Empowering In-Network Classification in Programmable Switches by Binary Decision Tree and Knowledge Distillation

Guorui Xie, Qing Li, *Member, IEEE,* Guanglin Duan, Jiaye Lin, Yutao Dong, Yong Jiang, *Member, IEEE,* Dan Zhao, and Yuan Yang

***Abstract*—Given the high packet processing efficiency of programmable switches (e.g., P4 switches of Tbps), several works are proposed to offload the decision tree (DT) to P4 switches for in-network classification. Although the DT is suitable for the match-action paradigm in P4 switches, it has several limitations. First, the range match rules used in the DT may not be supported across diverse hardware configurations. Second, the current DT implementation in switches consumes excessive resources such as switch stages and memory. Moreover, some emerging sophisticated ML models, including neural networks (NNs) and ensemble ML models, have shown their superior performance for networking tasks. However, their high computational complexity and large storage requirements pose new challenges to the deployment of these models in switches.**

In this paper, we propose Mousikav2, a knowledge distillation-based in-network classification solution that addresses these drawbacks successfully. First, we design a new tree model, i.e., the binary decision tree (BDT). Unlike the DT, our BDT consists of classification rules in the form of bits, which is a good fit for the standard ternary match used in different switches. Second, we introduce a teacher-student knowledge distillation architecture in Mousikav2, which enables the general transfer from other sophisticated models to the BDT. Through this transfer, sophisticated models are indirectly deployed in switches to avoid switch constraints. Finally, we develop a lightweight P4 program to perform classification tasks in switches with the BDT after knowledge distillation. Experiments on three networking tasks and three commodity switches show that Mousikav2 not only improves the classification accuracy by 1.93%, but also reduces the switch stage and memory usage by 2.00× and 17.70×, respectively.

***Index Terms*—In-network classification, programmable switch, decision tree, knowledge distillation.**

I. Introduction

RECENT years have witnessed an emerging trend of applying machine learning (ML) to many networking classification tasks [1], [2]. For example, in [3], the authors propose a scheme to classify flows into mice or elephants (i.e., flow size prediction) by utilizing ML models such as Gaussian process regression and neural networks (NNs). In [4], the authors propose a convolutional NN to classify packets into the application types (e.g., browsing, file transferring) that generated them. Thus, a customized routing can be provided to guarantee the differentiated QoS requirements. In [5], ML is also introduced into the malware detection and the authors present a neural framework, FFDNN, to analyze the input traffic and detect malware attacks.

Traditionally, these ML models are implemented in x86 servers to support their complicated ML operations (e.g., floating-point multiplications and divisions). Especially, the computation-intensive NNs even require specific GPU-equipped servers for acceleration. As such, traffic has to be redirected to specialized servers for further processing. Although these ML solutions provide promising accuracy, they can cause unsatisfactory throughput and capacity for large-scale data centers and cloud networks [6].

Compared with x86 servers, the modern programmable switches (e.g., P4 switches [7]) support up to Tbps of throughput and enable the programmable logic, which offers the opportunity of deploying in-network ML models. Nonetheless, such a high throughput is achieved by sacrificing the programmable flexibility. Only simple instructions like integer additions and bit shifts are allowed in switch actions. These actions also should be triggered by the predefined match rules, i.e., the match-action paradigm. Besides, each switch has compact resources (e.g., memory and the number of stages) for programmable processing. These programmable limitations hinder the deployment of sophisticated ML models in switches' data plane for throughput gains. For example, the floating-point operations in NNs cannot be accomplished through the match-action paradigm and instructions like integer addition and bit shifts. Moreover, these models usually require massive storage which can not be provided by switches [8].

To tackle these switch limitations, some works [9]–[11] are proposed to deploy the simple rule-based ML model, i.e., the decision tree (DT), in P4 switches for high-speed in-network classification (aka in-network intelligence). For instance, in [9], the authors propose IIsy, which utilizes several

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, and the Shenzhen Key Lab of Software Defined Networking under grant No. ZDSYS20140509172959989. *(Corresponding author: Qing Li)*

Guorui Xie, Guanglin Duan, Yutao Dong, 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-mail: xgr19@mails.tsinghua.edu.cn; duangl16@tsinghua.org.cn; dyt20@mails.tsinghua.edu.cn; jiangy@sz.tsinghua.edu.cn)

Qing Li and Dan Zhao are with the Peng Cheng Laboratory (PCL), Shenzhen 518066, China. (e-mail: liq@pcl.ac.cn; zhaod01@pcl.ac.cn)

Jiaye Lin is with the Tsinghua Shenzhen International Graduate School, Shenzhen 518055, China. (e-mail: lin-jy22@mails.tsinghua.edu.cn)

Yuan Yang is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. (e-mail: yangyuan_thu@mail.tsinghua.edu.cn)

programmable stages and lots of range match rules to offload a DT in the switch for the IoT traffic classification by device type. Although the DT seems to fit the switches' match-action paradigm, there are still significant challenges that remain unsolved:

- The range match used in the DT is not widely supported. While diverse software/hardware programmable switches following different P4 standards [12], [13], the range match is not defined in the core library that must be strictly complied with by various manufacturers.
- The learning capability of the rule-based DT is not as powerful as other sophisticated models (e.g., NNs and ensemble models), resulting in inferior accuracies [14].
- The current DT-P4 implementation consumes lots of resources such as switch stages and memory. Notably, these resources are precious in switches and should be carefully shared among different basic networking tasks (e.g., fault tolerance [15] and load balancer [16]).

Therefore, we propose Mousikav2[1] in this paper, which makes a further step towards enhancing the performance of in-network classification. Mousikav2 supports deployment in switches of different P4 standards, enhances rule-based classifiers with competitive accuracy similar to more sophisticated ML models, and requires fewer programmable resources than previous solutions. In summation, **we address the aforementioned challenges with the following key ideas**:

- We redesign the DT to a new rule-based model, the *binary decision tree* (BDT), whose classification rules are bits and thus can be efficiently encoded into the ternary match. The ternary match is defined in the standard core.p4 library and is well supported by most P4 devices.
- We adopt a teacher-student *knowledge distillation* architecture to train the BDT. The knowledge of sophisticated teacher models such as NNs is transferred into the BDT through this architecture. In other words, we distill the powerful knowledge into the BDT to avoid deployment constraints in switches, while improving the classification accuracy in Mousikav2.
- We design a delicate *P4 program* to use the classification rules of the BDT. This program only takes up two tables and two stages of the switch, which is lightweight enough to cooperate with other networking tasks and share the compact switch resources.

We conduct thorough experiments on three classification scenarios (flow size prediction, traffic type classification, and malware detection) and three commodity P4 switches to evaluate the performance of Mousikav2. The experimental results reveal that: **1)** The BDT after knowledge distillation usually has a better classification performance. For the task of traffic type classification, the BDT after knowledge distillation improves the accuracy of the DT by 1.93% (97.90% vs. 95.97%) **2)** Knowledge distillation is helpful to reduce the training time and classification rules of the BDT. E.g., the training time and the number of classification rules are reduced by 1.60× and 4.20×, respectively. **3)** Due to the efficient processing performance of the hardware switch, deploying Mousikav2 in the three switches has little impact on their packet forwarding. For traffic speeds of 10Gbps and 100Gbps, the traffic can still be transmitted at line rate (the packet latency < 550 nanoseconds). **4)** Compared with the existing DT implementation, Mousikav2 only occupies a small amount of switch resources. For the malware detection task, Mousikav2 only takes up 2 stages and 1.00% of the TCAM, which is 2.00× and 17.70× less than IIsy [9].

The rest of this paper is organized as follows. We first provide a background of the ML-based networking classification, P4 switch, and in-network classification in Section II. Then section III presents the Mousikav2 overview. Section IV, V, and VI introduce the details of Mousikav2 (i.e., BDT, knowledge distillation, and P4 program). Section VII discusses the experimental results. Last, we conclude this paper in Section VIII.

II. Background

A. Machine Learning for Networking Classification

In recent years, machine learning has been employed in every possible field to leverage its amazing learning power, e.g., computer vision and natural language process [18]–[21]. The networking field has also seen several schemes proposed to exploit ML for networking classification tasks [1].

