Generating Neural Networks for Diverse Networking Classification Tasks via Hardware-Aware Neural Architecture Search

Guorui Xie, Qing Li, *Member, IEEE*, Zhenning Shi, Hanbin Fang, Shengpeng Ji, Yong Jiang, *Member, IEEE*, Zhenhui Yuan, Lianbo Ma, *Senior Member, IEEE*, and Mingwei Xu, *Senior Member, IEEE*

Abstract—Neural networks (NNs) are widely used in classification-based networking analysis to help traffic transmission and system security. However, there are heterogeneous network devices (e.g., switches and routers) in a network. Manually customizing NNs with specific device requirements (e.g., max allowed running latency) can be time-consuming and labor-intensive. Furthermore, the diverse data characteristics of different networking classification tasks add to the burden of NN customization. This paper introduces Loong, a neural architecture search (NAS) based system that automatically generates NNs for various networking tasks and devices. Loong includes a neural operation embedding module, which embeds candidate neural operations into the layer to be designed. Then, the layer-wise training is used to generate a task-specific NN layer by layer. This layer-wise scheme simultaneously trains and selects candidate neural operations using gradient feedback. Finally, only the important operations are selected to form the layer, maximizing accuracy. By incorporating multiple objectives, including deployment memory and running latency of devices, into the training and selection of NNs, Loong is able to customize NNs for heterogeneous network devices. Experiments show that Loong's NNs outperform 13 manual-designed and NAS-based NNs, with a 4.11% improvement in F1-score. Additionally, Loong's NNs achieve faster (7.92X) speeds on commodity devices.

Index Terms—Neural network, automated design, traffic classification, attack detection.

1 Introduction

Networking classification refers to the process of categorizing network elements (e.g., byte streams or domain names) into predefined classes. The outcomes of networking classification can assist network administrators in comprehending activities in the network and taking appropriate actions to achieve various purposes, such as enhancing system security and Quality-of-Service (QoS) [1]. Given the complexity of computer networks, we argue that there are several characteristics of current networking classification. The first is the task diversity. Network administrators are interested in classifying various network elements, such as the raw bytes [2], [3] or statistics [4], [5] in the traffic, and the strings in URLs or DNS [6], [7]. The second is the multiple deployment devices and objectives. A network in the real world consists of different devices, including gateways, routers, and switches that vary in memory and computation performance. Therefore, besides the accuracy, networking classification is expected to run efficiently on these devices in terms of memory usage and latency [8]–[10].

Recently, machine learning (ML) approaches, particularly those based on neural networks (NNs), have been proposed for networking classification [4], [11]–[13]. For instance, the authors in [11] propose a hybrid convolutional and recurrent NN to map byte streams to the applications that generated them. In [12], a long short-term memory neural architecture is designed to determine whether domain name strings are generated by Botnets. In addition to achieving high accuracy, network administrators may also consider the memory/latency requirements. In [8], the authors modify the NN architecture to address memory and computation resource constraints by incorporating lightweight ensemble autoencoders for low-latency attack detection on routers. However, these manual-designed NNs require a significant investment of time and human resources [14]. Procedures like evaluating the NNs consisting of multiple candidate neural operations and debugging the improper connections between neural operations are required [15]. Besides, such procedures must be repeated for each individual networking task. Even for the same task, the final selected NN may still need to be frequently adjusted, due to the

- *Guorui Xie and Yong Jiang are with the Tsinghua Shenzhen International Graduate School, Shenzhen 518055, China, and also with the Peng Cheng Laboratory (PCL), Shenzhen 518066, China. E-mails: xgr19@mails.tsinghua.edu.cn; jiangy@sz.tsinghua.edu.cn*
- *Qing Li is with the Peng Cheng Laboratory (PCL), Shenzhen 518066, China. E-mail: liq@pcl.ac.cn*
- *Zhenning Shi is with the Tsinghua Shenzhen International Graduate School, Shenzhen 518055, China. E-mail: zhenningshi@foxtail.com*
- *Hanbin Fang is with the Jilin University, Changchun 135124, China. E-mail: fanghb2120@mails.jlu.edu.cn*
- *Shengpeng Ji is with the Zhejiang University, Hangzhou 310058, China. E-mail: jisp5519@mails.jlu.edu.cn*
- *Zhenhui Yuan is with the Department of Computer and Information Sciences, Northumbria University, Newcastle Upon Tyne, NE1 8ST, UK. E-mail: zhenhui.yuan@northumbria.ac.uk*
- *Lianbo Ma is with the College of Software, Northeastern University, Shenyang 110169, China. E-mail: malb@swc.neu.edu.cn*
- *Mingwei Xu is with the Department of Computer Science and Technology, Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China, and also with the Quan Cheng Laboratory, Jinan 250103, China. E-mail: xumw@tsinghua.edu.cn*

Corresponding author: Qing Li.

various memory and computation in network devices.

A promising solution to automatically design NNs for networking classification is the Neural Architecture Search (NAS) [16]–[20]. DARTS [17] focus on the possible neural operations and their connections within a layer. Then, the searched layer is stacked multiple times to form the final NN. Nonetheless, DARTS does not consider the layer diversity [18], which may hurt the accuracy given the diverse networking classification tasks of different input data. Also, DARTS does not consider the hardware characteristics (e.g., memory and latency of network devices). ProxylessNAS [18], LightNAS [19], and [20] are hardware-aware mainly in terms of running latency. But network devices like routers have small memory (hundreds of MB), which makes memory usage also critical [8], [10]. Besides, ProxylessNAS models the latency as a soft constraint in the search loss, which indicates that ProxylessNAS can only guide the NN to be lightweight on latency but it cannot design the NN to have a desired specific latency. More importantly, given the diverse network devices, the NAS training of [18]–[20] should be inefficiently repeated for each device.

In this paper, we propose Loong[1], a hardware-aware NAS-based system for networking classification, which considers task diversity, multiple devices, and multi-objectives (e.g., accuracy, latency, and memory). Loong allows designing NNs of layers consisting of different neural operations, improving the layer diversity within the searched NN to handle a wide range of networking classification tasks. After the one-shot automated design, Loong generates a series of NNs varying in the number of layers. For multiple network devices, each of them can be efficiently recommended an accurate NN that meets the specific memory or latency constraint. In summation, **we make the following contributions when designing Loong**:

- We comprehensively review the state-of-the-art ML literature [21]–[28] and define far more candidate neural operations than previous solutions (25 operations in total). By embedding these sufficient operations in the neural layers to be designed, Loong is poised to adaptively select the most suitable operations when presented with varying characteristics of task data, resulting in high-quality NNs.
- We propose a Layer-wise Training (LWT) approach to design the optimal NNs progressively. Unlike DARTS [17], our layers can choose neural operations different from each other for accuracy gains on diverse networking tasks. However, naively including all candidate operations of all layers can lead to GPU memory explosion [18]. As such, we use a divide-and-conquer manner, designing the NN layer by layer. In designed layers, they only maintain the determined operations of high importance. In the layer to be currently designed, each possible operation is assigned an importance weight. The trainable parameters and the importance weights of operations are updated based on the gradients. By greedily exploring the layers instead of constructing numerous possible layers and operations, Loong effectively reduces the GPU demand.
- We incorporate the multiple objectives and devices in Loong. We utilize Objective Regularization (OR) in Loong's LWT as part of the loss function. The OR is a weighted sum of the memory/latency costs of candidate operations that are measured on one proxy device. As we discussed ProxylessNAS previously, soft constraints like OR cannot guide Loong to design the NN to meet a specific hardware objective. To obtain NNs for exact devices and objectives without repeated NAS training like [17]–[20], we propose an objective-based selection. That is, consecutive designed layers can be arbitrarily selected as a NN for a device through the running test on the device. This selection can be accelerated by the binary search.
- We conduct intensive experiments to compare Loong with 13 state-of-the-art NAS/handcrafted NNs on three tasks of different input data, and also deploy Loong's NNs on three network devices[2]. To the best of our knowledge, this is the first work to comprehensively analyze the feasibility and effectiveness of hardware-aware NAS in the field of networking. Our experiments reveal that: 1) When compared with other schemes, NNs generated by Loong increase accuracy by 3.94% and F1-score by 4.11%. 2) Loong shows superior generalization on various networking tasks, achieving accuracies of 94+% across all evaluated tasks. 3) Loong succeeds in returning optimal NNs for given objectives. E.g., for the max allowed latency of 0.25ms, the returned NN can classify 9188.98 samples per second on the device.

2 Background

2.1 Networking Classification Diversity

Through the training, NNs can intelligently learn representative patterns from the observed data, and thus improve the performance for many use-cases [14], [15]. As such, NNs have been promising solutions for solving a multitude of networking classification tasks [1]. These tasks are usually associated with various types of data as follows.

Classification on Byte Streams. In network communication, packet delivery enables the exchange of information between hosts. Network administrators can capture the flowing packets, extract their raw bytes, and analyze the communication for various purposes. In [2], the authors propose DeepPacket, a convolutional classification framework that handles tasks on traffic service classification (packets are classified into application protocols including FTP and P2P) and traffic application identification (packets are classified into applications like Gmail and Skype). In [3], the authors present BGRUA to identify the web services running upon the packets. BGRUA uses a NN based on the gated recurrent units and takes the first 2700 bytes of each session as input for classification. Other related works are [11], [29], [30].

