $f \in \mathbb{R}^N$, are first fed into the embedding layer. The embedding layer expands the channels of the input features and enriches their representation [47]. After the embedding layer, we get an output $f' \in \mathbb{R}^{N \times C}$, where C is the number of channels expanded by the embedding layer. Next, f' is sent to DFC, where features with "high contributions" to the results will draw significant attention. Before entering TM, the representation is fed into the activation function (Activate), and the Batch Normalization (BN) layer in order. TM then perceives the current representation, evaluates its detection difficulty, and crops out the corresponding sub-network to compute a new representation through f'. Finally, this representation is passed on to the fully connected and the softmax layers.

Overall, with the help of DFC, DDCN guarantees detection accuracy while TM can reduce inference time. Combining the two, DDCN maintain low latency and high accuracy detection.

B. Dynamic Feature Campaign Module

Often, some features of an input sample play more decisive roles in the inference, while the rest are less influential. If the model can take into account the importance of the features, the overall performance can potentially be improved [48]. Additionally, decisive features may vary based on the sample, so adaptive methods are necessary to identify them.

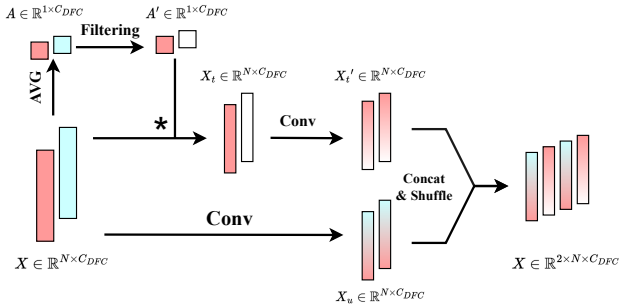


Fig. 3: The dynamic feature campaign module.

In contrast, classical convolution ignores the importance of features. In DL based NID schemes, the input to the model is usually a one-dimensional sequence, which is processed by traditional one-dimensional convolution as follows: let $x^{in} \in \mathbb{R}^{N \times C}$ be an input sample of length N with channel number C. The convolution kernel $k \in \mathbb{R}^{W \times C}$ filters a window of W inputs at a time and fuses the filtered results of the C channels to obtain the output $x^{out} \in \mathbb{R}^{N' \times M}$, where $N' = N - W + 1$. The process of one operation of a convolution kernel can be expressed in the form of the following equation:

$$x_s = \sum_{i=1}^{W} \sum_{j=1}^{C} k_{i,j} \times x_{s+i,j}^{in}, \tag{1}$$

where $s \in [1, N']$. When there are M kernels for processing, the final output is $x^{out} \in \mathbb{R}^{N' \times M}$, where $x^{out} = \{x^1, \ldots, x^M\}$ is the set of the M kernel outputs. During the above process, the parameters of the convolution kernel remain constant regardless of the input features, which may limit lightweight models from achieving higher detection accuracy.

We propose a new Dynamic Feature Campaign module (DFC) that automatically recognizes the importance of different sample features to overcome these limitations. As illustrated in Figure 3, the DFC processes the input features by two branches. The top branch retains features that contribute more to the result while filtering out the rest. Specifically, the features of different channels compete by comparing the averages of the features of different channels. We first calculate the mean of the features of each channel to obtain the mean vector $\mathbf{A} \in \mathbb{R}^{1 \times C_{DFC}}$. Then, we perform the filtering operation by ranking the features from highest to lowest based on their mean value. The features with lower rankings are considered "low contribution", while the rest are considered "high contribution". We use hyperparameter γ to control the proportion of "low contribution" features:

$$\gamma = \frac{|\{a \mid a \leq \tau, a \in \mathbf{A}\}|}{|\{a \mid a \in \mathbf{A}\}|}, \tag{2}$$

where τ denotes the threshold value, i.e., the percentage of the elements less than τ in $\mathbf{A}$ is γ. After that, we set elements in $\mathbf{A}$ that are smaller than τ to 0 to obtain $\mathbf{A}' \in \mathbb{R}^{1 \times C_{DFC}}$ and mask the original input features with $\mathbf{A}'$ so that the processed feature $X_t \in \mathbb{R}^{N \times C_{DFC}}$ will have $\gamma \times C_{DFC}$ channels with all 0 vectors. Finally, we perform a convolution operation on the X_t to obtain $X_t' \in \mathbb{R}^{N \times C_{DFC}}$, focusing only on more important features since the low contributing features have been set to 0.

The lower branch performs a convolution operation directly on the original input features to get $X_u \in \mathbb{R}^{N \times C_{DFC}}$. Even though some features are small contributors, ignoring them completely could result in insufficient information and loss of prediction accuracy.

Last, we stitch and shuffle the output results of both upper and lower branches to obtain the final output $X' \in \mathbb{R}^{2 \times N \times C_{DFC}}$. It should be noted that the purpose of shuffling is to fully integrate the features obtained from different branches, avoiding the independence of features from each other [49].

C. Tailor Module

To ensure low latency detection, our key observation is that traffic with different detection difficulties can be processed using models of different scales. Inspired by one-shot NAS methods [50]–[52], we propose the tailor module (TM), which detects and assigns samples to different classes (simple and difficult), and crops the model in real-time.

As shown in Figure 4, the TM consists of a Difficulty Perception Module (DPM) and a Dynamic Cropping Module (DCM). Specifically, assuming that the input is given as $\mathbf{X} \in \mathbb{R}^{N \times C_T}$, it first passes through DPM, which senses the difficulty level of the sample and provides an appropriate cropping rate $\rho \in [0, 1]$. After that, DCM crops the entire network

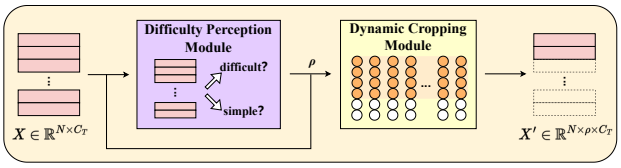


Fig. 4: The talior module.

structure according to the cropping rate ρ. The resulting data is given as $\mathbf{X}' \in \mathbb{R}^{N\times\rho\times C_T}$, i.e., the number of channels is reduced to ρ times the original number. Similarly, DCM simultaneously crops the corresponding convolution kernels.