In [3], the authors concern with the problem of predicting the size of a flow and detecting elephant flows (very large flows). They describe the problem as a learning-based classification task and employ machine learning models like Gaussian process regression (GPR) and neural network (NN) for flow size prediction. In [4], the authors focus on the problem of classifying Internet traffic by application type. They designed an NN-based system that can classify IP packets into application protocols (e.g., FTP and P2P) or applications (e.g., web browsing and file transferring). The works in [22], [23] also focus on employing the power of ML for the traffic classification, the employed models are recurrent NN and one-dimensional convolutional NN, respectively. In [5], the authors propose a feed-forward deep NN (FFDNN) and utilize 48 statistical features of flows (e.g., the average packet sending rate and the inter-packet arrival time) for detecting malware attacks. In [14], the authors utilize different numbers of the fully-connected neural layers to build two models, i.e., ANN and DNN. Both models show superiority in the task of malware detection.

Now, many ML-based solutions have reached promising accuracy on networking classification tasks [2]. Based on them, network administrators can yield many management and security gains, e.g., offering differentiated QoS provision by the traffic type classification and defending against attacks by the malware detection. Nevertheless, the current solutions of ML deployment require high-performance x86 servers to support the complicated ML operations. Not to mention that computation-intensive NNs even rely on specific GPUs for acceleration. Hence, traffic must be redirected to specialized servers for analysis, which induces significant processing latency and severely degrades throughput in large-scale commercial networks [6].

[1]This work was presented in part at the 2022 IEEE International Conference on Computer Communications (INFOCOM) [17].

B. P4 Switch and Its Constraints

Recently, P4 switches of Tbps have been deployed in several commercial data centers [24], [25]. Besides the high throughput, P4 switches also support the programmable logic, which capacitates the direct deployment of ML models inside network devices.

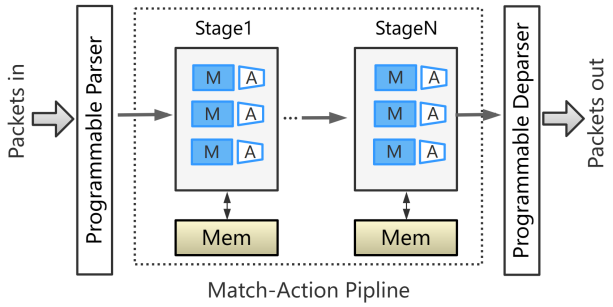


Fig. 1: The general architecture of a P4 switch.

There exist various software and hardware switches that follow different P4 standards [12], [13]. Fig. 1 demonstrates the general architecture of a switche' data plane, i.e., the protocol independent switch architecture (PISA). As shown, the arriving packet is first mapped into a packet header vector (PHV) by the programmable parser. Then, the PHV is passed to a match-action pipeline. The pipeline consists of match-action units (MAUs) arranged in N stages (typically, $N = 12$). Each stage is assigned a memory block (mem). If a header field (e.g., the destination port) in the PHV matches (M) a given table entry (stored in mem), further processes in the action unit (A) associated with the matched entry are triggered. Finally, the processed PHV is reorganized into a packet by the programmable deparser. In summary, PISA allows network administrators to define customized processes (e.g., tables and actions) in P4 language and then instantiate them inside MAUs. This novel architecture empowers P4 switches with a packet processing throughput of Tbps. However, a shortcoming of the architecture is that it has rather limited programmable capability. Now we discuss three main programmable constraints below.

Processing constraints. To guarantee the high-speed processing, complex instructions such as multiplications, divisions, and other floating-point operations (e.g., polynomials or logarithms) are not supported [9], [26]. Packet processing in MAUs is limited to very simple instructions like integer additions and bit shifts. Besides, the number of supported instructions in a specific action is also limited. As a result, it is difficult to deploy most ML models that require complicated floating-point operations in P4 switches.

Matching constraints. For P4 language, the standard library (i.e., core.p4 [27]) defines three kinds of matching which are widely supported by diverse devices: **1)** Exact match. The input key (e.g., headers in the PHV) has to match exactly with the field in the rule. **2)** Ternary match (wildcard). The input key is AND with the Mask associated with each rule, and then compared with a corresponding Value for an exact match. **3)** Longest prefix match (LPM). Compared with the ternary match, this case guarantees that the Mask is a series of consecutive bits 1 followed by a series of consecutive bits 0 [28]. Other libraries (e.g., v1model.p4 [12]) may define additional match kinds such as range match and fuzzy match. But these additional match kinds may not be available in many switch targets [9].

Resource constraints. Each stage is evenly equipped with two high-speed types of memory. One is TCAM, which is a content-addressable memory suitable for fast table lookups. TCAM is used to store table entries with match kinds including ternary, LPM, and range match [29]. The other is SRAM which is used to store exact match table entries and stateful registers. Unfortunately, the total amount of memory in the switch is small. The amount of SRAM is in the order of 100MB, while TCAM is far less than that [30]. Furthermore, the number of available switch stages is also small (typically 12 [9]) as passing too many stages will delay packet forwarding. As such, many networking tasks (e.g., NAT, fault tolerance, and load balancing) have to compete to share these precious resources.

C. In-Network Classification with the Decision Tree

Given the discussed switch constraints, an emerging trend is to implement the rule-based ML model, decision tree (DT) [31], in switches for high-speed processing, i.e., in-network classification (also known as in-network intelligence).

The authors in [9] propose a framework, namely IIsy, which maps a DT into table entries of switch tables for the IoT traffic classification by device type. In IIsy, a table takes up one switch stage. The total number of stages is equal to the number of features used as the DT input plus one. At each table/stage, IIsy compares a feature with all candidate thresholds in the DT by the range match. Then, the matched result is encoded into the metadata field in the PHV, indicating the branch taken in the tree. The last stage retrieves all the encoded values from the metadata and maps (matches) the values to the leaf nodes, obtaining the classification results. In addition to IIsy, there also exist several works that deploy the DT in switches. In [10], the authors propose SwitchTree for the DT-P4 conversion. SwitchTree maps every DT level to a single P4 table. Similar to IIsy, a table in SwitchTree also occupies one switch stage. Once a PHV passes the P4 program in SwitchTree, the matched results of the previous table are encoded into the metadata fields in the PHV and act as the match key for the next table. In [11], [32], the authors also develop P4 programs to speed up networking classification tasks based on DTs.

Although the widely used DT is without floating-point operations and fits the switch processing constraints, we argue that there remain some limitations in the current DT-P4 implementation. For instance, the range match used in the DT is not one of the standard match types in P4, which may not be supported by some P4 switches. Additionally, the implementation consumes lots of switch resources (e.g., stages and memory), leaving little resources for a switch to take on other networking responsibilities at the same time. These limitations motivate us to design Mousikav2, a new in-network classification approach that not only is well supported by different switches but also improves the performance

(e.g., resource consumption and classification accuracy) of in-network classification.

III. Mousikav2 Overview

Fig. 2 presents the Mousikav2 framework. It mainly contains three key components: the binary decision tree (BDT), the teacher-student knowledge distillation architecture to train the BDT, and the P4 program to install the trained BDT.

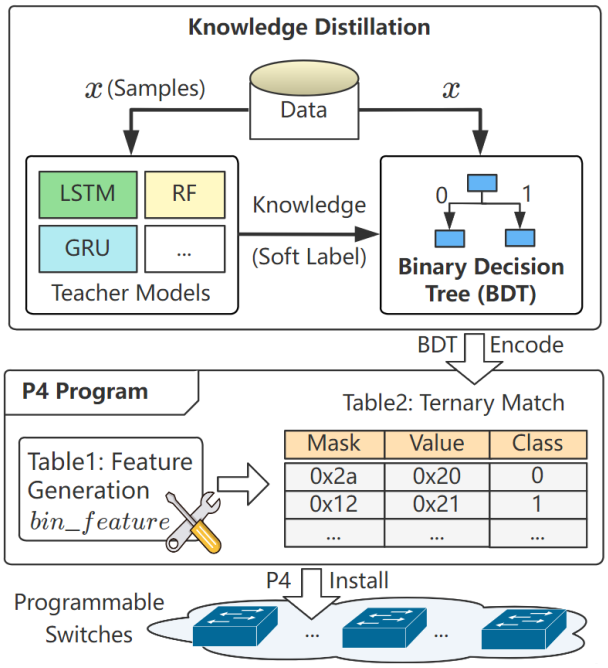


Fig. 2: The Mousikav2 framework.