Classification on Strings. In some scenarios, strings like URLs and domain names play a significant role in delivering information across networks and are valuable sources for analysis. For example, in [6], the authors present a model,

1. In Chinese mythology "Nine Sons of Loong", Loong can give birth to numerous children with different magics.

2. The code will be open-source after the acceptance.

CNN-MHSA, to detect phishing websites based on the clicked URLs. The model is designed with a Convolutional Neural Network (CNN) and takes URL strings as input. On the other hand, domain names in the DNS packets are often not monitored by firewalls, making them easier to be tunnels for data leakage. Hence, the authors in [7] present a model, BCNN, to identify whether a domain name is used as a tunnel for private data exfiltration. Also, domain names can be generated by families of domain generation algorithms (DGA) for Botnet activities. Therefore, LSTM is used for the DGA family classification in [12]. Other works are [31], [32].

Classification on Flow Statistics. Network administrators also gather various statistical features of a flow for analysis. Typically, a flow is defined as a series of packets that have the same five-tuple (i.e., source and destination IP addresses, source and destination ports, and transmission protocol number). Statistical features, such as the average packet transmission rate and inter-packet arrival time, are represented as floating-point numbers. In the study presented in [33], the authors propose a Feed-Forward Deep Neural Network (FFDNN) and utilize 48 statistical features of flows for anomaly detection. Similarly, in [4], [5], the authors extract statistical features from IoT network flows and feed them into either a Convolutional Neural Network (CNN) or Recurrent Neural Network (RNN) model for cyber attack detection. Additional studies leveraging statistical features for network analysis include [13], [34], [35].

2.2 Multiple Devices and Objectives in Networking

In networking, classification accuracy is not the sole concern for the variety of tasks at hand. Given the limited memory and computation capacity of off-the-shelf network devices like gateways, routers, and even programmable switches [8], [9], network administrators must also consider the memory usage and running latency of NNs for tasks on these heterogeneous devices [36]. This ensures that NNs can operate efficiently across the network's devices, enabling effective network-wide tasks.

Selecting the proper NN according to desired memory/latency objectives is critical for network devices. For instance, in resource-constrained routers, memory is scarce (hundreds of MB) and should be often a shared resource, used by multiple critical networking functions (e.g., routing and network address transmission). If the designed NN is too large in terms of memory consumption, it will be failed to be deployed with other functions, dramatically weakening the fundamental traffic transmission [8]. Also, if the deployed NN is of high latency and runs slowly, even though it is highly accurate, the reactions on the classified traffic cannot be performed timely to guarantee the network QoS or security (e.g., failing to block detected attack activities in time). These concerns highlight the importance to design suitable NNs that should not only have high classification accuracy, but also meet specific memory/latency objectives [30].

Unlike the accuracy, the memory/latency requirements of an NN are related to both the neural architecture and the hardware factors [16], [37]. As such, there are two aspects to be considered for these objectives. First, during the design, one should select the efficient neural operations to build the NN. E.g., selecting the operations that require less memory/latency but do not degrade the classification performance dramatically. Second, after the design is completed, the designed NN is tested on the deployment device to check whether or not it meets the objectives. If not, the design process should be repeated or fine-tuned.

2.3 Neural Architecture Search

As networking classification varies in the tasks and objectives, previous solutions that require researchers to tailor appropriate NNs for specific requirements are too expensive in time and human resources [14], [15]. A promising solution to these obstacles is the Neural Architecture Search (NAS) technique, which aims to automate the generation of NNs [16]–[20], [38]–[40].

At the early stage, NAS is based on Reinforcement Learning (RL) or Evolution Algorithm (EA). In [38], the authors use an RL-based agent to replace the manual effort of designing NNs. At first, the agent generates numerous candidate NNs and trains them from scratch. Then the evaluated performance of these NNs is used as the reward to train the agent's NN-generation strategy. In [39], the authors randomly produce a large number of NNs as a population and then use the EA to evolve the population. That is, training all NNs, removing the ones that are below the accuracy threshold, and then using the survivors to produce new NNs. Nevertheless, RL and EA solutions train candidate NNs discretely and are computationally expensive [16].

Recently, the more efficient differentiable NAS has been proposed, which explores all possible neural operation connections in a differentiable architecture. DARTS [17] regard a neural layer (aka cell) as a directed acyclic graph (DAG) where edges in the DAG are candidate neural operations. The training in DARTS is to find several important operations to form the layer. The final NN is formed by stacking this layer multiple times. However, DARTS does not consider the layer diversity [18], which may degrade the NN performance given the task diversity in networking. Besides, DARTS does not incorporate hardware characteristics. Solutions [18]–[20] are hardware-aware. ProxylessNAS [18] records operations' latencies in a lookup table (LUT). Then, values in the LUT are weighted and summed as a soft constraint to guide the operation selection in the training. This soft constraint, however, can not find an NN of a desired latency. Instead, [19], [20] use special neural models to predict the NN latency of a device during the training so as to design NNs of specific latency. But these schemes require inefficient repeat to design NNs for multiple devices. Besides, the mentioned hardware-aware methods do not consider the vital memory constraints in network devices.

As these methods are not specifically designed for networking and cannot well meet the requirements in Section 2.1 and 2.2, we want to design a new generic NAS system, which can yield high accuracy on the various networking tasks and efficiently generate NNs for multiple network devices and objectives (memory/latency).

3 Loong Overview

3.1 Problem Statement

The generic multi-objective neural automation is to build an approach to automatically generate (design and train together) NNs for various networking classification tasks, devices, and objectives, while the approach itself should be computationally efficient in GPU memory/time. Network administrators can organize data via the pair of (x, y) where x is the data fed to the NN and y is the class label. Also, an extra input z is accepted to indicate the allowed max memory usage or running latency of the desired NN on network devices, i.e., the objectives. The approach will output NNs satisfying the provided objectives with high accuracy.

3.2 Framework Overview

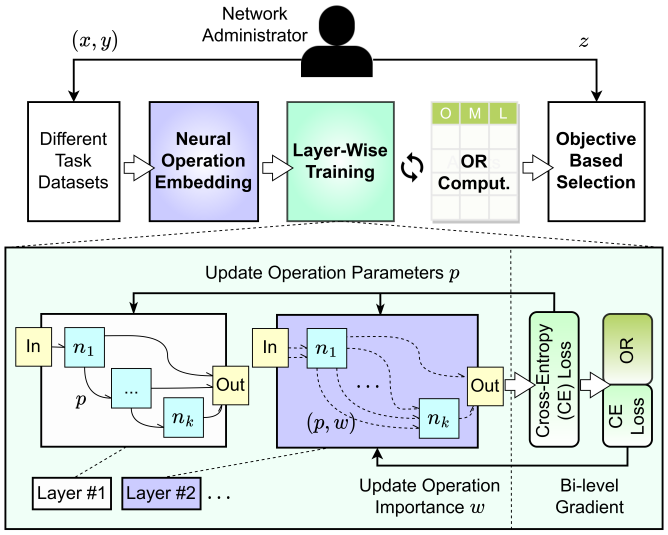

Fig. 1: The framework of Loong.

The framework of Loong is presented in Fig. 1. After the network administrator provides the dataset (x, y) and the objective z, Loong can be run on a powerful server of GPUs to generate NNs, and then recommends suitable NNs to run on network devices (e.g., routers, programmable switches) with the objective z.

The **Neural Operation Embedding** (detailed in Section 4) embeds the candidate operations into the layer by a directed acyclic graph. This means that the layer to be designed (e.g., Layer #2) consists of several inner nodes $n_1, \ldots, n_k$. The nodes represent the processed intermediate data, and each node is connected by several directed dashed edges from its predecessor nodes. These dashed edges represent the candidate operations. For each edge (operation), besides the trainable built-in parameter p, an importance weight w is assigned.

The **Layer-wise Training** (detailed in Section 5) trains and designs the NN layer by layer. The parameter p and the importance weight w of each edge (operation) are updated via the bi-level gradient. The first gradient is computed from the cross-entropy (CE) loss between the predictions and the true labels in y, and is used to train p of all existing layers. The second gradient, which is computed from the sum of CE and OR, is used to update w of the layer currently being designed. After the bi-level gradient update converges, the dashed edges of the directed node are eliminated, and only the edge with the highest w is kept (e.g., the solid edge in Layer #1).

The **OR Comput.** (detailed in Section 6) utilizes a lookup table to compute the OR weighted by w. The lookup table records three elements (O, M, L) per row, i.e, the operations (O) and the corresponding measured memory/latency (M/L) on a network device. During the layer design, we compute the OR by summing the results of $w \times M$ (or L). Then, the OR is reflected in the gradient to penalize the high importance w of unsuitable operations. When there are numerous deployment devices, constructing a lookup table for each device can be labor-intensive. Therefore, we propose to utilize a lookup table of the proxy device in that multi-device scenario.

The **Objective-based Selection** (detailed in Section 7) runs tests on the target device, finding the NN (aka sub-NN) that has the most consecutive generated layers while does not exceed the objective memory/latency (as more layers are more accurate). Finally, this sub-NN is returned to be the device-selected one that satisfies the hardware constraints and yields high accuracy on the given task. Moreover, we also accelerate the sub-NN selection for multiple devices based on a binary search.

With these delicate modules, Loong is able to generate NNs for a wide variety of networking tasks, satisfying the objectives of devices. Unlike previous NAS solutions (e.g., [38], [39]) which train and evaluate numerous NNs in vain, Loong combines appropriate operations layer by layer to greedily find the task-specific NN, reducing the required GPU memory and time during the search.