1) Difficulty Perception Module (DPM): DPM calculates a suitable cropping rate ρ based on the current input sample. In a way, ρ indicates the difficulty level of distinguishing the sample. A larger ρ means the sample is more difficult to detection correctly, and vice versa. For simplicity, we give some preset values for ρ and transform the calculation of ρ into a classification problem.

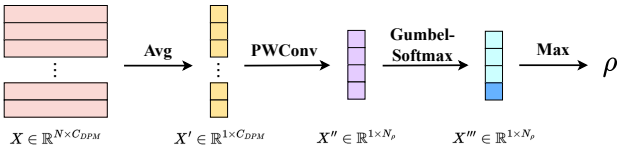


Fig. 5: The difficulty perception module.

As shown in Figure 5, given the input $\mathbf{X} \in \mathbb{R}^{N\times C_{DPM}}$, DPM first calculates the mean value of each channel to get $\mathbf{X}' \in \mathbb{R}^{1\times C_{DPM}}$. Then, it alters the number of channels of $\mathbf{X}'$ by pointwise convolution to obtain $\mathbf{X}'' \in \mathbb{R}^{1\times N_\rho}$, where N_ρ represents the number of preset values for ρ. After that, $\mathbf{X}''$ passes through a gumbel-softmax[3] layer to get $\mathbf{X}''' \in \mathbb{R}^{1\times N_\rho}$. Finally, the channel with the highest prediction probability is selected to obtain the cropping rate ρ. In practical applications, channel 0 represents the cropping rate of the model that requires the smallest size, that is, $\rho = 0.25$, channel 3 represents the full-scale model, which means $\rho = 1$. DPM tends to produce an output with no dynamics during training (i.e., cropping rates remains the same regardless of input). Inspired by [34], we introduce the Sandwich Gate Sparsification (SGS) technique to train the DPM separately after the overall network has been trained.

The input samples can be classified into three difficulty classes: i) easy samples $\mathbf{X_e}$, which can be correctly classified by the smallest sub-network, corresponding to $\rho = 0.25$; ii) super samples $\mathbf{X_s}$, even the largest network finds it difficult to classify them correctly, corresponding to $\rho = 1$; and iii) other samples $\mathbf{X_o}$, whose difficulty levels are between those of $\mathbf{X_e}$ and $\mathbf{X_s}$, corresponding to other cropping rates.

As analyzed by Yu et al [53], larger networks tend to be more accurate. To minimize inference time, $\mathbf{X_e}$ should always be routed to the smallest sub-network (i.e., $\mathbf{T}(\mathbf{X_e}) = \{1, 0, \ldots, 0\}$). For $\mathbf{X_s}$ and $\mathbf{X_o}$, we encourage them to pass through the largest sub-network, even if the $\mathbf{X_s}$ may not be classified correctly (i.e., $\mathbf{T}(\mathbf{X_s}) = \mathbf{T}(\mathbf{X_o}) = \{0, 0, \ldots, 1\}$). Based on this, we define the SGS loss as:

$$\begin{aligned} L_{SGS} =& \mathbb{T}_{smallest}(\mathbf{X}) \times L_{CE}(\mathbf{X}, \mathbf{T}(\mathbf{X_e})) + \\ & (\neg\mathbb{T}_{smallest}(\mathbf{X})) \times L_{CE}(\mathbf{X}, \mathbf{T}(\mathbf{X_s})), \end{aligned} \tag{3}$$

where $\mathbb{T}_{smallest}(\mathbf{X}) \in \{0, 1\}$ represents whether $\mathbf{X}$ is accurately detected by the smallest sub-network and L_{CE} is the cross-entropy loss calculated based on the softmax activated cropping rate scores and the generated target. It should be noted that we first use the largest sub network to detect $\mathbf{X_o}$, with the aim to ensure that the model accuracy does not significantly decrease. If a $\mathbf{X'_o}$ is correctly detected by the smallest model, that is, $\mathbb{T}_{smallest}(\mathbf{X'_o}) = 1$, according to the L_{SGS}, the scale of the model used in its training process will gradually decrease.

2) Dynamic Cropping Module (DCM): When the cropping rate ρ is obtained, DCM needs to crop the model accordingly. Most of the previous dynamic pruning methods [35], [54] change the channel sparsity according to the input. However, it is difficult for these methods to provide a practical speedup, even though they reduce the computational overhead in theory. This is because the altered channel sparsity is often hardware incompatible [34]. To achieve practical speedup, one should avoid sparsification of the convolution kernels, i.e., keep them continuous and relatively static even during dynamic pruning. Based on this analysis, the convolution kernel K after DCM cropping should always be a dense architecture, e.g., a sliceable architecture. Therefore, when the number of kernels is C, DCM will keep the first $\rho \times C$ channels and crop the remaining channels. The operation of DCM is defined as follows:

$$K' = K[:\rho \times C], \tag{4}$$

where $[:]$ is a slicing operation represented in a Python-like style.

V. Auxiliary Detection Module

Proteus involves multiple coordinated tasks in the detection process, such as receiving packets and extracting features. In order to achieve more efficient detection, we introduce an Auxiliary Detection Module deployed on network device CPUs, which mainly adopts two techniques, i.e., multi-core processing mechanism and buffering mechanism. The structure of the Auxiliary Detection Module is shown in Figure 6.

A. Multi-core Processing Mechanism

Most CPUs nowadays are equipped with multi-core processing capability to improve the overall performance of the system. Based on this characteristic, we suggest splitting the Proteus detection process into different CPU cores for processing. Specifically, we use core#1 to receive packets that need to be detected. core#2 will integrate a flow of scattered packets and extract the required features. The DDCN be deployed on the remaining cores for detection.

[3]Gumbel-softmax is used for the calculation since it avoids the non-differentiable problem of argmax.

B. Buffering Mechanism

The main challenge in the multi-core processing mechanism is the synchronization across multiple processes. In high-speed network scenarios, packets arrive much faster than feature extraction and DDCN detection. To facilitate synchronization between these tasks, we utilize two data structures, namely the ring buffer and the flow queue.

The **ring buffer** is a fixed size array with its head and tail connected. When a packet is received, it will be added to the tail of the array, and when feature extraction is performed, the packet will be extracted from the head. Through the use of ring buffers, packet reception and feature extraction processes can be avoided to some extent, which reduces the overhead of inter-process communication.The **feature queue** is used to store the extracted flow features. If DDCN detection is available, these features will be immediately provided to the corresponding DDCN. A timer is set for each flow in the queue to force detection of a flow once its storage time has expired in order to prevent certain flows from occupying the queue for a prolonged period of time.