The BDT (detailed in Section IV) has many inner and leaf nodes. Each inner node checks one bit of the sample features and routes the sample to the left subnode (if the bit value is 0) or the right subnode (if the bit value is 1). When the sample is routed to a leaf node, the predicted class label is obtained. **The knowledge distillation (detailed in Section V)** leverages sophisticated ML models to train (teach) a BDT. Many powerful models can be employed as the teacher model, including but not limited to different NNs (e.g., LSTM [33] and GRU [34]) and ensemble models (e.g., RF [35]). The teacher model is trained in advance. Then, for each sample x in the dataset, the teacher model transfers its learned knowledge (aka soft label) to the BDT, supervising the growth of inner and leaf nodes. **The P4 program (detailed in Section VI)** mainly consists of two match-action tables (two separate switch stages between the programmable parser and deparser). Table1 has no table entries and its default action is to map the features (e.g., ports, packet size in the PHV) of a packet into the bit string $bin\_feature$. Then, Table2 classifies $bin\_feature$ according to the ternary match entries (i.e., encoded rules of the BDT). Each entry contains three fields, i.e., Mask, Value, and Class. As discussed in Section II-B, the ternary match is performed by finding the matched Value after $bin\_feature$ AND Mask and returning the corresponding Class label.

With the three delicate components, Mousikav2 perfectly fits all mentioned P4 switch constraints. First, Mousikav2 performs classification tasks by the rule-based BDT and avoids the complicated floating-point instructions in other ML models (processing constraints). Second, the rules of the BDT can be encoded into the ternary match table entries. As the ternary match is a standard matching kind, Mousikav2 is well supported by diverse switches (matching constraints). Third, unlike previous in-network schemes, the P4 program in Mousikav2 has fewer match-action tables and table entries, requiring less switch memory and stages (resource constraints). Moreover, in Mousikav2, existing sophisticated ML models can be indirectly deployed by transferring their knowledge to the BDT, improving the classification accuracy.

IV. Binary Decision Tree

In this section, we first introduce the BDT training algorithm which is the base of our knowledge distillation. Then, we use an example to demonstrate the training algorithm intuitively.

A. BDT Training

Algorithm 1 BDT Training without Knowledge Distillation

Input: Training set $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, Bit features $A = \{a_1, a_2, ..., a_m\}$.

function BDTGENERATE(D, A)
  Generate node N;
  if $\forall y \in D, \text{Cls}(y) \equiv C$ **then**
    $N.cls \leftarrow C$; // Leaf node
    return;
  end if
  if $A = \emptyset$ **or** $\forall x_1, x_2 \in D, x_1.A \equiv x_2.A$ **then**
    $N.cls \leftarrow \text{CntMaxCls}(\forall y \in D)$; // Leaf node
    return;
  end if
  Select the optimal bit a_* by $\text{Gini}(D, \forall a \in A)$;
  for $v \in \{0, 1\}$ **do**
    Create a branch of node N as $N.brc$;
    $D_v \leftarrow \{(x, y) \mid x.a_* = v,\ x \in D\}$;
    if $\mid D_v \mid \leq min\_samples\_leaf$ **then**
      $N.brc.cls \leftarrow \text{CntMaxCls}(\forall y \in D_v)$; // Leaf
      return;
    else // Inner node
      $N.brc \leftarrow$ BDTGENERATE($D_v, A \setminus \{a_*\}$);
    end if
  end for
end function

Output: The trained binary decision tree.

Based on DT [31], we design Algorithm 1 that grows a BDT on the given networking classification dataset D. Training samples (packets) in D have the form of (x_i, y_i). x_i is a bit string of length $|A|$, i.e., $bin\_feature$ of Section III. $y_i \in \mathbb{R}^K$ denotes the one-hot representation of the class label (K-dimensional vector[2], where the value of the corresponding

[2]Using vectors to represent class labels is convenient for the following knowledge distillation.

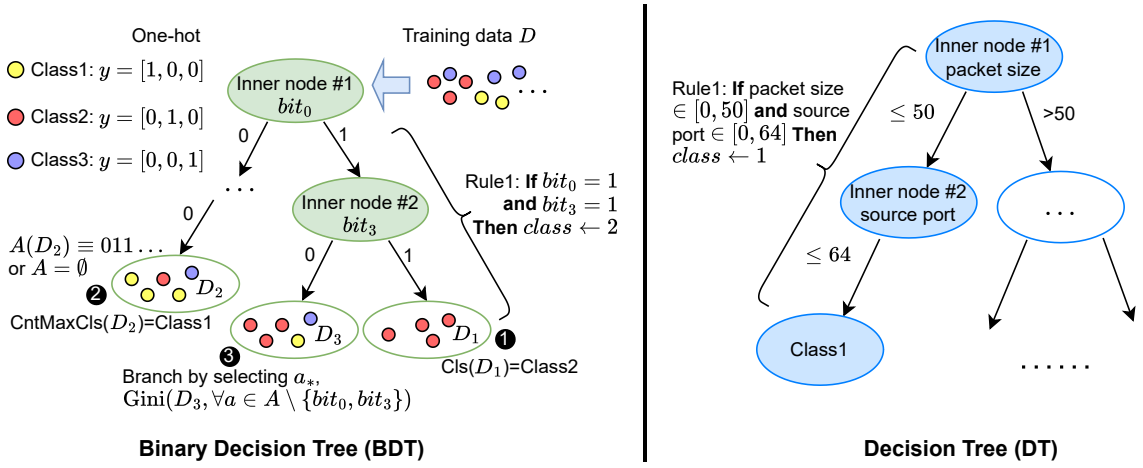


Fig. 3: The binary decision tree (BDT) and the ordinary decision tree (DT).

dimension of the category is 1, and the remaining are 0). The main parts of the BDT training algorithm are:

- Lines 3 $\sim$ 6 indicate that if all samples in D belong to the same category ($\text{Cls}(y) \equiv C$), the BDT generation process stops and the current node N is regarded as a leaf with class label C. Here, function Cls(.) finds the corresponding class label in y:

$$\text{Cls}(y) = \arg\max_j \{y^1, \ldots, y^j, \ldots, y^K\}, \tag{1}$$

  where $j \in [1, K]$ is the index that has the value of 1 in the one-hot vector y.
- Lines 7 $\sim$ 10 reflect another stop processing: if the current bit set A is empty (the bits in A are removed one by one during the generation process, see Line 19) or all samples have the same value (0 or 1) on each bit in A, the current node N is also denoted as a leaf and its class label $N.cls$ is assigned by the returned value of CntMaxCls(.). CntMaxCls(.) is a function to generate a class label from all one-hot vectors $y \in D$:

$$P = (p_1, \ldots, p_j, \ldots, p_K) = \frac{1}{\mid D \mid} \sum_{y \in D} y, \tag{2}$$

$$\text{CntMaxCls}(\forall y \in D) = \arg\max_j P, \tag{3}$$

  where P is the vector maintaining the probability of each class, and the index j of the class with the maximum probability is returned.
- Line 11 selects the optimal branch bit a_* by the Gini index criterion [31]:

$$a_* = \arg\min_{a \in A} \text{Gini}(D, a), \tag{4}$$

$$\text{Gini}(D, a) = \sum_{v \in \{0,1\}} \frac{\mid D_v \mid}{\mid D \mid} (1 - P_{D_v}^\top P_{D_v}), \tag{5}$$

  where $D_v = \{(x, y) \mid x.a = v,\ x \in D\}$ contains all samples whose value of bit a is v, P_{D_v} is calculated by Equation 2 denoting the class probability vector of subset D_v, and the optimal branch bit a_* is the one that has the minimum Gini index. The minimum Gini index reflects the desired effect in the splitting, i.e., the samples from the same category are likely to be assigned in the same node.
- Lines 12 $\sim$ 21 are the loop for tree branching. For each value $v \in \{0, 1\}$ of the optimal a_*, if the corresponding subset D_v contains samples fewer than a predefined threshold (e.g., $min\_samples\_leaf = 5$ in our experiments), the tree generation stops (Lines 15 $\sim$ 17). Otherwise, the tree will grow on the new subset D_v and new bit set $A = A \setminus \{a_*\}$ (Line 19).