4 Neural Operation Embedding

4.1 Candidate Neural Operations

Previous solutions only repeat three or four kinds of neural operations to build NNs, which reduces human resources but may lead to poor generalization for different tasks. To help Loong adapt well to various tasks, we provide a set of sufficient candidate neural operations for each layer to be designed. In summary, we consider operations from three aspects: convolution block, pooling function, and *skip*. In fact, the candidate operations in Loong are easy to be extended (we will further study this in the future).

Like [4], [7], our **convolution block** consists of operations in the order of {*convolution, activation function, batch normalization*}. The difference is that we explore more kinds of convolutions and activation functions. Besides the ordinary convolution, the newly proposed techniques are considered, including group convolution [22] and dilated convolution [21]. Additionally, the kernel sizes of convolutions can be 3 or 5. As for the activation functions, we employ the widely used ReLU [25] and its variants (LeakyReLU [23], ELU [24]). The **pooling functions** are also of different types (average [27], max [26], power-average [28]) and of different kernel sizes (3 or 5). The ***skip*** is a special operation, indicating that there is no applied neural operation from the source node to the destination node. In total, 25 neural operations are defined in the candidate set as illustrated in Table 1.

Table 1: The candidate neural operations in Loong.

	Description				Sum
Convolution Block	Convolution		Three Activation Functions	One Normalization	
	Three Types	Two Kernel Sizes			
	Ordinary, Group, Dilated	3, 5	ReLU, Leaky, ELU	Batch Normalization	3x2x3x1 =18
Pooling	Three Types	Two Kernel Sizes			
	Average, Max, Power-average	3, 5			3x2=6
Skip	Apply no neural operation				1

4.2 Operations in a Layer

After providing the set O of candidate neural operations, we embed O into the layer. We view the layer as a directed acyclic graph. The inner nodes of the graph are intermediate representations of the input data, i.e., the output of some operation edges. For instance, for the data (node n_i) and a neural operation $o \in O$, the output (successor node n_j) can be obtained by $n_j = o(n_i)$, where $i \in [0, j)$.

For the layer to be designed, the computation of n_j is complex as we must consider all the combinations of predecessor nodes $0 \leq i < j$ and candidate operations $o \in O$ to find the optimality. That is,

$$n_j = \sum_{0 \leq i < j} \sum_{o \in O} \mathrm{SoftMax}(w^o_{i,j}) \times o(n_i), \tag{1}$$

where $w^o_{i,j}$ is the trainable importance weight of operation o directed from n_i to n_j. $\mathrm{SoftMax}(w^o_{i,j}) = \frac{\exp(w^o_{i,j})}{\sum_{o' \in O} \exp(w^{o'}_{i,j})}$ is a commonly used function (e.g., [29], [30]) to normalize its input ($w^o_{i,j}$) into some probability $\in [0, 1]$. In the following, we refer to $w^o_{i,j}$ as the $\mathrm{SoftMax}(w^o_{i,j})$ for simplicity.

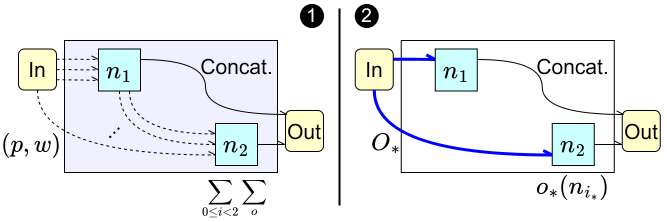

Fig. 2: Train and select the optimal operations in a being designed layer.

As an illustration, Fig. 2 depicts a designed layer of $k = 2$ inner nodes. The input (In) of this layer is regarded as n_0. The output (Out) of this layer is the *concatenation* of all inner nodes (excluding n_0). According to Eq. 1, the neural operations (the dashed edges) are all applied on the nodes in ❶. p is the trainable parameters of operations to process data. Also, an extra trainable weight w is assigned to each operation, denoting the processing importance of the specific operation.

After the update of (p, w) converges (will be discussed in the next Section 5), we select one optimal operation o_* and the connected predecessor node n_{i_*} for n_1 and n_2 in ❷. In detail, this selection is done by finding the highest importance w. That is,

$$(i_*, o_*) = \underset{0 \leq i < j,\ o \in O}{\arg\max}\ w^o_{i,j}. \tag{2}$$

Recall that we define a special *skip* operation in the candidate set. If *skip* is selected, n_j is regarded to be deleted as *skip* reflects that there is no applied operation. By doing so, *skip* empowers Loong to adaptively choose the final number (k') of inner nodes in a layer (i.e., $k' \leq k$ is legal).

5 Layer-wise Training

5.1 Bi-level Gradient

Through the NN design in Loong, our goal is twofold. The first is to select the optimal operations (viewed as directed edges) to connect the nodes. The second is to train the parameters p of these operations.

We solve this via the bi-level gradient descent-based optimization similar to [17]. After dividing the given task data into a training dataset and a validation dataset, w and p in a layer can be optimized with the following two steps:

1) Calculate the cross-entropy loss CE_{val} on the validation data and update all w by descending $\nabla_w CE_{val}$;
2) Calculate CE_{train} on the training data and update all p by descending $\nabla_p CE_{train}$.

Here CE is a common loss used to train NNs [11], [29]. The two steps are executed alternately to minimize the CE_{val} and CE_{train} until the layer is converged (i.e., the loss stops decreasing). During the bi-level gradient descent, p and w of different neural operations are adjusted to fit the data. The operation that contributes the most to reducing the loss (i.e., improving accuracy) is getting a higher w. Finally, only the operation (with trained p) that has the highest importance $w^o_{i,j}$ is selected to be applied between n_i and n_j. In the next section, we will discuss how to adjust this gradient scheme to consider the optimization of device objectives.

5.2 Overall Algorithm

The overall algorithm of the layer-wise update with bi-level gradient is shown in Algorithm 1. The network administrator provides the task data and the device objective z. Then, the task data is divided into the training data D_{train} (for updating p) and the validation data D_{val} (for updating w). Both D_{train} and D_{val} have a pair of (x, y) where x represents the data samples and y indicates the corresponding class labels of the samples. The core ideas of this algorithm are:

- In Lines 2∼4, we first check whether or not the number of layers in the current NN has reached the max value MAX. If so, we terminate the layer-wise training. Otherwise, we continue to initialize a new layer for NN. $\mathrm{OpEmbed}(k, O)$ is the operation embedding discussed in Section 4 and Fig. 2, where all the candidate operations O are the directed edges, connecting the k inner nodes.

Algorithm 1 Layer-wise Training

Input: $D_{train} = (x, y)$, $D_{val} = (x, y)$, and objective z.

1: $NNs = [\,]$;
2: **while** current $NN.layer\_length \leq$ MAX **do**
3: Initialize new $layer$ with OpEmbed(k, O); // Operation embedding of Section 4
4: NN.add_new_layer($layer$);
5: **while** training NOT converged **do**
6: $\hat{y} = NN(D_{val}.x)$;
7: $CE_{val} = CE(\hat{y}, D_{val}.y)$;
8: $OR =$ ORCompute($layer$); // Objective regularization computation of Section 6
9: Update $layer.w$ by $\nabla_w(CE_{val} + OR)$;
10: $\hat{y} = NN(D_{train}.x)$;
11: $CE_{train} = CE(\hat{y}, D_{train}.y)$;
12: Update $\forall p$ in NN by $\nabla_p CE_{train}$;
13: **end while**
14: Eliminate redundant $layer.op$ in NN by $layer.w$;
15: **if** fine-tuned NN improve accuracy **then**
16: NNs.add_new_NN(NN);
17: **end if**
18: Early stop if the accuracy is not improved twice;
19: **end while**
20: $sub\text{-}NN =$ ObjSelect(NNs, z); // Objective-based selection of Section 7

Output: The desired $sub\text{-}NN$.

- Lines 6∼9 update w. We first use NN to predict the class labels $\hat{y}$ on D_{val}. For the layers that have been designed, their inner nodes use the optimized operations to process the data in the prediction, i.e.,

$$n_j = o_*(n_{i_*}). \tag{3}$$

  For the layer being designed, its computation follows the Eq. 1. Then, the loss CE_{val} is derived via computing the difference between the predicted labels $\hat{y}$ and the true labels $D_{val}.y$. Notably, unlike the original bi-level gradient scheme in Section 5.1, we also add OR (will be discussed in next Section 6) to compute the gradient, reflecting the objective influences of different operations on the device.
- Lines 10∼12 update p of NN through descending the gradient on CE_{train}. In this update, parameters of all layers in the current NN are updated, i.e., including the already designed layers, and the currently being designed layer.
- In Line 14, the training of the current NN finishes and we reconstruct the newly added $layer$ by eliminating all the operations except for the optimal one for each inner node (see Fig. 2). According to Eq. 2, in each inner node, the optimal operation has the max importance weight w.
- In Line 16, NNs records all the intermediate NN of various consecutive layers. After the generation ends, we select the optimal $sub\text{-}NN$ in NNs according to the provided objective z in Line 20. Actually, ObjSelect(.) can return $sub\text{-}NN$ for one or multiple devices (it is detailed in Section 7).
- Line 18 is the quick stop for the loop of layer generation. As the NN is generated layer by layer, we should guarantee the necessity of adding more layers. Therefore, after a new layer is obtained, we fine-tune NN on the training set and check its validation accuracy. If adding consecutive two layers does not improve the accuracy, we stop generating more layers.