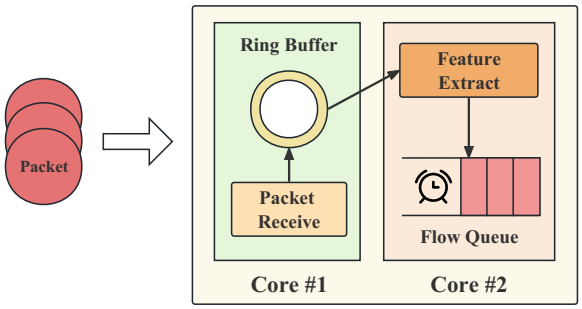


Fig. 6: Structure of the auxiliary detection module.

VI. Evaluation

In this section, we first describe the datasets used for the experiments and the related configurations. Then, we compare the detection performance of DDCN with other state-of-the-art methods. In addition, we deploy Proteus on a variety of devices and analyze its efficiency under a variety of scenarios.

A. Datasets

To evaluate the detection effectiveness of the model, we use three public datasets for all experiments, i.e., CICIDS2017 [55], UNSW-NB15[4] and DGA[5]. These three datasets were collected by different scientific institutions around the world and contain normal activities in network traffic as well as different attacks. The CICIDS2017 dataset includes benign traffic and 14 types of common attack traffic. Every sample in this dataset is precisely labeled with its category. Since the original CICIDS2017 dataset was found to contain errors, we utilized the corrected version for our analysis [56]. The UNSW-NB15 dataset was provided Cyber Range Lab of UNSW. This dataset simulates network traffic in real network environments and includes 9 common network attacks and normal traffic. The DGA data, sourced from 360Netlab, relies heavily on a comprehensive collection of PDNS data and malware samples in its detection system.

Flow-level feature extraction requires different approaches for different datasets. In the CICIDS2017 dataset and UNSW-NB15 dataset, we extract the length, sum of lengths, mean and variance of the first three packets of each flow, as well as the first 250 payload bytes, to construct features. Meanwhile, in the DGA dataset characterized by domain names, we translate each character into its ASCII equivalent and restrict the length to 100.

B. Experiment Configurations

Table I: The hardware equipment to perform the evaluation.

		Environment 1 **Raspberry PI 4B**	*Environment 2* **Ubuntu VM**	*Environment 3* **WRT32X**
CPU	Type	BCM2711	Inter i7-7700	Armada 385
	Clock	1.5GHz	2.8GHz	1.6GHz
	Cores	4	6	2
RAM		4GB	6GB	0.5GB

1) Hardware: We train DDCN and baseline models on the controller, equipped with Intel Xeon Gold 6230R CPU @ 2.10GHz and GTX 2080Ti. To evaluate the overall efficiency of the Proteus prototype, we deploy it on different devices for testing, including a Raspberry Pi 4B, an Ubuntu Linux virtual machine running on a Windows 10 PC, and a Wi-Fi router. The detailed configurations of these devices are listed in Table I.

2) Model Settings: We implement DDCN using Pytorch[6]. Table II shows the detailed settings of DDCN, i.e., the model structure of DDCN, on the DGA dataset. To further improve the inference speed of the model on the CPU, the trained DDCN on the controller is converted to run in MNN format [57].

Table II: DDCN implementation.

Layer	**Description**	**Output**
Input	Flow-level features in Section VI-A	$f \in \mathbb{R}^{100}$
Embedding	The number of chnnels $C = 16$	$f \in \mathbb{R}^{100\times16}$
DFC	The number of kernels $M_1 = 32$, the filter window $W = 4$, stride 4	$f \in \mathbb{R}^{25\times32}$
DPM	Get the cropping rate ρ	$f \in \mathbb{R}^{25\times32}$
DCM	Crop channels according to ρ	$f \in \mathbb{R}^{25\times32\times\rho}$
PW Conv	The number of kernels $M_2 = 16$, the filter window $W = 1$, stride 1	$f \in \mathbb{R}^{25\times16}$
Fully-connect		$f \in \mathbb{R}^{10}$
Softmax	Calculate class probabilities	Class label

[4]https://research.unsw.edu.au/projects/unsw-nb15-dataset

[5]https://data.netlab.360.com/dga/

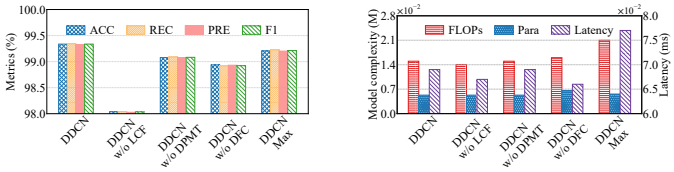
[6]https://pytorch.org/

We reproduce a series of advanced detection methods for comparison with DDCN, including FS-Net [39], SAM [58], 1DCNN [59], 2DCNN+LSTM [60], 2DCNN [18], DCNN [61] and BCN [25], all of which are implemented by Pytorch.

In order to make the model evaluation results more accurate and reliable, we used five-fold cross-validation for each scenario. We divide the dataset into five equal parts, four of which are used for model training, and the remaining is used as the test set. The five parts are repeatedly rotated and the corresponding evaluation metrics are calculated each time. The final results are obtained by averaging the five evaluation metrics.

3) Auxiliary Detection Module Settings: The whole auxiliary detection module is built in C++. According to [62], [63], 80% of the traffic in the data center can be transmitted in 10s, so we set the timeout of the flow queue to 10s.

C. Ablation Experiments



(a) Detection performance.

(b) Model complexity and detection latency.

Fig. 7: Comparison between different variants of DDCN.

To verify the effectiveness of our proposed module and training method, we conducted a series of ablation experiments using different variants of DDCN:**DDCN** is the original model; **DDCN w/o LCF** directly omits "low contribution" features judged by DPM; **DDCN w/o DPMT** is the model obtained by omitting further training on DPM; **DDCN w/o DFC** replaces the DFC module with a normal convolution; **DDCN max** eliminates the TM module and uses the largest DCNN for detection.