B. Training Example

The left side of Fig. 3 depicts how Algorithm 1 grows a BDT. As shown, there are samples of three classes (illustrated as red, purple, and yellow circles, respectively) in the training dataset D, where the label y of samples in a class is represented by a one-hot vector. Then, all training samples are recursively split into nodes until leaf nodes are reached. We summarize three cases (❶ $\sim$ ❸) during this splitting:

- In ❶, the samples in D_1 are from the same class (i.e., the Class2). In light of Lines 3 $\sim$ 6 in Algorithm 1, we set the node of D_1 as a leaf node, stopping the splitting of D_1.
- In ❷, D_2 holds samples of different classes. As there have been several times of sample splitting (node branching) before D_2, the current bit set A may be empty or each sample has the same bit values (i.e., $A(D_2) \equiv 011\ldots$). Thus, we also set the node of D_2 as a leaf node by Lines 7 $\sim$ 10 in Algorithm 1 with Equation 2 and 3. Apparently, CntMaxCls(D_2) returns Class1 as Class1 has the maximum probability.
- In ❸, we continue to split the samples in current inner node by Lines 11 $\sim$ 21 in Algorithm 1. Notably, as inner nodes #1 and #2 have used bit_0 and bit_3, these two used bits are removed in the current splitting.

After the BDT is trained, we can extract several rules of bits. For example, the Rule1 in the left side of Fig. 3 utilizes bit_0 and bit_3 to classify the new samples. As discussed, the bits are from the selected packet features (e.g., ports, packet

size). The right side of Fig. 3 depicts a trained ordinary DT. During the training, the DT also uses the Gini index to choose optimal features and their thresholds for sample splitting [31]. Compared with the DT, the input of the BDT is bits, which makes the Gini index calculation more efficient, and therefore saves training time. E.g., in the right side of Fig. 3, if the packet size has m bits, the number of the Gini index calculation in the DT is 2^m as all thresholds must be compared to select the optimal one (i.e., 50 in Fig. 3). But the BDT only needs to calculate m times to select the optimal bit. Also, according to Line 12 in Algorithm 1, the BDT uses bits of 0 or 1 rather than ranges as branch conditions to split samples. Hence, from the root to each leaf, there is a classification rule consisting of zeros and ones, which is a good fit for the well-supported ternary match in different switches (see Section VI-A for encoding the BDT into ternary match entries).

V. Knowledge Distillation

In this section, we first introduce the preliminaries of knowledge distillation, and then adapt knowledge distillation for our BDT.

A. Preliminaries

Due to the limited computation capacity and memory of mobile devices, deploying sophisticated ML models in these devices encounters great challenges. To this end, the idea of learning a lightweight student model from a sophisticated teacher model is formally popularized as knowledge distillation [36]. In [37], the authors define the class probability vectors of samples predicted by the teacher model as the "knowledge" (i.e., the soft label). Then, the teacher model directly supervises the gradient descent training of the student model on a transfer dataset through the following soft loss function L_{soft}:

$$L_{soft} = -\sum s_i \log(p_i), \tag{6}$$

where for a training sample x_i, the soft label of the teacher is s_i, and the student predicted probability vector is p_i. However, superior teacher models also have a certain error rate by using s_i to predict the class labels of samples. Hence, the ground truth (aka the hard label, h_i), i.e., the one-hot vector in Section IV (representing the true class label) is also considered in the student training:

$$L_{hard} = -\sum h_i \log(p_i). \tag{7}$$

Then, the final gradient descent loss L is the weighted sum of L_{soft} and L_{hard}, that is:

$$L = \alpha L_{soft} + \beta L_{hard}, \tag{8}$$

where weights α and β are in the range of (0, 1).

The great success in practice shows that the student model can mimic the classification behaviors of the teacher model and obtain a comparable or even superior performance [38]–[41]. For example, the authors in [41] present a tree-structure neural network, i.e., the soft decision tree (SDT), as the student model. After the knowledge distillation-based gradient descent, SDT yields a competitive performance on image classification. Furthermore, the interpretability of the SDT can be partly shown by tracing the classification paths in the tree structure.

B. Knowledge Distillation in BDT

The conventional knowledge distillation requires the student model to be parametric and optimizable by the gradient descent. However, our BDT is a rule-based classifier with no parameters to be optimized. Fortunately, some attempts at knowledge distillation without gradient descent seem promising. For instance, the authors in [42] propose the rectified decision tree (ReDT). In ReDT, the soft labels of the teacher model incorporate the impurity calculation to determine the feature selection and node splitting. Following these efforts, we can adapt the conventional knowledge distillation for our rule-based BDT without gradient descent.

For a K-class classification problem, we still use the class probabilities output of the teacher model as the soft label (i.e., the knowledge). The sample x_i in the transfer dataset is first fed to the trained teacher model to output the soft label $s_i \in \mathbb{R}^K$. Besides, each sample has a hard label h_i, i.e., the one-hot vector indicating its true class label (ground truth). Then, we denote

$$\hat{y_i} = \alpha s_i + \beta h_i, \tag{9}$$

where $\hat{y_i}$ is the weighted sum of the soft and hard labels. The pairs of $\{(x_1, \hat{y_1}), \ldots, \{(x_i, \hat{y_i}), \ldots, (x_n, \hat{y_n})\}$ are formed the training set D in Algorithm 1 to train a BDT by knowledge distillation. That is, all we need to do for training the BDT by knowledge distillation is to change the algorithm input (replacing the one-hot vector y_i by the weighted label $\hat{y_i}$) while leaving everything else in the algorithm intact.

Though the alteration is straightforward, the learned knowledge of the teacher model deeply influences the growth of the BDT. Formally, the class probability distribution in a dataset is changed from Equation 2 to

$$\hat{P} = (\hat{p_1}, \ldots, \hat{p_j}, \ldots, \hat{p_K}) = \frac{1}{\mid D \mid} \sum_{\hat{y} \in D} \hat{y}. \tag{10}$$

Accordingly, the calculation of Equation 3 $\sim$ 5 is also changed. In other words, by introducing $\hat{y}$, the knowledge from the teacher can cooperate with the ground truth, teaching the BDT where to stop branching (Lines 7 $\sim$ 10 in Algorithm 1 with changed Equation 2 and 3) and which is the optimal branch bit (Line 11 with changed Equation 4 and 5). Training by knowledge distillation helps the BDT to mimic the classification of the teacher model and obtain a relatively high performance [41], [42]. In addition, Mousikav2 supports a wide variety of models as the teacher, ranging from different NNs to ensemble models, provided that they have good performance and can output class probabilities as the soft labels.

VI. P4 Program

In this section, we discuss the P4 program for in-network classification. First, we introduce how to encode the BDT

rules into the ternary match table entries. Then, we install two P4 tables in a switch, utilizing the encoded entries for classification tasks.

A. BDT Encoding

According to the left side of Fig. 3, the classification rules of a trained BDT consist of zeros and ones. Starting from the first inner node (i.e., the root), a bit feature is checked to be zero or one and the corresponding left or right branch is selected. This branching procedure is repeated until a leaf node is reached, which maintains the predicted class label. In other words, we only need to check the bit value in specific positions according to the traversed inner nodes, which is similar to the ternary match and makes it easy to transfer a BDT rule to a ternary match entry. For example, let $x = bit_0 bit_1 \dots bit_7$ denotes one input sample which has 8 bits. A rule output by the BDT in Fig. 3 is:

$$\textbf{If } bit_0 = 1 \textbf{ and } bit_3 = 1 \textbf{ Then } class \leftarrow \text{Class2}.$$

As bit values in positions 0 and 3 need to be checked, the ternary Mask is 0b10010000 and the corresponding ternary Value is 0b10010000. We just need to examine whether x AND Mask equals the ternary Value to validate the rule matching. If it does match (i.e., x AND Mask = Value), the class label (Class2) will be used as a parameter in the assignment action. Therefore, we fit the sequential decision process of a BDT into the ternary match and action of switch tables. Notably, there is no accuracy decrement after converting the BDT into the ternary match entries (see evaluations in Section VII-E).