6 Objective Regularization Computation

6.1 Lookup Table and Proxy Device

Typically, Loong can design NNs for heterogeneous network devices that have varying objective requirements. However, it is time-consuming to interrupt Loong's layer-wise training to wait for the running cost (memory/latency) feedback on the target device [19]. Similar to [18], [41], we also build a lookup table to record the previously tested costs of the candidate neural operations so that the feedback computation in Loong can be fast. To build the lookup table, we implement an automated script (based on standard APIs in [42]) to run on devices, testing and recording the costs of each operation. To make sure that the lookup table can reflect the cost well, each operation is tested 10K times, and then its average cost is recorded.

Nevertheless, given the diverse network devices, it may still take some time for our script to run on each device. Fortunately, as previously reported in [43], operations typically exhibit correlated performance across different devices. As such, with one device as the proxy, we can re-use its lookup table for new target devices without losing optimality.

6.2 Objective Regularization

As many network devices are resource-constrained, the network administrator may request lightweight NNs. Besides, with the same accuracy, we prefer to select operations that are more computationally friendly to reduce power and time overhead. In such cases, we should take the device costs of neural operations into account when updating their importance weights in Loong.

Thus, we utilize OR to embed the potential device costs in the loss. Given the $layer$ being designed, we set

$$\begin{aligned} OR &= \text{ORCompute}(layer) \\ &= \sum_{n_j} \sum_{0 \leq i < j} \sum_{o \in O} w^o_{i,j} \times \text{Lookup}(o), \end{aligned} \tag{4}$$

where Lookup(.) reflects the function to check the operation o in the lookup table and obtain the operation memory/latency cost. Specifically, in Algorithm 1 Lines 8∼9, we first compute OR and then update w with descending $\nabla_w(CE + OR)$. By doing so, we can penalize the high-value w of operations that have high memory or latency costs and guide the NN toward a lightweight design. It is important to note that this procedure is optional. If one demands Loong to design NNs and is not concerned with the potential device constraints, he can set $OR = 0$ in Lines 8 of Algorithm 1. Similar to ProxylessNAS [18], OR is a soft constraint in the training loss. OR can guide the NN to be lightweight, but it cannot reach a NN whose memory/latency $\leq$ the given objective z. As such, the objective-based selection is proposed in the next Section 7 to find the desired NN for each device.

7 Objective-based Selection

7.1 Single Device Scenario

By using the lookup table from the same device, we can obtain an array NNs that maintains multiple $sub\text{-}NNs$ of different consecutive layers in Algorithm 1. Nonetheless, one obtained $sub\text{-}NN$ can have varying memory/latency values on different devices. Our concern is how to recommend the right $sub\text{-}NN$ for a specific memory/latency objective z of a device through $\mathrm{ObjSelect}(.)$ (Line 20, Algorithm 1). Basically, $\mathrm{ObjSelect}(.)$ can be implemented by running all $sub\text{-}NNs$ on the specific device to find the optimality $sub\text{-}NN^*$ such that

$$sub\text{-}NN^* = \underset{sub\text{-}NN \in NNs}{\arg\max} \ \mathrm{Layers}(sub\text{-}NN), \\ \mathrm{Obj}(sub\text{-}NN) \leq z. \tag{5}$$

That is, among all the $sub\text{-}NNs$ that have a lower memory/latency requirement than z, $sub\text{-}NN^*$ is the one that has the most layers (according to Algorithm 1, the NN of more layers is more accurate).

7.2 Multi-Device Scenario

The NN selection becomes tricky as testing all $sub\text{-}NNs$ for each individual device is time-consuming. Due to the neural monotonicity — NNs of more layers are expected to be more accurate but consume more memory/latency, we propose to utilize the binary search for the implementation of $\mathrm{ObjSelect}(.)$ in the multi-device scenario. Fig. 3 shows the example binary search on a given device to find the $sub\text{-}NN$ of specific latency. As illustrated, $sub\text{-}NNs$ of the NNs array (Line 16, Algorithm 1) have been naturally ordered: $sub\text{-}NNs$ on the right side have more layers, better accuracy, and higher latency. During this binary search, we first test the middle 5-layer $sub\text{-}NN$ of the array, testing its actual latency on the device. As the actual latency is larger than z, we continue to search the left half elements in NNs and repeat this until the exact value (or the value closest to z) is found. Finally, we only test two (instead of all) $sub\text{-}NNs$ to find the optimality in this example.

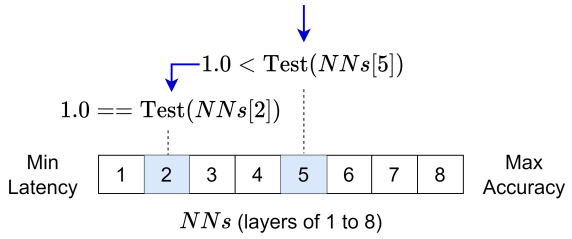

Fig. 3: The binary search example on a device to find the $sub\text{-}NN$ that satisfies a latency $z = 1.0ms$.

In summation, when multiple devices are waiting for being deployed, we can again consider the series of $sub\text{-}NNs$ with different memory/latency requirements without rerunning Loong. This is feasible for deployment devices of the same task. When the task and the training data change, we should run Loong again.

8 Evaluation

8.1 Experimental Settings

Tasks and Datasets. According to Section 2.1, we leverage three networking classification tasks with data of byte streams, domain name strings, and flow statistics: **1)** The UNIBS[3] dataset contains packets of bytes captured in the University of Brescia. We use the UNIBS for the traffic application identification with six application classes. **2)** The 360[4] dataset consists of DNS strings generated by the DGA. We conduct the DGA family classification (18 classes, 17 malicious families plus one benign class) on the 360. **3)** The NB15[5] dataset consists of statistics of flows from benign and nine types of attacks. We conduct the attack detection on the NB15. Each of these datasets is randomly divided into three parts, with 40% of the data used for training, 40% for validation, and 20% for testing.

Baselines and Platforms. Loong is compared with 13 manual and NAS designed NNs: CNN [4], BCNN [7], FFDNN [33], LSTM [31], RNN [5], DeepLSTM [34], LuNet [35], DeepPacket [2], CLSTM [32], and BGRUA [3] are 10 manual approaches; DARTS [17], ProxylessNAS [18], and ENAS [40] are three NAS-based solutions. The 10 manual approaches are reproduced by ourselves with PyTorch 1.10. The NAS baselines are implemented by the Microsoft NNI library[6]. NNI reproduces several NAS solutions according to their papers. We just reset their search spaces to be the same as Loong (Table 1). The training of NNs is conducted on a server with CPU of Intel (R) Xeon (R) Silver 4210 CPU @ 2.20GHz and GPU of NVIDIA RTX 2080 Ti (11GB). To test the multi-objective deployment scenarios of NNs generated by Loong, we further consider three network devices: a virtual machine (VM), the EdgeCore Wedge 100BF-65X[7], and the H3C S9850-32H[8] (as listed in Table 2). The VM (with limited performance settings) serves as a representation of a low-performance router, while EdgeCore and H3C are two commodity programmable switches. The tests of NNs (including Loong and compared schemes) on network devices are facilitated by the use of MNN [42], a lightweight ML library.

Table 2: Devices for the deployment of Loong's NNs.

	System	CPU			RAM
		Type	Clock	Cores	
VM (Virtual Machine)	Ubuntu 16.04	Intel i7-10875H	1.2GHz	4	2GB
EdgeCore (Wedge 100BF-65X)	Open Network 4.14	Intel D-1517	1.6GHz	8	8GB
H3C (S9850-32H)	Open Network 4.14	Intel D-1527	2.2GHz	8	8GB

3. http://netweb.ing.unibs.it/~ntw/tools/traces/
4. http://data.netlab.360.com/
5. https://research.unsw.edu.au/projects/unsw-nb15-dataset
6. https://github.com/microsoft/nni
7. https://www.edge-core.com/productsInfo.php?cls=1&cls2=5&cls3=181&id=334
8. https://www.h3c.com/en/Products_Technology/Enterprise_Products/Switches/Data_Center_Switches/H3C_S9850/

Hyper-parameters of Loong. The max number of inner nodes per layer is $k = 4$, the max number of generated layers is $MAX = 10$, and the candidate operation set O has been discussed in Table 1. Other hyper-parameters (e.g., neural parameter initialization) are the default values suggested by PyTorch. According to the fed task data, our layer-wise training (Algorithm 1) allows Loong to adaptively design neural layers with suitable candidate operations for high accuracy. Hence, we no longer pay more attention to the time-consuming hyper-parameter tuning (experiments in Section 8.2 and 8.3 will show Loong's superiority with the same hyper-parameters on multiple tasks). Though tuning hyper-parameters is time-consuming, it undeniably will also yield further performance gains, and we leave this for future work.

8.2 Classification Performance

As most compared NNs do not consider the device objectives, in this section (and the following Section 8.3, 8.4), we set $OR = 0$ (Lines 8, Algorithm 1) and select the generated highest-accuracy NN (i.e., the last NN in the NNs array) for the comparison with other schemes.

As shown in Fig. 4, the performance on different classification tasks is evaluated in terms of four metrics: average accuracy, precision, recall, and F1-score. Although the considered networking task changes, Loong exhibits superior performance compared to other solutions. As an illustration, when compared with CLSTM on the traffic application identification (in Fig. 4a), Loong improves the accuracy and F1-score by 1.71% (92.90% vs. 94.61%) and 1.80% (92.89% vs. 94.69%), respectively. When compared with ENAS on the DGA family classification (in Fig 4b), Loong improves the accuracy and F1-score by $\sim$3.94% (93.05% vs. 96.99%) and $\sim$4.11% (92.83% vs. 96.94%). A main reason is that after being run on each task, Loong is able to adapt the neural operations to generate optimal NNs, achieving better classification performance.