Figure 7a shows the detection accuracy (ACC), F1 score (F1), precision (PRE), and recall (REC) of these variants of DDCN on the DGA dataset. As can be seen, DDCN has the best performance in all four metrics, reaching a high ACC of $\sim 99.34\%$. When "low contribution" features are entirely ignored, the detection performance drops by around 1.5%. In addition, when the DFC module is replaced, the accuracy of the model decreases, confirming our hypothesis that focusing more on relevant features can enhance model effectiveness.

Figure 7b shows the model complexity and the average detection time per flow on the controller CPU for each model. In our experiments, we choose two metrics, the floating point operations (FLOPs) and the number of parameters (Para), to evaluate the complexity of the model. From Figure 7a and Figure 7b, we can see that after training the DPM separately, the DDCN not only further improves detection effectiveness but also reduces computational overhead. Compared to DDCN max, DDCN reduces FLOPs by about $\sim 26\%$ and the inference time by about $\sim 10\%$. It is worth noting that the detection performance of DDCN is better than that of DDCN Max. This may be because models of different sizes are more accurate at detecting traffic types (simple and difficult samples) that they specialize in.

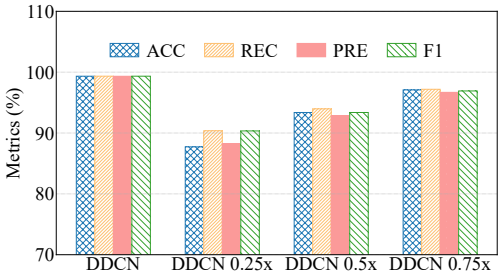
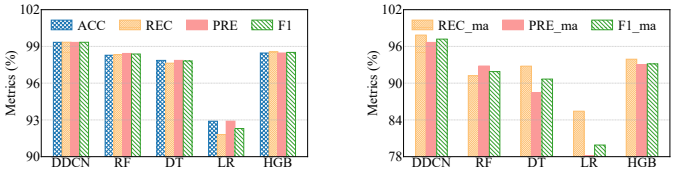


Fig. 8: Detection performance of DDCN of different sizes on the DGA dataset test.

Figure 8 illustrates the detection performance of DDCNs of different scales. Even the smallest size of DDCN achieves an ACC of 87.724%, proving that most of the traffic in the network belongs to easy samples.

D. DDCN Performance



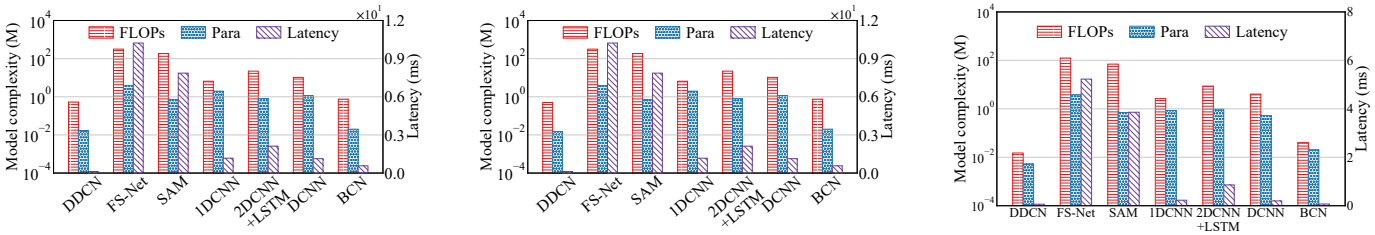
(a) Weighted measurement.

(b) Average measurement.

Fig. 9: Comparison of DDCN and ML methods.

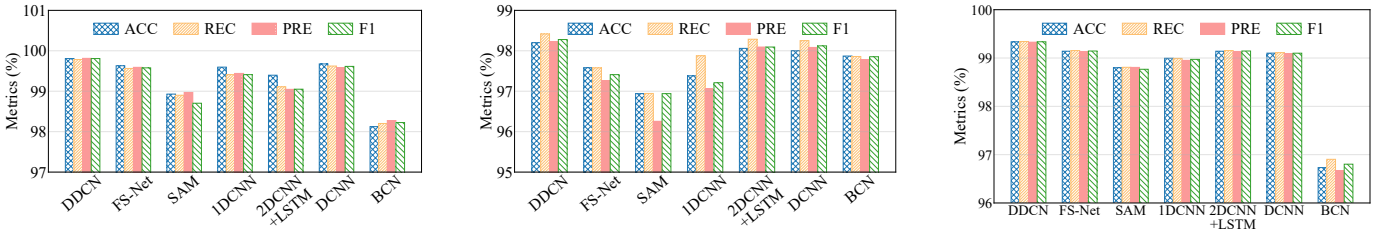
1) Comparison with shallow ML schemes: Compared with DL methods, shallow machine learning (ML) methods have long been the main approach in the field of NID due to their lower detection latency [5], [6]. However, as the complexity of detection tasks increases, their detection performance is limited. We evaluate the detection performance of DDCN against various shallow ML techniques, such as Random Forests (RF), Decision Trees (DT), Linear Regression (LR), and HistGradientBoosting (HGB), using the DGA dataset. The evaluation employed two distinct criteria sets.

The results obtained using the weighted measurement method are shown in Figure 9a, indicating that DDCN notably outstripping other ML methods. Interestingly, HGB also exhibits commendable detection capabilities, registering a REC of 98.562%. However, this assessment is significantly influenced by unbalanced data. To address the issue and ensure unbiased detection across all sample categories, we employ average evaluation method: summing the scores of each indicator within a category and subsequently comparing the average. The results, as depicted in Figure 9b, underscore



(a) On CICIDS2017 dataset. (b) On UNSW-NB15 dataset. (c) On DGA dataset.

Fig. 10: Model complexity and detection latency of each DL scheme on different datasets.

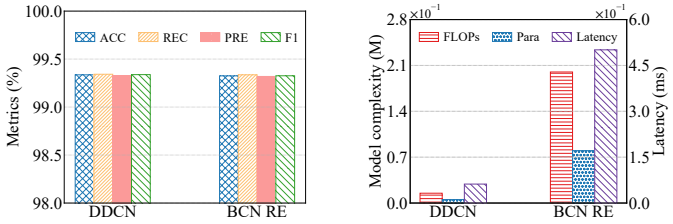


(a) On CICIDS dataset. (b) On UNSW-NB15 dataset. (c) On DGA dataset.