B. Switch Tables

As shown in Fig. 2, after encoding the BDT classification rules as ternary match table entries, we develop a P4 program of two switch tables to perform networking classification.

As shown in Listing 1, the first table $tb\_concat\_feature$ is for $bin\_feature$ initialization. $tb\_concat\_feature$ uses default action $ac\_concat\_feature$ to initialize $bin\_feature$ that is stored in the PHV's metadata. As the parsed packet is also stored in the PHV, a packet header can be easily assigned to the corresponding position of $bin\_feature$. As an illustration, Line 3 in Listing 1 assigns the IP protocol field of the parsed IPv4 packet to $meta.bin\_feature$'s low 8 bits.

Listing 1: P4 code fragment that initializes $bin\_feature$

```
1  // assign specific field to bin_feature
2  action ac_concat_feature() {
3      meta.bin_feature[7:0] = hdr.ipv4.protocol;
4      // the assignment of other packet headers is
           omitted ...
5  }
6  // stage 1: feature initialize
7  table tb_concat_feature{
8      actions = {
9          ac_parse_bin_feature;
10     }
11     default_action = ac_parse_bin_feature;
12 }
```

To speed up the operations, P4 switches allow separate tables allocated in the same stage have simultaneity. But we find that tables of $bin\_feature$ initialization and classification can not be performed in parallel at the same stage, as their operations have a dependence on $bin\_feature$. That is, $bin\_feature$ should be first initialized and then used as the key for classification. To resolve such a data dependency, we place the second table of a specific classification task at an additional switch stage (thus a total of 2 stages in our P4 program) and introduce it as follows.

With the table entries encoded from a BDT, the classification process is transformed into the ternary match-action of table tb_packet_cls in Listing 2. According to Line 8 in Listing 2, if there is a match, the corresponding action $ac\_packet\_forward$ (Lines 2 $\sim$ 4 in Listing 2) will be triggered. In $ac\_packet\_forward$, we map classes to forwarding ports of the switch, and packets of different classes will be forwarded to different ports by the switch. For instance, for a coming packet x, if its $bin\_feature$ = 0x0000 0022 0210 0000 0000, it will match (i.e., $x.bin\_feature$ AND Mask = Value) the entry shown in Table I. Then the $port = 1$ of the matched entry is passed to $ac\_packet\_forward$ to label the forwarding port ($ucast\_egress\_port$) of this packet.

Table I: An example of matched entry in tb_packet_cls

Mask	Value	Class (Port)
0x0001 0022 0250 0000 0000	0x0000 0022 0210 0000 0000	1

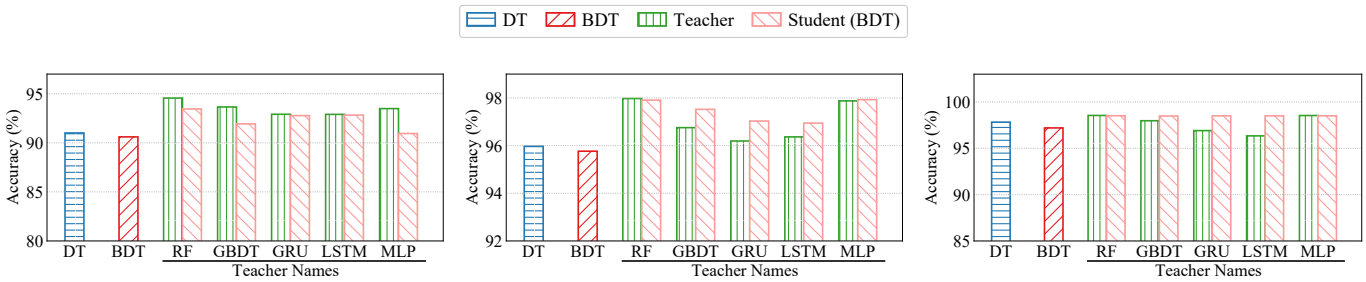
Listing 2: P4 code fragment that implements classification

```
1  // forward packets to different ports
2  action ac_packet_forward(PortId_t port){
3      ig_tm_md.ucast_egress_port = port;
4  }
5  // stage 2: BDT-based classification
6  table tb_packet_cls {
7      key = {
8          meta.bin_feature: ternary;
9      }
10     actions = {
11         ac_packet_forward;
12     }
13     size = 1024;
14 }
```

Actually, we only provide an example of operations for the classified packets in $ac\_packet\_forward$, and other operations can be easily adjusted according to the specific scenario. E.g., the network administrator can change the match entries so that packets of different predicted classes are forwarded to ports with predefined quality-of-service provisions.

VII. Evaluation

In this section, we first introduce the experimental settings. Then, the classification performance of the BDT, DT, and knowledge distillation is evaluated. Finally, we analyze the efficiency of Mousikav2 with different commodity switches and traffic speeds.



(a) Flow Size Prediction (b) Traffic Type Classification (c) Malware Detection

Fig. 4: The classification accuracy of different models on three networking tasks.

A. Experimental Settings

Networking tasks and datasets. To demonstrate the performance of Mousikav2, we build it to classify specific target classes within the context of three tasks:

- We utilize the UNIV1 dataset made available in [43] for the flow size prediction. UNIV1 dataset is collected in one university campus data center. During the classification in this work, we classify the packets that belong to the top 20% flows (w.r.t flow size) in UNIV1 as elephants, while the other packets are mice.
- We utilize the ISCX dataset [44] for the task of classifying traffic according to the application types (i.e., traffic type classification). ISCX dataset contains packets of six application types (Email, Chat, Streaming, File Transfer, VoIP, P2P).
- We leverage the Bot-IoT dataset [45] for the malware detection, identifying whether the packets are from legitimate activities or malicious attacks (e.g., DDoS and service scanning).

Among all the datasets, we split the traffic for training (50%) and testing (50%) according to the different dates and five-tuples, and make sure that there is no overlap between training and testing.

Hardware and tested traffic. Model training, knowledge distillation, and BDT/DT encoding are conducted on a high-performance server with Intel(R) Xeon(R) Gold 6230R CPU @ 2.10GHz and NVIDIA GPU RTX 2080 Ti. Besides, Mousikav2 is deployed in three commodity P4 switches, i.e., EdgeCore Wedge 100BF-65X[3], H3C S9850-32H[4], and OpenMesh BF-48X6Z[5] for traffic testing. The tested traffic of different speeds (10Gbps and 100Gbps) is generated by the traffic generator of KEYSIGHT XGS12-SDL[6] under the Internet mix (IMIX) mode. The IMIX mode generates traffic of hybrid-length packets, which is expected to approximate the real-world traffic [46].

Model settings. Teacher models include two ensemble models (RF [35], GBDT [47]) and three NNs (GRU [34], LSTM [33], MLP [48]). These teacher models are trained by the off-the-shelf ML libraries, i.e., Scikit-learn [49] and Pytorch [50]. The weights of the soft label and hard label are $\alpha = 0.25, \beta = 0.75$. Notably, the current ML libraries do not support training BDT by knowledge distillation. Hence, we implement[7] the BDT and its knowledge distillation from scratch in Python 3.6 with the acceleration of Numpy 1.19.3[8]. We also evaluate the DT by the knowledge distillation scheme (ReDT) in [42]. Learning from [9], we select a set of packet header fields as the model input, including IP protocol, time to live (TTL), and packet size. As suggested by [23], [51], some bias fields (e.g., IP/MAC addresses) and meaningless fields (e.g., checksum) are not considered here.

B. Classification Metrics

Fig. 4 demonstrates the classification accuracy on tasks of flow size prediction, traffic type classification, and malware detection. Among the three tasks, the DT is slightly better than the BDT, because converting the features into the form of bits actually reduces the useful information for classification. However, after the knowledge distillation from superior teacher models, the BDT outperforms the DT. As an illustration, in Fig. 4b, accuracies of the DT and the BDT are respectively 95.97% and 95.76% on the traffic type classification. But the BDT overtakes the DT by 1.93% (97.90% vs. 95.97%) after the knowledge distillation from the RF.