8.3 Task Generalization

As mentioned previously, there are various networking classification tasks with different data characteristics, which challenges the performance of NNs. To further demonstrate the adaptability of Loong, Fig. 5 illustrates the four classification metrics of each compared solution on different tasks.

As shown, with the exception of Loong, which shows high performance on all tasks due to the design versatility of NNs, achieving 94+% accuracy, the rest of the manual NNs fail to generalize effectively to different tasks. For instance, as shown in Fig. 5a, the accuracy of RNN is unsatisfactory for the DGA family classification task, which is only around 56.09%, a decrease of 40.86% compared to its performance in the attack detection task. Similarly, in Fig. 5c, the recall of BGRUA for the task of traffic application identification is poor, with a value of approximately 42.04%, which is a decrease of 52.10% compared to its recall for the attack detection. The poor performance of these manually designed neural networks highlights the limitations of a fixed neural structure in handling diverse tasks. The ability to automatically design NNs, as demonstrated by Loong, is desirable when facing a range of tasks.

8.4 Training Resource Cost

In this experiment, we examine the average training resource consumption of different schemes. We depict the training time and GPU memory cost in Fig. 6. As shown, Loong takes more time and GPU consumption than manually designed NNs. For instance, Loong consumes 3.93$\times$ more GPU memory than DeepPacket (4.05GB vs. 1.03GB) and takes 7.73$\times$ more time than BGRUA (11.05h vs. 1.43h). This is reasonable, as Loong must evaluate numerous possible neural combinations simultaneously. For a being-designed layer with 4 inner nodes, there are 25 neural operations in Table 1 that can be selected to connect these nodes, resulting in $4! \times 25^4$ possibilities. Besides, Loong will generate a series of NNs instead of only one. On the contrary, handcraft schemes only require training one structurally pre-determined NN. But researchers must invest a lot of manual intervention before getting the determined neural structures, which is much more expensive than the automation in Loong.

When compared with other NAS schemes, Loong still demonstrates advantages. The time and GPU memory of Loong are 50.02% and 40.83% less than DARTS (11.05h vs. 22.09h, 4.05GB vs. 9.92GB). Though ProxylessNAS and ENAS have a lower GPU memory consumption, their classification performance is poorer than Loong (e.g., Fig. 4b).

8.5 Multiple Objectives of Devices

For a fair comparison with other approaches, the previous experiments do not consider device objectives. In this section (and the next Section 8.6), we discuss the effects of the memory/latency objective regularization (OR) in Algorithm 1. Notably, we conduct experiments on network devices of Table 2 and use VM as the proxy device to construct the lookup table for the OR computation in Section 6.

Fig. 7 demonstrates the effects of memory objective regularization (MOR) on Loong's NN generation on the VM device. As shown, the MOR guides Loong to find more lightweight NNs in terms of device memory consumption while preserving better accuracy during the generation of midway NNs. However, the MOR also limits the highest accuracy that NNs can reach. A similar phenomenon is revealed in the use of latency objective regularization (LOR) in Fig. 8. That is, the LOR helps to reduce the device running latency and yields higher accuracy on midway NNs but also limits the final accuracy.

The reason for this interesting phenomenon can be explained by Fig. 9. By adding the OR, device constraints are reflected in Loong's operation selection and guide Loong to prefer simple layers. For example, compared with Fig. 9c, the layer in Fig. 9d has only two inner nodes, and several operations (LeakyReLU, ordinary convolution of kernel 3 and 5) are missed. Though lightened NNs have less memory/latency, their representation capability is weaker which causes the final accuracy reduction in Fig. 7 and 8. Another interesting phenomenon is that the accuracy of Loong either peaks before Layer10 or levels off near Layer10 on all networking tasks (in both Fig. 7 and 8). Hence, fixing the number of possible layers to 10 is reasonable for different networking tasks.

This article has been accepted for publication in IEEE Transactions on Computers. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TC.2023.3333253

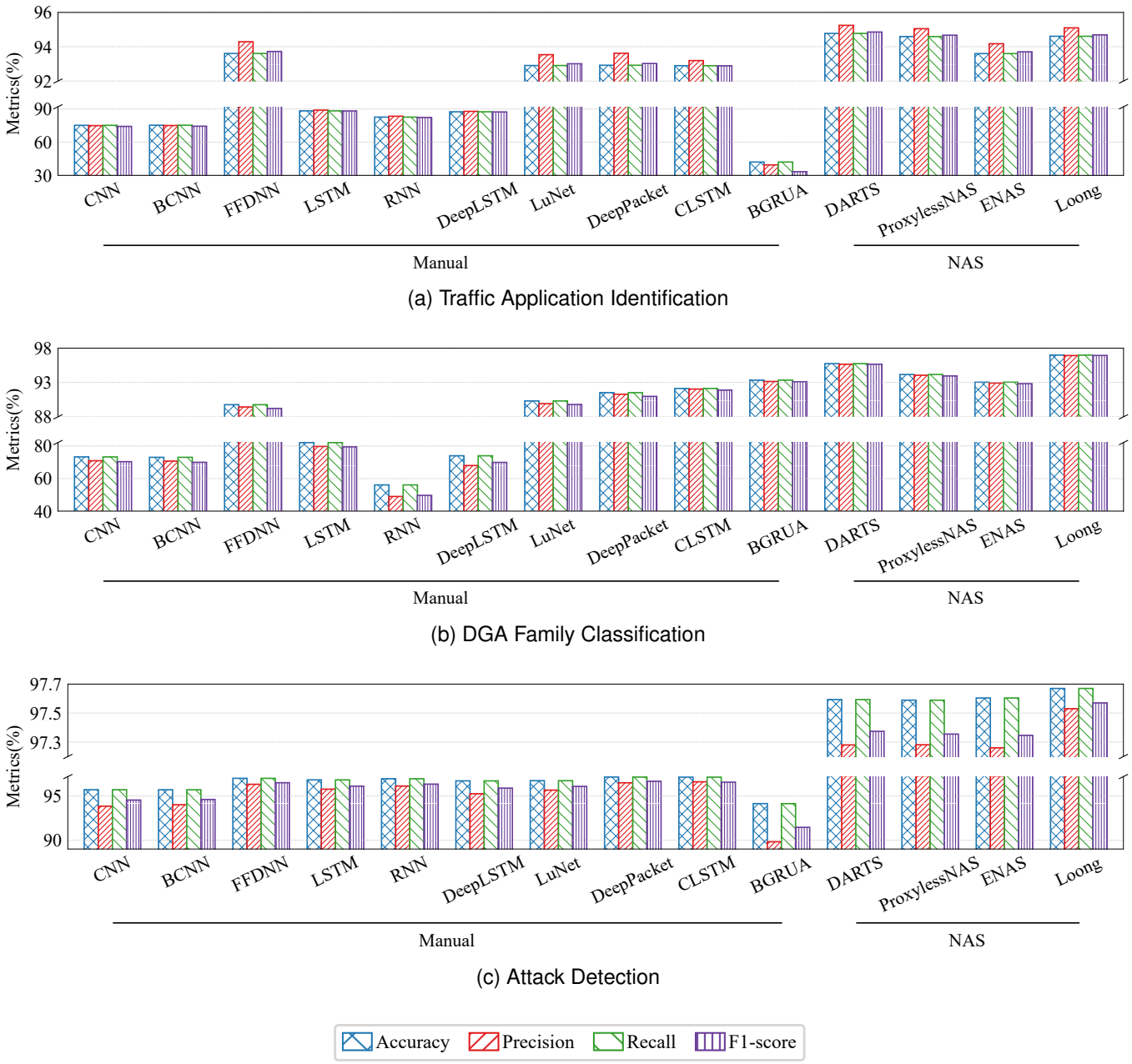

Fig. 4: The average classification performance of different NNs on three tasks.

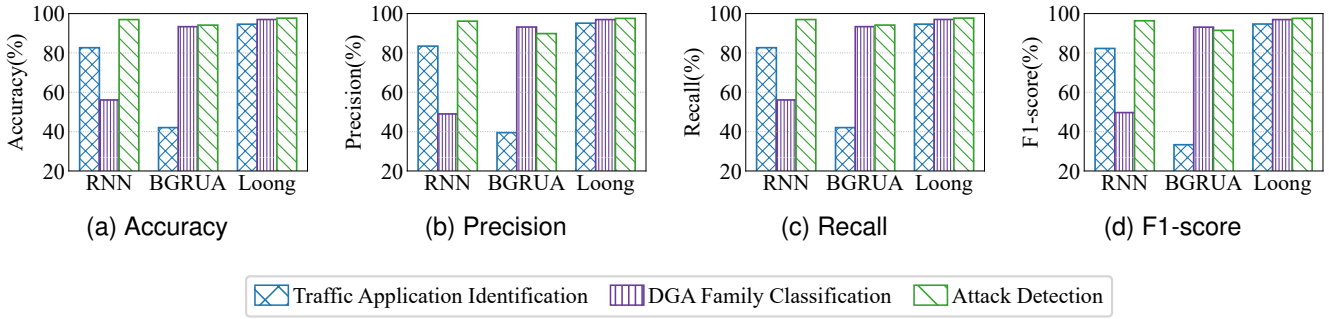

Fig. 5: The generalization performance of handcraft NNs and Loong on different task data.