Fig. 11: Detection performance of DDCN and DL schemes on different datasets.

DDCN's distinct superiority over its counterparts. Notably, its PRE towers at a remarkable 18% above linear regression.

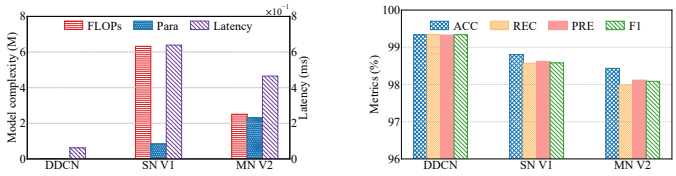
2) Comparison with DL schemes: Our experiments on three publicly available datasets, CICIDS2017, UNSW-NB15 and DGA, compare DDCN's performance to baseline DL solutions. Figure 10 illustrates the model complexity and the



(a) Detection performance. (b) Model complexity and detection latency.

Fig. 12: Comparison of DDCN and BCN RE.

detection time on the controller CPU for DDCN and baseline solutions, and DDCN has the lowest FLOPs and Para. For example, among the CICIDS2017 experiments, DDCN has 630 times lower FLOPs and 257 times lower Para compared to FS-Net [39]. Compared with BCN [25], a lightweight model, DDCN reduces FLOPs and Para by $\sim 50\%$. Regarding detection time, the detection time of DDCN is only $\sim 1.2\%$ of that of FS-Net and is slightly lower than that of BCN. All these results imply that DDCN has a significant advantage in lightweight. Figure 11 shows that DDCN outperforms other methods in the evaluation of detection performance for each metric cross datasets. For example, the detection results of DDCN on the DGA dataset are 99.34%, 99.33%, 99.34%, and 99.34% for ACC, REC, PRE, and F1, respectively. On the basis of detection time being 84 times faster than FS-Net, DDCN still outperforms it by $\sim 0.2\%$ in the four metrics.



(a) Model complexity and detection latency. (b) Detection performance.

Fig. 13: Comparison of DDCN and classic lightweight models (SN V1 represents Shufflenet V1 and MN V2 represents Mobilenet V2).

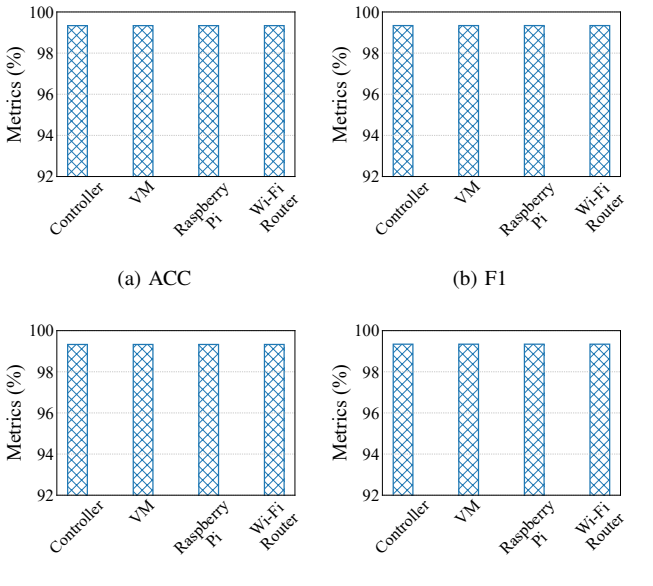
The above results prove that DDCN ensures fast inference without compromising its detection performance. On the contrary, the detection performance of BCN is not satisfactory. For example, BCN is approximately $\sim 5\%$ lower than DDCN in all indicators. On this basis, we expand the scale of BCN by increasing its computational and parameter complexity to reach a level similar to DDCN. We use BCN RE to represent the expanded BCN. Figure 12a and Figure 12b present the detection metrics, model complexity, and detection delay of DDCN and BCN RE on the DGA dataset, respectively. We can see that although they have similar results in terms of the four metrics, the computational complexity, parameter quantity, and inference delay of of BCN RE are 13 times, 15 times, and 8 times higher than DDCN, respectively. It further

demonstrates DDCN's superiority in inference accuracy over other lightweight models.

3) Comparison with lightweight models: As an additional evaluation of DDCN's performance, we compared it against two widely used lightweight models, Shufflenet V1 [49] andMobilenet V2 [29], with DDCN. Figure 13b and Figure 13a show their model complexity and inference latency as well as detection performance, respectively. As can be seen from Figure 13a, although both Shufflenet V1 and Mobilenet V2 are lightweight models, they still have higher inference latency compared to DDCN. Figure 13b shows that DDCN outperforms these two models in terms of detection results, indicating that DDCN is more effective in NID tasks.

E. Correctness of Implementation

Figure 14 shows the performance of DDCN implemented on the controller, virtual machine, Raspberry Pi, and Wi-Fi router. On the controller, DDCN is built and performs detection tasks based on Pytorch. On the other three devices, DDCN is first converted to an MNN program and then performs the detection task. During testing, the verification traffic in the DGA dataset is replayed to pass through different devices. As shown in Figure 14, DDCNs of the four devices have the same results for the four evaluation metrics, which confirms that the detection performance of our model is still guaranteed after conversion to the MNN program.



(a) ACC (b) F1

(c) PRE (d) REC

Fig. 14: Detection performance of DDCN implemented on Controllers, VM, Raspberry Pi and Wi-Fi routers.

F. Prototype Analysis

We carry out a prototype implementation of Proteus on each of the three different devices shown in Table I to show the adaptability of Proteus in different hardware environments. To ensure a fair comparison, we replace DDCN in the Proteus architecture with other DL solutions and conduct comparative experiments while keeping other modules unchanged. It should be noted that due to the large computational requirements of FS-Net and SAM, deploying these two methods on the aforementioned devices will cause system crashes, and therefore no response results will be displayed. We use artificial traffic in 10K, 20K, 30K, 40K, and 50Kfps scenarios to evaluate the performance of Proteus. We simulate the network attack scenario by constructing DGA dataset as DNS traffic and replaying it using the Linux *tcpreplay* function.