We also consider other three classification metrics in this paper, i.e., the F1-score in Fig. 5, the precision in Fig. 6, and the recall in Fig. 7. Overall, we achieve a similar conclusion. That is, with the sophisticated ML models as the teacher, we can train a better BDT on networking tasks. However, this conclusion does not hold in some special cases. For example, in Fig. 6a, the precision of the BDT after knowledge distillation from the MLP is lower (0.40%↓) than the precision of the original BDT. In Fig. 7a, the recall of the BDT after knowledge distillation from the MLP is the same as the original one (both 90.60%). These special cases are reasonable as just like other ML training techniques, knowledge distillation also has some randomness [36], [37].

[3]https://www.edge-core.com/productsInfo.php?cls=1&cls2=5&cls3=181&id=334

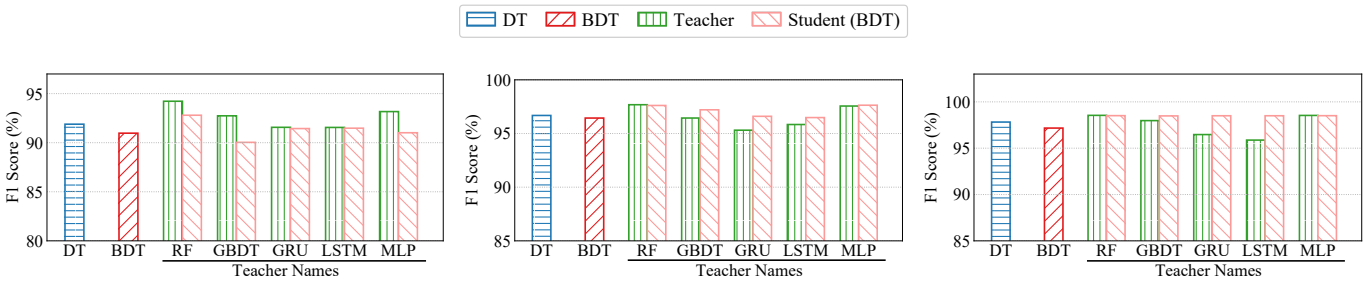
[4]https://www.h3c.com/en/Products_Technology/Enterprise_Products/Switches/Data_Center_Switches/H3C_S9850/

[5]http://www.tooyum.com/products/OpenMesh_BF48X6Z.html

[6]https://www.keysight.com/us/en/products/network-test/network-test-hardware/xgs12-chassis-platform.html?rd=1

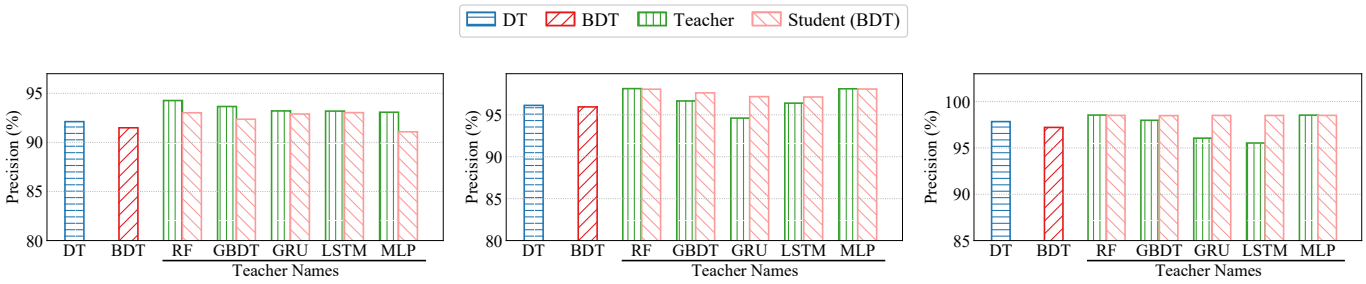
[7]We have optimized our code so that it is much faster than the previous version presented in [17]

[8]https://numpy.org/



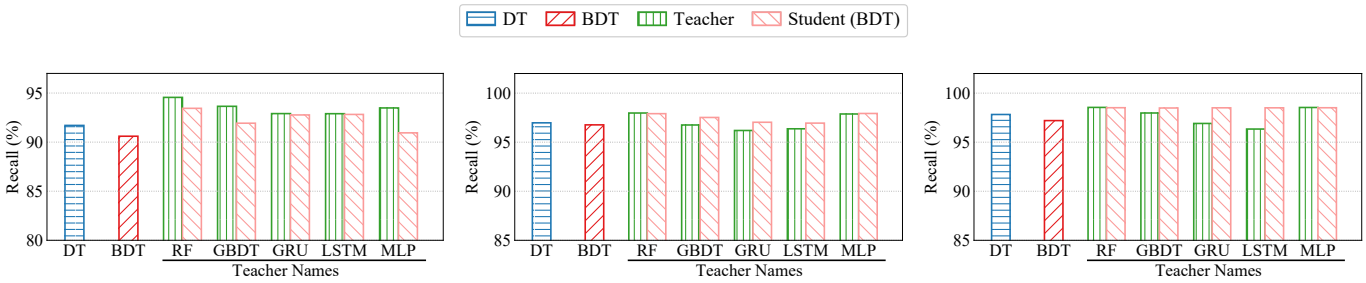
(a) Flow Size Prediction (b) Traffic Type Classification (c) Malware Detection

Fig. 5: The classification F1-score of different models on three networking tasks.



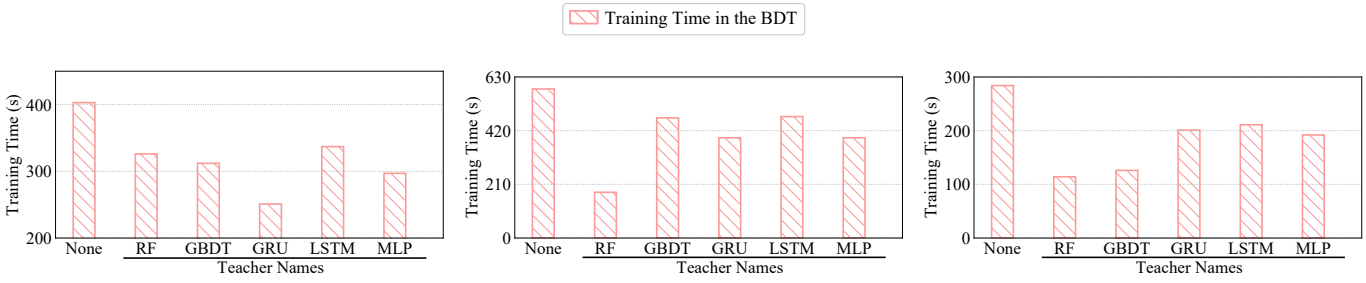
(a) Flow Size Prediction (b) Traffic Type Classification (c) Malware Detection

Fig. 6: The classification precision of different models on three networking tasks.



(a) Flow Size Prediction (b) Traffic Type Classification (c) Malware Detection

Fig. 7: The classification recall of different models on three networking tasks.



(a) Flow Size Prediction (b) Traffic Type Classification (c) Malware Detection

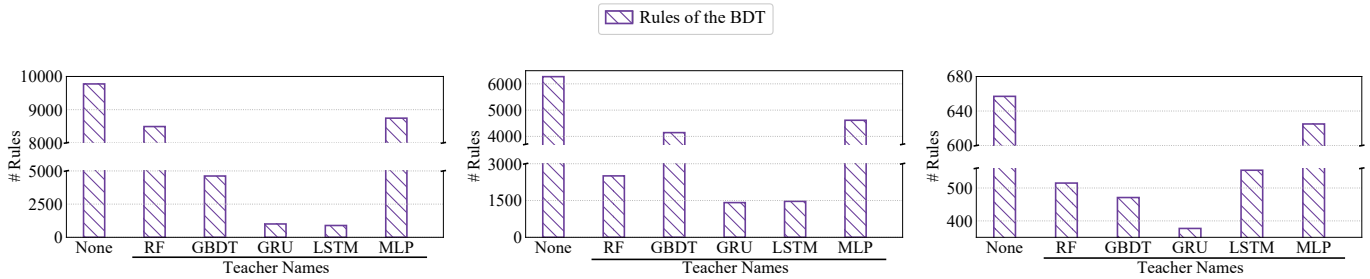
Fig. 8: The training time of the BDT with and without knowledge distillation on three networking tasks.