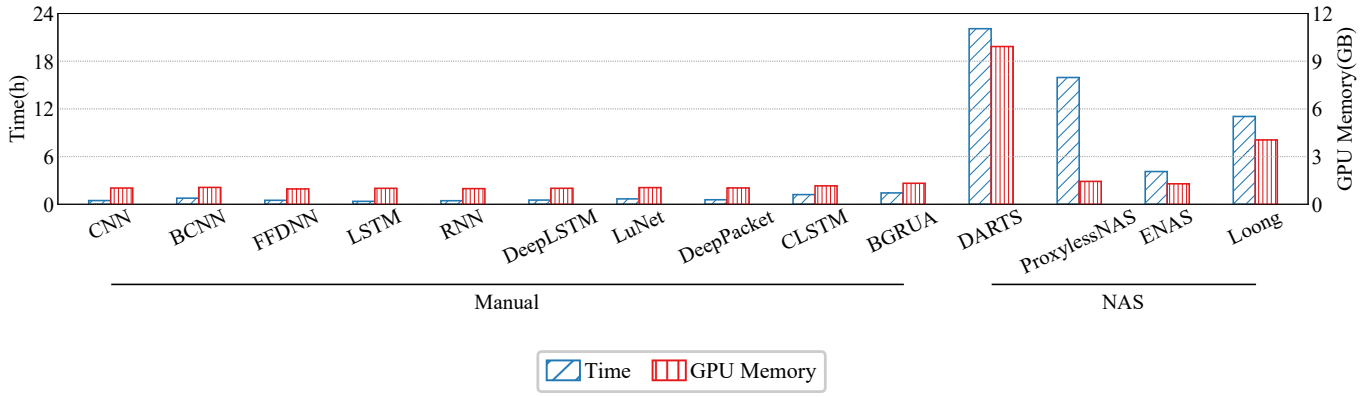

Fig. 6: The average training time and GPU memory cost of the traffic application identification.

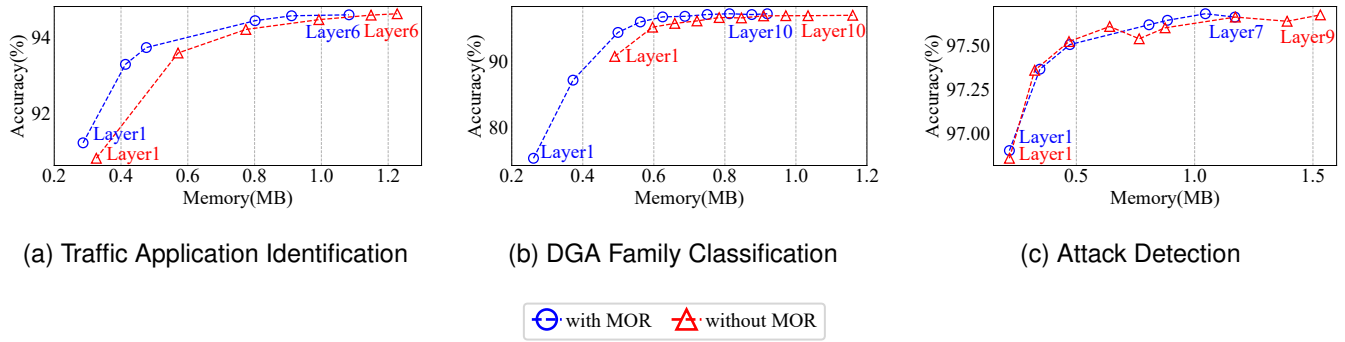

Fig. 7: NNs of different tasks with (or without) memory objective regularization (MOR) on VM. According to Algorithm 1, the NNs (layers) are added one by one. Due to the early stopping in Algorithm 1, the max number of layers can be ≤ 10.

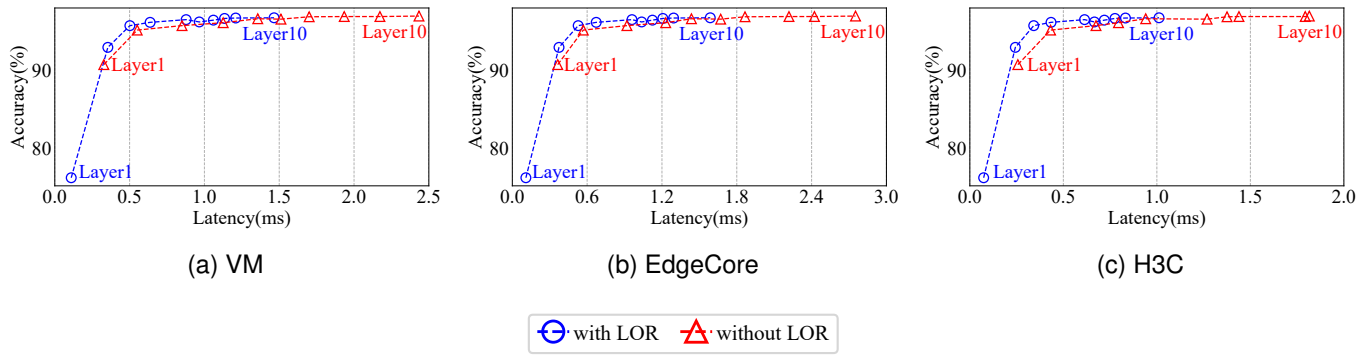

Fig. 8: NNs of DGA family classification with (or without) latency objective regularization (LOR) on three devices. According to Algorithm 1, the NNs (layers) are added one by one.

8.6 Runtime Performance

After Loong generates a series of NNs in the NNs array, the last mission of Loong is to return optimal NNs for a given device objective. In this section, we ask Loong to return NNs that have a max running latency of 0.25ms/0.50ms on the three network devices for the hardware deployment test.

Fig. 10 depicts the tested running rates of the returned NNs and the compared approaches. To satisfy the latency objective of 0.25/0.50ms, the returned NN should classify at least 4000/2000 samples per second on the tested device. As shown, Loong successfully returns the right NN for each device. On the contrary, the compared manual or NAS solutions fail to run with the desired latency on the devices. The results also demonstrate a clear speed advantage of Loong's generated NNs. In Fig. 10b, the NN generated by Loong ($\leq$0.25ms) can classify 9188.98 samples per second on the EdgeCore, which is 7.92$\times$ faster than LuNet (9188.98sample/sec vs. 1159.84sample/sec).

8.7 Experiments on NAS Benchmarks

We further compare Loong with previous NAS solutions on NAS benchmarks of NAS-Bench-201 [44] and HW-NAS-

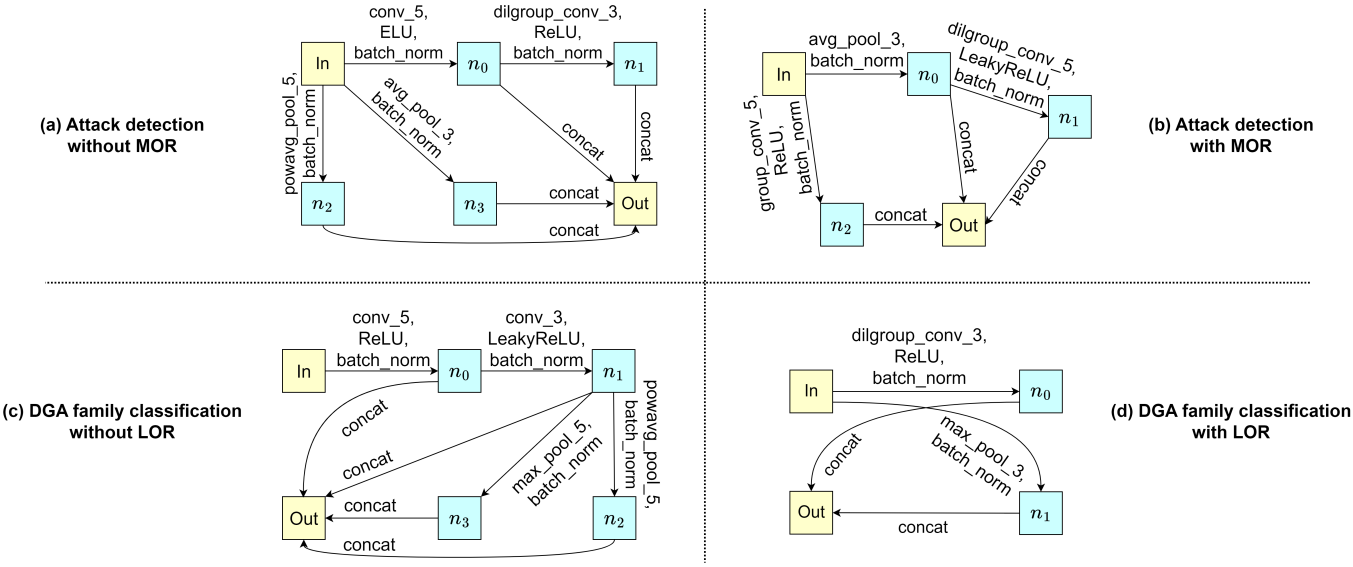

Fig. 9: The layer visualization of different tasks and objective regularization (see Table 1 for operation details).