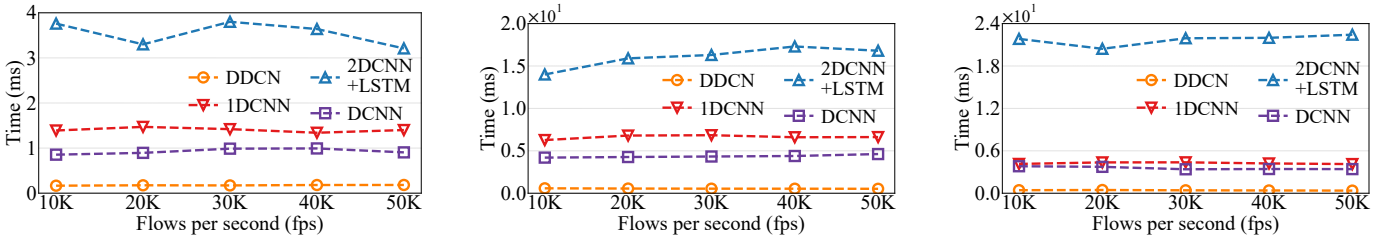
Figure 15a shows that Proteus can control the average detection time under different network traffic scenarios in a virtual machine environment with ~ 0.175 ms, which is even more efficient than some GPU-based detection solutions (e.g., $2ms$ for [58]). Figure 15b and Figure 15c show the detection time of Proteus on Raspberry Pi and Wi-Fi routers, respectively. The inference time of Proteus is found to be below 0.6 milliseconds in different scenarios, confirming its potential to be applied to real networks. Meanwhile, from the results displayed in Figure15b, it can be seen that on Raspberry Pi, the detection time of Proteus under different network conditions is about 0.55ms, while the detection time of 2DCNN+LSTM is as high as 17.3ms, which is 31 times that of Proteus. The detection time of the remaining two DL schemes 1DCNN and DCNN is also 11 times and 8 times of Proteus respectively. Clearly, Proteus is able to detect malicious traffic more efficiently and effectively than traditional DL schemes when used in real-life environments.

Considering that the performance of the auxiliary detection module is closely related to different traffic scenarios, we compare the memory usage of the ring buffer in the Auxiliary Detection Module on two schemes, Proteus and 2DCNN+LSTM. Figure 16 shows the comparison of ring buffer memory usage between DDCN and 2DCNN+LSTM on different devices in different network environments.

In general, ring buffer memory usage is positively correlated with the packet sending rate and negatively correlated with the overall system processing speed. That is, the faster the packet sending rate, the higher the usage, and the faster the processing speed, the lower the usage. Compared with 2DCNN+LSTM, Proteus has a lower memory usage. For example, experiments on a virtual machine show that at 50Kfps, Proteus has a memory usage of only 0.05MB, while 2DCNN+LSTM reaches 0.76MB.

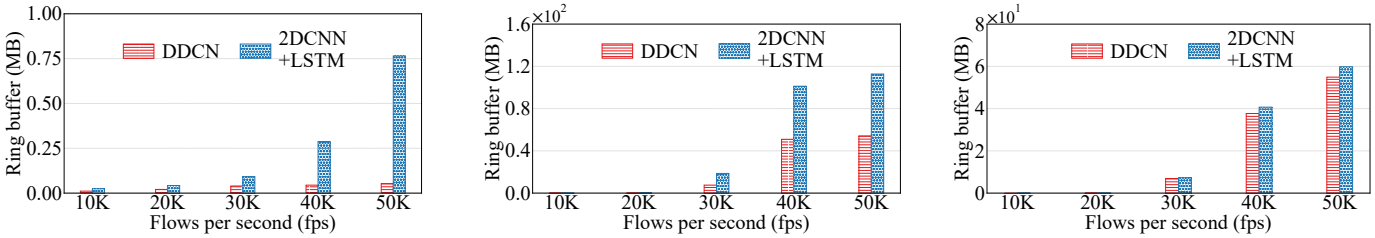
VII. Limitation and Future Work

One limitation of the proposed Proteus is that its training and detection processes are conducted under the "closed-world" assumption, where the model predicts that each sample is similar to those encountered during training. However, given the continuous emergence of new attacks, it is essential for Proteus to be capable of detecting zero-day traffic. To enhance the capability of neural networks in recognizing novel attacks, one can employ unsupervised learning methods such as Autoencoders (AE) [64], [65]. Once trained, an AE effectively reconstructs inputs that mirror patterns observed during its



(a) on VM. (b) on Raspberry Pi. (c) on Wi-Fi router.

Fig. 15: The average detection latency of Proteus and other DL schemes under different network environment conditions on various devices.



(a) On VM. (b) On Raspberry Pi. (c) On Wi-Fi router.

Fig. 16: Comparison of Ringbuffer memory usage between DDCN and 2DCNN+LSTM in different network environments on different devices.

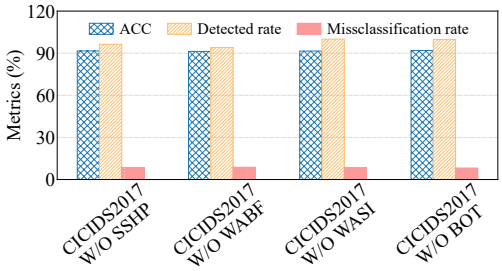


Fig. 17: The AE+DDCN performance: the detected rate of find- ing flows from new (unknown) attacks correctly, the misclassification rate of classifying flows from known classes as unknown, and the overall accuracy

training phase. Conversely, its reconstruction capability diminishes when presented with unfamiliar class inputs. Hence, we suggest integrating a three-layer AE into the DDCN to identify new attacks. We assess the enhanced DDCN in an "open world" context. Specifically, using the CICIDS2017 dataset, we systematically omit certain attacks from the training data on each occasion (e.g., SSH-Patator, Web Attack Brute Force, Web Attack Sql Injection, or Bot). As illustrated in Figure 17, the combined AE+DDCN method can identify over 94% of traffic stemming from new attacks. It is worth noting that this is an initial plan to help Proteus detect new attacks, and future research will focus on enhancing its ability to detect unknown types of network intrusions.

Additionally, we are considering combining Proteus with high throughput programmable switches (e.g., P4 switches [66]) to further reduce the detection latency. Specifically, a two-stage detection scheme can be employed: quickly filter out a large amount of benign traffic on the data plane, and only clone suspicious traffic to the CPU of the switch and using Proteus for fine-grained detection.