C. Training Time and Classification Rules

Fig. 8 and 9 depict the training time and the number of classification rules in the BDT. As shown, knowledge distillation not only improves the classification performance, but also reduces the training time and the number of classification rules in the BDT. For instance, in Fig. 8a, the training time of the BDT after the knowledge distillation from the GRU is 251s which is 1.60× faster than that of the original BDT (403s). In Fig. 9b, the knowledge distillation from the LSTM helps to reduce the number of classification rules by 4.20× (i.e., from 6273 to 1466). One significant reason is that knowledge distillation is useful to transfer the learned classification experience from existing teacher models to the BDT, helping the BDT to find the optimal node branching in time, and thus reducing the training time and unnecessary node branching.

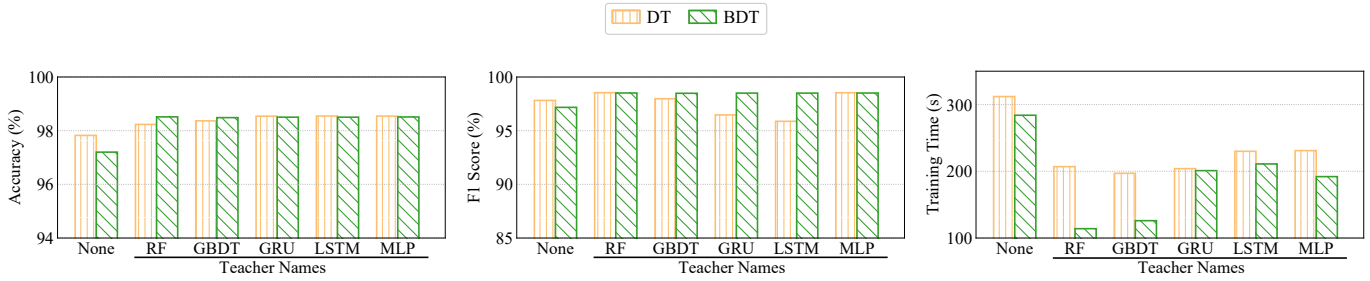
D. Compare with the DT after Knowledge Distillation

To further analyze the performance of the BDT after knowledge distillation, we also compare the BDT with the DT after knowledge distillation on the task of malware detection. Fig. 10 shows the classification accuracy, F1-score, and training time of both tree models with and without knowledge distillation. As demonstrated, the BDT has a better performance after knowledge distillation. Especially in Fig. 10b), the F1-scores of the DT even decrease after the knowledge distillation from the GRU (from 97.82% to 96.47%) and the LSTM (from 97.82% to 95.87%). In Fig. 10c, the training time of the BDT has a more significant reduction after the knowledge distillation from the RF than the DT. In detail, the training time of the BDT is from 284s to 114s (2.49× ↓). But from the same RF teacher model, the training time of the DT is from 312s to 207s (only 1.50× ↓).



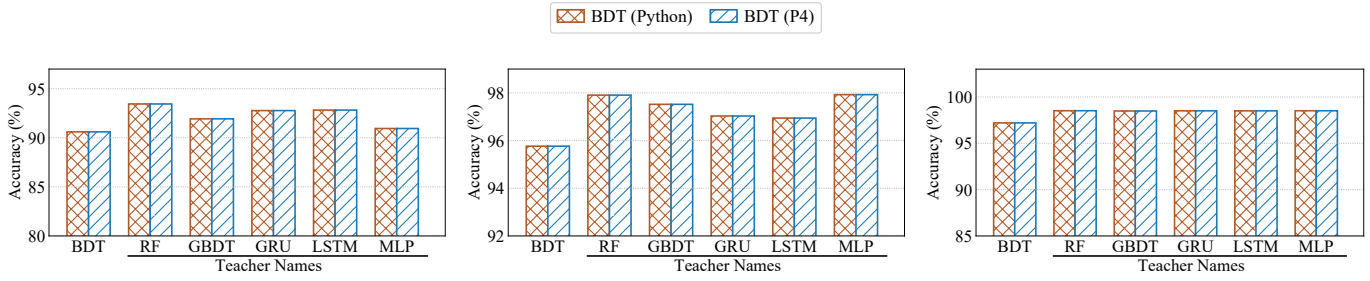
(a) Flow Size Prediction (b) Traffic Type Classification (c) Malware Detection

Fig. 9: The number of classification rules in the BDT with and without knowledge distillation from different teacher models on the three networking tasks.



(a) Accuracy (b) F1-score (c) Training Time

Fig. 10: The performance of DT and BDT on the task of malware detection. Both tree models are trained with and without knowledge distillation.



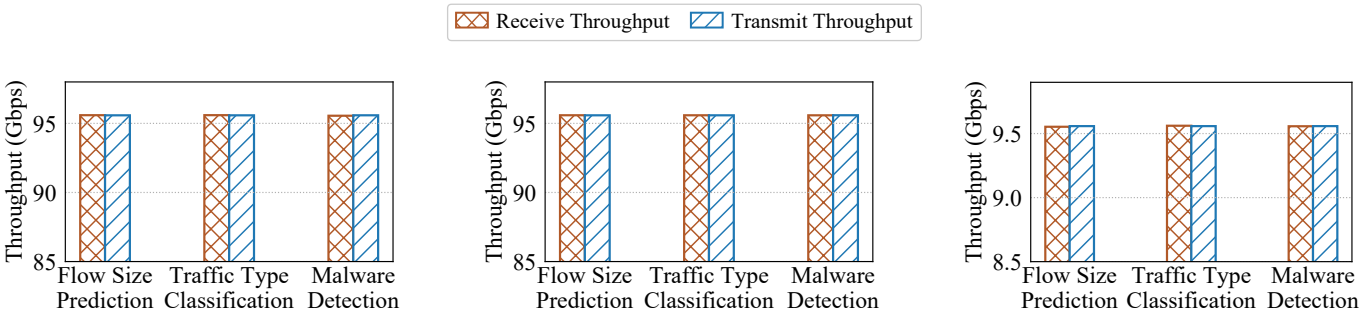
(a) Flow Size Prediction (b) Traffic Type Classification (c) Malware Detection

Fig. 11: The classification accuracy of different BDT versions (Python and P4) on three networking tasks. The P4 version is evaluated on the H3C switch.

E. Mousikav2 in Switches

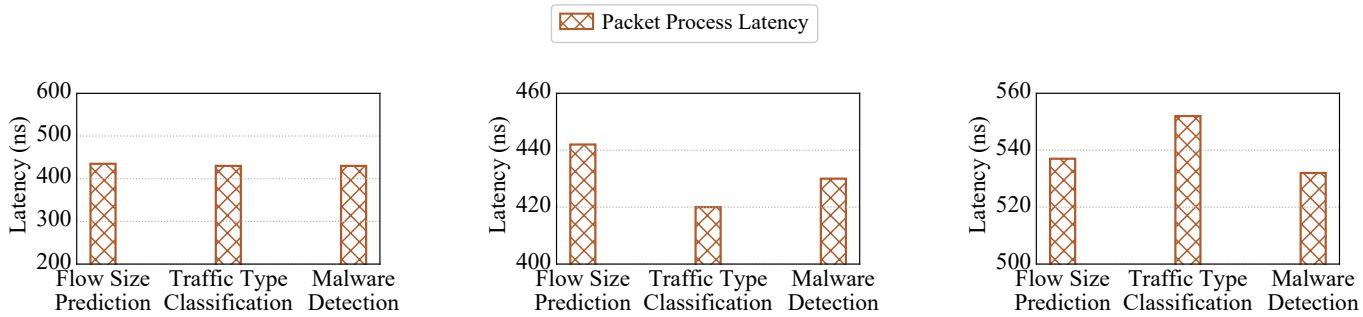
The previous experiments are performed with Python. We now further evaluate our scheme on three commodity P4 switches. As mentioned, the switches are from brands of EdgeCore, H3C, and OpenMesh.