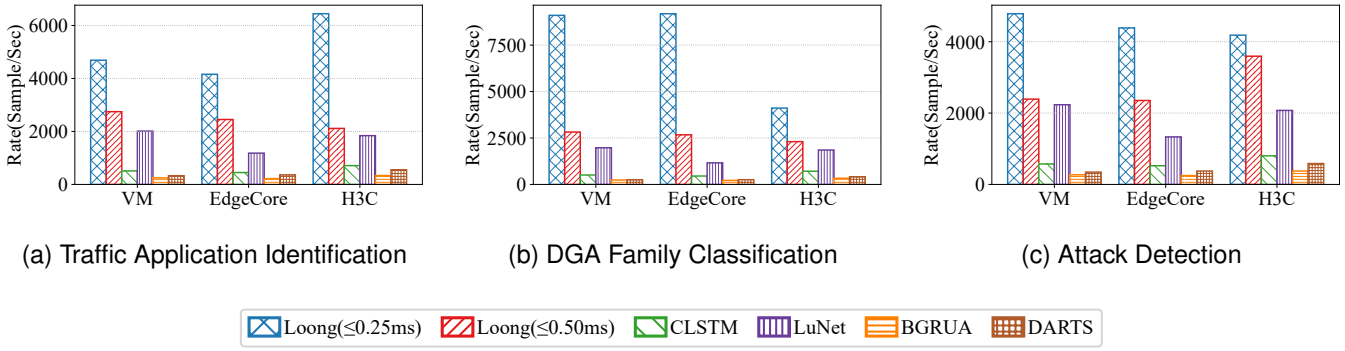

Fig. 10: Inference rates on network devices. The latency objective regularization is used in Loong. We ask Loong to return NNs that have a max running latency of 0.25ms/0.50ms, which indicates the returned NN should classify $\geq$ 4000/2000 samples per second and is as accurate as possible.

Bench [45]. NAS-Bench-201 defines a standard search space (i.e., five candidate neural operations), and several hyper-parameters (learning rate, epoch, optimizer, etc.). Then it reproduces several cell-based NAS solutions (e.g., DARTS, ENAS) to design NNs of 15 cells (layers), reporting NNs' accuracies on image datasets of CIFAR-10, CIFAR-100, and ImageNet. Besides, HW-NAS-Bench provides the latency values of 15-layer NNs generated by NAS-Bench-201 on six devices including Raspberry Pi 4 (Raspi 4), TPU, and ASIC.

To obtain Loong's performance on benchmarks, we make our training settings similar to NAS-Bench-201, and set the max layer $MAX = 15$ in Algorithm 1. In NAS-Bench-201, DARTS and ENAS each generate a 15-layer NN after the NAS training. Differently, our Algorithm 1 will generate 15 NNs (consisting of 1∼15 cells, we denote them as Loong(1)∼Loong(15)), which is more efficient for the multi-device and multi-latency deployment. As Loong's NNs can have different layers instead of 15, we cannot find their reported latencies on HW-NAS-Bench. Hence, we use devices at hand for latency testing, including Raspi 4 (similar to HW-NAS-Bench) and three network devices in previous Table 2.

Table 3: The performance on CIFAR-10.

Solution	#Params (M)	Device Latency (ms)				Accuracy (%)
		Raspi 4	VM	EdgeCore	H3C	
DARTS	0.07#	6.90#	12.69	2.58	1.65	54.30#
ENAS	0.07#	6.90#	12.37	2.56	1.63	53.89#
Loong(15)	0.37	13.15	22.04	4.14	2.66	**91.89**
Loong(1)	0.01	1.82	3.12	0.62	0.40	67.17
Loong*(1)	**0.005**	**0.997**	**1.78**	**0.388**	**0.247**	50.77

#: Values from official NAS-Bench-201 and HW-NAS-Bench

*: Train with latency objective regularization (LOR)

Table 3 shows the performance of NNs generated by Loong and other NAS methods reported on NAS-Bench-201's CIFAR-10 dataset. The NN with 15 cells in Loong (Loong(15)) yields higher accuracy than DARTS and ENAS. Also, when trained with the latency objective regularization, the NN with 1 cell in Loong (Loong*(1)) has the fewest parameters and runs fastest on all tested devices. Additionally, Fig. 11 shows the accuracy of another two datasets in NAS-Bench-201, which also reveals the superiority of Loong.

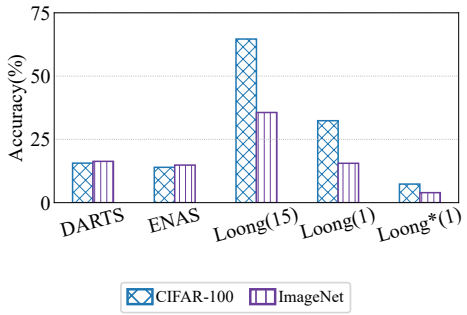

Fig. 11: The classification accuracy on NAS-Bench-201 datasets: CIFAR-100 and ImageNet.

8.8 Discussion on Kernel Sizes

In Loong, we set the kernel size of candidate convolutional operations to be 3 or 5. Actually, several schemes [16], [18], [41] prefer to choose kernel sizes of 3, 5, and 7. Fig. 12 demonstrates the effect of different kernel size choices. On three networking tasks, adding more kernel size choices has limited advantages. For example, after adding the kernel size of 7, the max accuracy achieved by designed NNs on the task of traffic application identification is only improved by 0.02%. On the contrary, the training time is dramatically incremented by 1.65× as Loong has to explore more possible settings when designing the NN.

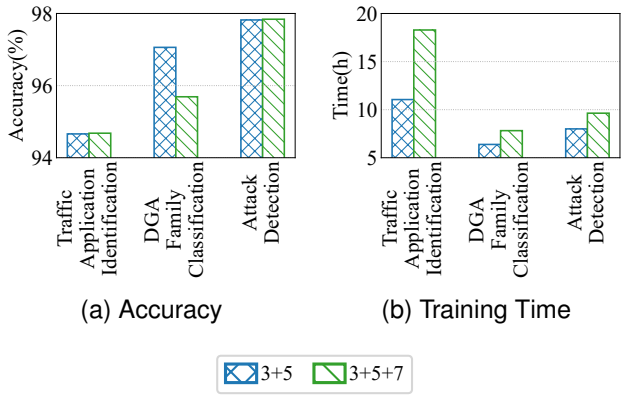

Fig. 12: The kernel size effect in the candidate convolutions.

8.9 Comparison with Computation Mapping

To intuitively demonstrate the superiority of Loong over the manual NN design, we assume that the manual designer can try to map each designed NN on the device to find the desired one. As the max layer of Loong is 10, and each layer has 25 candidate operations to connect 4 inner nodes, the possible NNs in our search space are $\sum_{i=1}^{10}(4! \times 25^4)^i$. It is hard for the manual designer to map and test every NN in this space on the device, so we randomly select 10K NNs for the manual mapping. By our testing on the device EdgeCore (in Table 2), there are multiple candidate NNs meet different latency requirements, i.e., 1938 NNs $\leq$ 0.5ms, 4218 NNs $\leq$ 1.0ms, and 6380 NNs $\leq$ 1.5ms.

The next step for the manual designer is to train these candidate NNs and select the one with the highest accuracy.

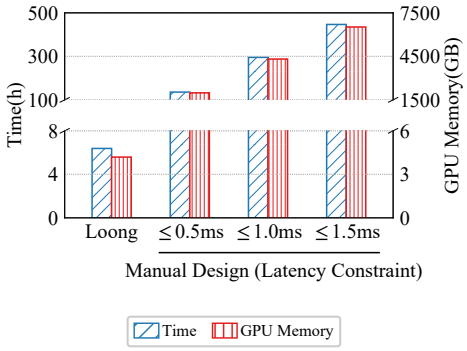

Fig. 13: The accumulated training time and GPU memory usage of Loong and manual design.

Though manually checking each NN's accuracy may find a better NN than Loong, the training time and GPU costs are expensive. Fig. 13 depicts the accumulated training time and GPU memory for Loong and manual design. As shown, manual design has huge time and memory costs. For example, to select the NN $\leq 0.5ms$, Loong is 21.26× $\downarrow$ (in time) and 473.09× $\downarrow$ (in memory) than the manual design.

9 Conclusion

Though neural networks have been applied to several networking classification tasks with promising accuracies, given the diversity of networking tasks and devices in a network, designing suitable neural networks remains a challenging problem for researchers. In this paper, we introduce Loong, a generic automated system that designs and trains NNs for various networking classifications with multiple device objectives (e.g., memory/latency constraints). We embed sufficient 25 neural operations in the layer to be designed to face the task and data diversity. The NN design follows a layer-wise strategy to reduce the GPU computation. Also, the memory and latency costs of deployment devices are reflected in Loong by the objective-regularized training and selection of NNs. Thorough experiments on tasks with different data show the superiority of Loong, e.g., improving the accuracy and F1-score by 3.94% and 4.11% when compared with other NNs. Besides, the deployment evaluation of heterogeneous devices demonstrates that the fastest NN generated by Loong can classify 16438.18 samples per second.

Acknowledgments

This work is supported by National Key Research and Development Program of China under Grant 2020YFB1804704, National Natural Science Foundation of China under Grant No. 61972189, the Shenzhen Key Lab of Software Defined Networking under Grant No. ZDSYS20140509172959989, and the China Scholarship Council (CSC202306210169).

References

[1] J. Zhao, X. Jing, Z. Yan, and W. Pedrycz, "Network traffic classification for data fusion: A survey," *Information Fusion*, vol. 72, pp. 22–47, 2021.

[2] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, "Deep packet: a novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.

[3] X. Liu, J. You, Y. Wu, T. Li, L. Li, Z. Zhang, and J. Ge, "Attention-based bidirectional GRU networks for efficient HTTPS traffic classification," *Information Sciences*, vol. 541, pp. 297–315, 2020.

[4] R. Ahmad, I. Alsmadi, W. Alhamdani, and L. Tawalbeh, "A comprehensive deep learning benchmark for iot IDS," *Computers & Security*, vol. 114, p. 102588, 2022.