VIII. Conclusion

In this paper, we propose Proteus, a difficulty-aware deep learning framework for real-time malicious traffic Detection. Proteus includes a Double Dynamic Convolutional Neural Network (DDCN), using a Dynamic Feature Campaign module (DFC) to focus on features that have a significant impact on the results, and a Tailor Module (TM) to adaptively differentiate the detection difficulty of various samples and crop the structure of the model. Results show that Proteus can detect flows accurately in different network environments on low-performance network devices with low latency. In summary, Proteus can be deployed on network devices without acceleration hardware such as GPUs to achieve faster and more accurate detection of malicious network attacks. We hope that Proteus will be helpful to researchers in their related work.

IX. Acknowledgments

This work is supported by the Major Key Project of PCL under grant No. PCL2023A06, the National Key Research and Development Program of China under grant No. 2022YFB3105000, and the Shenzhen Key Lab of Software Defined Networking under grant No. ZDSYS20140509172959989.

REFERENCES

[1] S.-W. Lee, H. Mohammed sidqi, M. Mohammadi, S. Rashidi, A. M. Rahmani, M. Masdari, and M. Hosseinzadeh, "Towards secure intrusion detection systems using deep learning techniques: Comprehensive analysis and review," *Journal of Network and Computer Applications*, vol. 187, p. 103111, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804521001314

[2] P. Illy, G. Kaddoum, C. M. Moreira, K. Kaur, and S. Garg, "Securing fog-to-things environment using intrusion detection system based on ensemble learning," in *Proceedings of the 2019 IEEE Wireless Communications and Networking Conference*. IEEE, 2019, pp. 1–7.

[3] T. Komviriyavut, P. Sangkatsanee, N. Wattanapongsakorn, and C. Charnsripinyo, "Network intrusion detection and classification with decision tree and rule based approaches," in *2009 9th International Symposium on Communications and Information Technology*, 2009, pp. 1046–1050.

[4] S. Sahu and B. M. Mehtre, "Network intrusion detection system using j48 decision tree," in *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2015, pp. 2023–2026.

[5] A. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2017.

[6] G. Apruzzese, P. Laskov, and J. Schneider, "Sok: Pragmatic assessment of machine learning for network intrusion detection," in *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, 2023, pp. 592–614.

[7] C. Fu, Q. Li, M. Shen, and K. Xu, "Realtime robust malicious traffic detection via frequency domain analysis," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2021.

[8] N. Doulamis and A. Voulodimos, "Fast-mdl: Fast adaptive supervised training of multi-layered deep learning models for consistent object tracking and classification," in *2016 IEEE International Conference on Imaging Systems and Techniques (IST)*. IEEE, 2016, pp. 318–323.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[10] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, "Learning deep structured semantic models for web search using clickthrough data," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2013, pp. 2333–2338.

[11] R. Yang, J. Zhang, X. Gao, F. Ji, and H. Chen, "Simple and effective text matching with richer alignment features," *arXiv preprint arXiv:1908.00300*, 2019.

[12] J. Xiao, Q. Zou, Q. Li, D. Zhao, K. Li, W. Tang, R. Zhou, and Y. Jiang, "User device interaction prediction via relational gated graph attention network and intent-aware encoder," in *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2023, pp. 1634–1642.

[13] J. Xiao, Q. Zou, Q. Li, D. Zhao, K. Li, Z. Weng, R. Li, and Y. Jiang, "I know your intent: Graph-enhanced intent-aware user device interaction prediction via contrastive learning," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMUWT/UbiComp)*, vol. 7, no. 3, pp. 1–28, 2023.

[14] Q. Zou, Q. Li, R. Li, Y. Huang, G. Tyson, J. Xiao, and Y. Jiang, "Iotbeholder: A privacy snooping attack on user habitual behaviors from smart home wi-fi traffic," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 7, no. 1, pp. 1–26, 2023.

[15] J. Xiao, Z. Xu, Q. Zou, Q. Li, D. Zhao, D. Fang, R. Li, W. Tang, K. Li, X. Zuo *et al.*, "Make your home safe: Time-aware unsupervised user behavior anomaly detection in smart homes via loss-guided mask," *arXiv e-prints*, pp. arXiv–2406, 2024.

[16] P. Illy, G. Kaddoum, C. Miranda Moreira, K. Kaur, and S. Garg, "Securing fog-to-things environment using intrusion detection system based on ensemble learning," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, 2019, pp. 1–7.

[17] A. Meliboev, J. Alikhanov, and W. Kim, "1d CNN based network intrusion detection with normalization on imbalanced data," in *Proceedings of the 2020 International Conference on Artificial Intelligence in Information and Communication*. IEEE, 2020, pp. 218–224.

[18] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *Proceedings of the 2017 International Conference on Information Networking*. IEEE, 2017, pp. 712–717.

[19] Y. Dong, Q. Li, K. Wu, R. Li, D. Zhao, G. Tyson, J. Peng, Y. Jiang, S. Xia, and M. Xu, "HorusEye: A realtime IoT malicious traffic detection framework using programmable switches," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 571–588. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/dong-yutao

[20] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41 525–41 550, 2019.

[21] G. Marín, P. Casas, and G. Capdehourat, "Deep in the dark - deep learning-based malware traffic detection without expert knowledge," in *2019 IEEE Security and Privacy Workshops (SPW)*, 2019, pp. 36–42.

[22] K. Lin, X. Xu, and F. Xiao, "Mffusion: A multi-level features fusion model for malicious traffic detection based on deep learning," *Computer Networks*, vol. 202, p. 108658, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128621005399

[23] A. Parmisano, S. Garcia, and M. J. Erquiaga, "A labeled dataset with malicious and benign iot network traffic," *Stratosphere Laboratory: Praha, Czech Republic*, 2020.

[24] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, 2018.

[25] G. Xie, Q. Li, C. Cui, P. Zhu, D. Zhao, W. Shi, Z. Qi, Y. Jiang, and X. Xiao, "Soter: Deep learning enhanced in-network attack detection based on programmable switches," in *2022 41st International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2022, pp. 225–236.

[26] "A lightweight neural network with strong robustness for bearing fault diagnosis," *Measurement*, vol. 159, p. 107756, 2020.