[5] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. P. Turnbull, "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset," *Future Generation Computer Systems*, vol. 100, pp. 779–796, 2019.

[6] X. Xiao, D. Zhang, G. Hu, Y. Jiang, and S. Xia, "CNN-MHSA: A convolutional neural network and multi-head self-attention combined approach for detecting phishing websites," *Neural Networks*, vol. 125, pp. 303–312, 2020.

[7] C. Liu, L. Dai, W. Cui, and T. Lin, "A byte-level CNN method to detect DNS tunnels," in *Proceedings of the 38th International Performance Computing and Communications Conference*. IEEE, 2019, pp. 1–8.

[8] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *Proceedings of the 25th Annual Network and Distributed System Security Symposium*. The Internet Society, 2018.

[9] 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 *Proceedings of the 41st International Symposium on Reliable Distributed Systems*. IEEE, 2022, pp. 225–236.

[10] J. Lin, W. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "Mcunet: Tiny deep learning on iot devices," in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2020.

[11] X. Wang, S. Chen, and J. Su, "App-net: A hybrid neural network for encrypted mobile traffic classification," in *Proceedings of the 39th Conference on Computer Communications, Workshops*. IEEE, 2020, pp. 424–429.

[12] T. A. Tuan, H. V. Long, and D. Taniar, "On detecting and classifying DGA botnets and their families," *Computers & Security*, vol. 113, p. 102549, 2022.

[13] S. S, V. R., M. Alazab, and K. P. Soman, "Network flow based iot botnet attack detection using deep learning," in *Proceedings of the 39th Conference on Computer Communications, Workshops*. IEEE, 2020, pp. 189–194.

[14] B. Arzani, K. Hsieh, and H. Chen, "Interpretable feedback for automl and a proposal for domain-customized automl for networking," in *Proceedings of the 20th Workshop on Hot Topics in Networks*. ACM, 2021, pp. 53–60.

[15] J. Holland, P. Schmitt, N. Feamster, and P. Mittal, "New directions in automated traffic analysis," in *Proceedings of the Conference on Computer and Communications Security*. ACM, 2021, pp. 3366–3383.

[16] K. T. Chitty-Venkata and A. K. Somani, "Neural architecture search survey: A hardware perspective," *ACM Computing Surveys*, vol. 55, no. 4, pp. 78:1–78:36, 2023.

[17] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," in *Proceedings of the 7th International Conference on Learning Representations*. OpenReview.net, 2019.

[18] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," in *Proceedings of the 7th International Conference on Learning Representations*. OpenReview.net, 2019.

[19] X. Luo, D. Liu, H. Kong, S. Huai, H. Chen, and W. Liu, "Lightnas: On lightweight and scalable neural architecture search for embedded platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 6, pp. 1784–1797, 2023.

[20] Y. Wu, Y. Gong, P. Zhao, Y. Li, Z. Zhan, W. Niu, H. Tang, M. Qin, B. Ren, and Y. Wang, "Compiler-aware neural architecture search for on-mobile real-time super-resolution," in *Proceedings of the 17th European Conference on Computer Vision*, vol. 13679. Springer, 2022, pp. 92–111.

[21] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *Proceedings of the 4th International Conference on Learning Representations*, 2016.

[22] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37. JMLR.org, 2015, pp. 448–456.

[23] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[24] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," in *Proceedings of the 4th International Conference on Learning Representations*, 2016.

[25] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the Twenty-Seventh International Conference on Machine Learning*. Omnipress, 2010, pp. 807–814.

[26] Y. Boureau, F. R. Bach, Y. LeCun, and J. Ponce, "Learning mid-level features for recognition," in *Proceedings of the Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, 2010, pp. 2559–2566.

[27] M. Lin, Q. Chen, and S. Yan, "Network in network," in *Proceedings of the 2nd International Conference on Learning Representations*, 2014.

[28] J. B. Estrach, A. Szlam, and Y. LeCun, "Signal recovery from pooling representations," in *Proceedings of the 31th International Conference on Machine Learning*, vol. 32. JMLR.org, 2014, pp. 307–315.

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

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

[31] S. Chen, B. Lang, H. Liu, D. Li, and C. Gao, "DNS covert channel detection method using the LSTM model," *Computers & Security*, vol. 104, p. 102095, 2021.

[32] J. Namgung, S. Son, and Y. Moon, "Efficient deep learning models for DGA domain detection," *Security and Communication Networks*, vol. 2021, pp. 8 887 881:1–8 887 881:15, 2021.

[33] S. M. Kasongo and Y. Sun, "A deep learning method with wrapper based feature extraction for wireless intrusion detection system," *Computers & Security*, vol. 92, p. 101752, 2020.

[34] J. Ashraf, A. D. Bakhshi, N. Moustafa, H. Khurshid, A. Javed, and A. Beheshti, "Novel deep learning-enabled LSTM autoencoder architecture for discovering anomalous events from intelligent transportation systems," *Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4507–4518, 2021.

[35] P. Wu and H. Guo, "Lunet: A deep neural network for network intrusion detection," in *Proceedings of the Symposium Series on Computational Intelligence*. IEEE, 2019, pp. 617–624.

[36] M. AlSabah, K. S. Bauer, and I. Goldberg, "Enhancing tor's performance using real-time traffic classification," in *Proceedings of the Conference on Computer and Communications Security*. ACM, 2012, pp. 73–84.

[37] 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*, vol. 11218. Springer, 2018, pp. 122–138.

[38] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proceedings of the 5th International Conference on Learning Representations*. OpenReview.net, 2017.

[39] R. Lyu, M. He, Y. Zhang, L. Jin, and X. Wang, "Network intrusion detection based on an efficient neural architecture search," *Symmetry*, vol. 13, no. 8, p. 1453, 2021.

[40] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 2018, pp. 4092–4101.

[41] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," in *Proceedings of the 8th International Conference on Learning Representations*. OpenReview.net, 2020.

[42] 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.

[43] B. Lu, J. Yang, W. Jiang, Y. Shi, and S. Ren, "One proxy device is enough for hardware-aware neural architecture search," in *Pro-*

ceedings of the International Conference on Measurement and Modeling of Computer Systems. ACM, 2022, pp. 55–56.

[44] X. Dong and Y. Yang, "Nas-bench-201: Extending the scope of reproducible neural architecture search," in *Proceedings of the 8th International Conference on Learning Representations*. OpenReview.net, 2020.

[45] C. Li, Z. Yu, Y. Fu, Y. Zhang, Y. Zhao, H. You, Q. Yu, Y. Wang, C. Hao, and Y. Lin, "Hw-nas-bench: Hardware-aware neural architecture search benchmark," in *Proceedings of the 9th International Conference on Learning Representations*. OpenReview.net, 2021.

Shengpeng Ji received the B.S. degree in Software Engineering from Jilin University, Changchun, China in 2023. He is currently working toward the M.S. degree in Software Engineering at the School of Zhejiang University, Hangzhou, China. His research interests include Speech, Diffusion, Multimodal Fusion and AutoML.

Guorui Xie received his B.S degree from Sun Yat-sen University (Guangzhou, China) in 2019. Now he is a Ph.D. candidate for computer science and technology at Tsinghua University. His main research interests are related to the computer network, including in-network intelligence, programmable switch development, and networking classification tasks based on machine learning.

Yong Jiang received the B.S. degree (1998) and the Ph.D. degree (2002) from Tsinghua University, Beijing, China, both in computer science and technology. He is currently a full professor at the Graduate school at Shenzhen, Tsinghua University. His research interests include the future network architecture, the Internet QoS, software defined networks, network function virtualization, etc.

Qing Li received the B.S. degree (2008) from Dalian University of Technology, Dalian, China, the Ph.D. degree (2013) from Tsinghua University, Beijing, China; both in computer science and technology. He is currently an associate researcher at Peng Cheng Laboratory, China. His research interests include reliable and scalable routing of the Internet, software defined networks, network function virtualization, in-network caching/computing, intelligent self-running network, etc.

Zhenhui Yuan is a Senior Lecturer at Northumbria University, UK. He received his B.degree in Software Engineering at Wuhan University, China, in 2008 and PhD in Electronic Engineering at Dublin City University, Ireland, in 2012. His research interests lie in wireless communication systems for robots and vehicles.

Zhenning Shi received the B.S. degree in Computer Science and Technology from Jilin University, Changchun, China, in 2023. He is currently pursuing the master degree in Computer Technology from Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen, China. His research interests include machine learning for network and in-network intelligence.

Lianbo Ma (M'16) received the B.Sc. degree in Communication Engineering and M.Sc. degree in Communication and Information System from Northeastern University, Shenyang, China, in 2004 and 2007 respectively, and the Ph.D. degree from University of Chinese Academy of Sciences, China, in 2015. He is currently a Professor of Northeastern University, China. His current research interests include computational intelligence and machine learning.

Hanbin Fang is pursuing the B.S. degree in Tang Aoqing Honors Program in Science (Computer Science and Technology) from Jilin University, Changchun, China. His research interests include Computer Network Technology and Deep learning.

Mingwei Xu is a Full Professor with the Department of Computer Science and Technology, Tsinghua University. He has chaired or participated in more than 30 research projects and published over 200 papers. His research interests include computer network architecture, Internet routing, and cybersecurity. He is the Winner of National Science Foundation for Distinguished Young Scholars of China. He has served as the TPC Chair or a member for several conferences including ICPP, Infocom, GLOBECOM, and ICC.