[27] T.-W. Hui, X. Tang, and C. C. Loy, "Liteflownet: A lightweight convolutional neural network for optical flow estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8981–8989.

[28] L. Sifre and S. Mallat, "Rigid-motion scattering for texture classification," *arXiv preprint arXiv:1403.1687*, 2014.

[29] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition*. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 4510–4520.

[30] N. Ma, X. Zhang, H. Zheng, and J. Sun, "Shufflenet V2: practical guidelines for efficient CNN architecture design," in *Proceedings of the 15th European Conference on Computer Vision, 2018*. Springer, 2018, pp. 122–138.

[31] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," 2017. [Online]. Available: https://arxiv.org/abs/1709.01686

[32] X. Wang and Y. Li, "Harmonized dense knowledge distillation training for multi-exit architectures," in *AAAI Conference on Artificial Intelligence*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:235349240

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[34] C. Li, G. Wang, B. Wang, X. Liang, Z. Li, and X. Chang, "Dynamic slimmable network," in *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, 2021, pp. 8607–8617.

[35] X. Gao, Y. Zhao, L. Dudziak, R. Mullins, and C. Z. Xu, "Dynamic channel pruning: Feature boosting and suppression," 2018.

[36] D. Javaheri, S. Gorgin, J.-A. Lee, and M. Masdari, "Fuzzy logic-based ddos attacks and network traffic anomaly detection methods: Classification, overview, and future perspectives," *Information Sciences*, vol. 626, pp. 315–338, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025523000683

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

[38] B. Yu, D. L. Gray, P. Jie, M. Cock, and A. Nascimento, "Inline dga detection with deep networks," in *IEEE International Conference on Data Mining Workshops*, 2017.

[39] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, "Fs-net: A flow sequence network for encrypted traffic classification," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 1171–1179.

[40] X. Lin, G. Xiong, G. Gou, Z. Li, J. Shi, and J. Yu, "Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification," in *Proceedings of the ACM Web Conference 2022*, ser. WWW '22. ACM, Apr. 2022. [Online]. Available: http://dx.doi.org/10.1145/3485447.3512217

[41] H. Zhang, L. Yu, X. Xiao, Q. Li, F. Mercaldo, X. Luo, and Q. Liu, "Tfe-gnn: A temporal fusion encoder using graph neural networks for fine-grained encrypted traffic classification," in *Proceedings of the ACM Web Conference 2023*, ser. WWW '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 2066–2075. [Online]. Available: https://doi.org/10.1145/3543507.3583227

[42] W. Wei, Z. Ming, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *2017 International Conference on Information Networking (ICOIN)*, 2017.

[43] G. Huang, D. Chen, T. Li, F. Wu, V. Laurens, and K. Q. Weinberger, "Multi-scale dense networks for resource efficient image classification," 2017.

[44] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," 2018.

[45] B. Yang, G. Bender, J. Ngiam, and Q. V. Le, "Condconv: Conditionally parameterized convolutions for efficient inference," 2019.

[46] Y. Chen, X. Dai, M. Liu, D. Chen, L. Yuan, and Z. Liu, "Dynamic convolution: Attention over convolution kernels," 2019.

[47] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proceedings of the 1st International Conference on Learning Representations, 2013*, 2013.

[48] Z. Su, L. Fang, W. Kang, D. Hu, M. Pietikäinen, and L. Liu, "Dynamic group convolution for accelerating convolutional neural networks," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16*. Springer, 2020, pp. 138–155.

[49] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," 2017.

[50] C. Li, J. Peng, L. Yuan, G. Wang, and X. Chang, "Block-wisely supervised neural architecture search with knowledge distillation," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[51] Z. Guo, X. Zhang, H. Mu, H. Wen, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," 2019.

[52] A. Brock, T. Lim, J. M. Ritchie, and N. J. Weston, "Smash: One-shot model architecture search through hypernetworks," in *International Conference on Learning Representations*, 2018.

[53] J. Yu and T. Huang, "Universally slimmable networks and improved training techniques," in *International Conference on Computer Vision*, 2019.

[54] W. Hua, Y. Zhou, C. Sa, Z. Zhang, and G. E. Suh, "Channel gating neural networks," in *Neural Information Processing Systems*, 2019.

[55] G. Engelen, V. Rimmer, and W. Joosen, "Troubleshooting an intrusion detection dataset: the cicids2017 case study," in *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2021, pp. 7–12.

[56] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *International Conference on Information Systems Security & Privacy*, 2018.

[57] X. Jiang, H. Wang, Y. Chen, Z. Wu, L. Wang, B. Zou, Y. Yang, Z. Cui, Y. Cai, T. Yu, C. Lyu, and Z. Wu, "MNN: A universal and efficient inference engine," in *Proceedings of Machine Learning and Systems*. mlsys.org, 2020, pp. 1–13.

[58] G. Xie, Q. Li, and Y. Jiang, "Self-attentive deep learning method for online traffic classification and its interpretability," *Computer Networks*, p. 108267, 2021.

[59] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *Proceedings of the 2017 International Conference on Intelligence and Security Informatics*. IEEE, 2017, pp. 43–48.

[60] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for internet of things," *IEEE Access*, vol. 5, pp. 18 042–18 050, 2017.

[61] D. Kwon, K. Natarajan, S. C. Suh, H. Kim, and J. Kim, "An empirical study on network anomaly detection using convolutional neural networks," in *Proceedings of the 38th IEEE International Conference on Distributed Computing Systems, 2018*. IEEE Computer Society, 2018, pp. 1595–1598.

[62] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference, 2010*. ACM, 2010, pp. 267–280.

[63] S. Kandula, S. Sengupta, A. G. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM Internet Measurement Conference, 2009*. ACM, 2009, pp. 202–208.

[64] H. Shin, M. Orton, D. J. Collins, S. J. Doran, and M. O. Leach, "Stacked autoencoders for unsupervised feature learning and multiple organ detection in a pilot study using 4d patient data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1930–1943, 2013.

[65] R. Tang, Z. Yang, Z. Li, W. Meng, H. Wang, Q. Li, Y. Sun, D. Pei, T. Wei, Y. Xu, and Y. Liu, "Zerowall: Detecting zero-day web attacks through encoder-decoder recurrent neural networks," in *Proceedings of the 39th IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2479–2488.

[66] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.