We first consider the correctness of converting the BDT into ternary match entries. Fig. 11 illustrates the classification accuracies of the BDT based on Python and P4 (in the switch of H3C). As depicted, on all three tasks, after encoding the BDT into the P4 program, there is no accuracy difference. Then, for each evaluated networking task, we deploy the BDT with the best accuracy on the three commodity switches. Fig. 12 shows the switch throughput under the traffic speeds of 10Gbps and 100Gbps (generated by the KEYSIGHT generator). As shown, after loading the P4 program and installing encoded entries from the BDT, the switch throughput changes little. Moreover, Fig. 13 shows the packet processing latency of different switches. The latency results show that after deploying the BDT for in-network classification tasks, the latency of switches is still small (< 550 nanoseconds). That is, the packet processing in switches is still at the line rate.



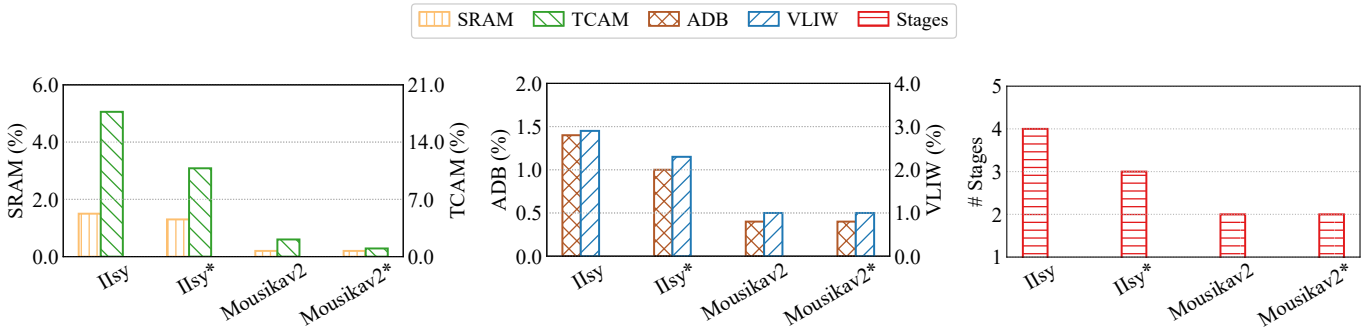
(a) EdgeCore (b) H3C (c) OpenMesh

Fig. 12: The throughput of switches (EdgeCore and H3C are 100Gbps, OpenMesh is 10Gbps). For each networking task, we select the BDT of best classification accuracy according to Fig. 4. Due to the simulation features (e.g., inter-frame gap and mixed packet sizes), it is impossible to generate traffic of exactly 10Gbps or 100Gbps by the KEYSIGHT generator.



(a) EdgeCore (b) H3C (c) OpenMesh

Fig. 13: The latency of different switches after deploying the BDT. For each networking task, we select the BDT of best classification accuracy according to Fig. 4.



(a) Memory Resources (b) Action Resources (c) Number of Stages

Fig. 14: The H3C resource consumption of in-network schemes (IIsy and Mousikav2) on the malware detection. IIsy* deploys the DT after knowledge distillation, and Mousikav2* deploys the BDT after knowledge distillation.

F. Compare with IIsy

To further demonstrate the superiority of Mousikav2, we now compare the switch resource consumption of Mousikav2 and IIsy (a state-of-the-art framework that installs the DT in switches for classification tasks, see Section II-C). For the resource consumption, we mainly focus on the following three aspects: 1) Memory resources, i.e., the percentage of the used TCAM and SRAM; 2) Action resources, i.e., the percentage of the used action data bus (ADB) and very long instruction word (VLIW); 3) The number of the used switch stages.

Fig. 14 depicts the consumption of the mentioned resources on the task of malware detection. Notably, we consider not only the ordinary DT and BDT (i.e., IIsy and Mousikav2), but also the DT and BDT after knowledge distillation (i.e., IIsy* and Mousikav2*). As shown, both the DT and BDT after knowledge distillation reduce the resource consumption and Mousikav2* has the lowest consumption. For instance, in Fig. 14a, the TCAM usage of IIsy and Mousikav2* is respectively 17.70% and 1.00% (17.70$\times$ $\downarrow$). In Fig. 14c, Mousikav2* uses only 2 stages. While IIsy leverages 4 stages, which is 2.00$\times$ more than Mousikav2*.

VIII. Conclusion

In this paper, we propose Mousikav2, a novel in-network classification framework. Mousikav2 aims to tackle the drawbacks of offloading the learning models to the switch. First, we redesign the DT algorithm, getting the binary decision tree (BDT). The classification rules of the BDT are of bits, which fits the widely used ternary match well. Second, we design a P4 program, encoding the rules of the BDT for classification tasks. This program only uses a small amount of the switch resources and is lightweight enough for current compact switches. Moreover, given the complexity of other ML models, in Mousikav2, we adopt a teacher-student knowledge distillation architecture to transfer the knowledge from other models to the BDT. By doing so, we can not only utilize their knowledge for better performance, but also avoid their complex deployments in switches.

We conduct comprehensive experiments to demonstrate the performance of Mousikav2. On the task of traffic type classification, the BDT after knowledge distillation improves the accuracy of the DT by 1.93%. The experiments at the traffic speeds of 10Gbps and 100Gbps show that after loading Mousikav2, the traffic can still be transmitted at a line rate with packet processing latency below 550 nanoseconds. Meanwhile, compared with IIsy in [9], Mousikav2 reduces the memory (TCAM) consumption by 17.70$\times$.

References

[1] M. Wang, Y. Cui, G. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," *IEEE Network*, vol. 32, no. 2, pp. 92–99, 2017.

[2] J. Li and Z. Pan, "Network traffic classification based on deep learning," *Transactions on Internet and Information Systems*, vol. 14, no. 11, pp. 4246–4267, 2020.

[3] P. Poupart, Z. Chen, P. Jaini, F. Fung, H. Susanto, Y. Geng, L. Chen, K. Chen, and H. Jin, "Online flow size prediction for improved network routing," in *Proceedings of the 24th IEEE International Conference on Network Protocols*, 2016, pp. 1–6.

[4] 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.

[5] 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.

[6] Z. Zhao, X. Shi, Z. Wang, Q. Li, H. Zhang, and X. Yin, "Efficient and accurate flow record collection with hashflow," *Transactions on Parallel Distributed System*, vol. 33, no. 5, pp. 1069–1083, 2022.

[7] 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.

[8] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. E. P. Reyes, M. Shyu, S. Chen, and S. S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Computing Surveys*, vol. 51, no. 5, pp. 92:1–92:36, 2019.

[9] Z. Xiong and N. Zilberman, "Do switches dream of machine learning?: Toward in-network classification," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, 2019, pp. 25–33.

[10] J.-H. Lee and K. Singh, "Switchtree: in-network computing and traffic analyses with random forests," *Neural Computing and Applications*, pp. 1–12, 2020.

[11] C. Zheng and N. Zilberman, "Planter: seeding trees within switches," in *Proceedings of the 2021 ACM SIGCOMM Conference*, 2021, pp. 12–14.

[12] P4 Language Consortium, "v1model.p4," Website, https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4, accessed: 2022-09-13.

[13] Barefoot Networks, "Tofino switch," Website, https://www.barefootnetworks.com/products/brief-tofino/, accessed: 2022-09-13.

[14] A. Aleesa, M. Younis, A. A. Mohammed, and N. Sahar, "Deep-intrusion detection system with enhanced unsw-nb15 dataset based on deep learning techniques," *Journal of Engineering Science and Technology*, vol. 16, pp. 711–727, 2021.

[15] D. Kim, J. Nelson, D. R. K. Ports, V. Sekar, and S. Seshan, "Redplane: enabling fault-tolerant stateful in-switch applications," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2021, pp. 223–244.

[16] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 15–28.

[17] G. Xie, Q. Li, Y. Dong, G. Duan, Y. Jiang, and J. Duan, "Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation," in *Proceedings of the International Conference on Computer Communications*, 2022, pp. 1938–1947.

[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 26th Annual Conference on Neural Information Processing Systems*, 2012, pp. 1106–1114.

[20] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019, pp. 4171–4186.

[21] H. Peng, J. Li, Y. He, Y. Liu, M. Bao, L. Wang, Y. Song, and Q. Yang, "Large-scale hierarchical text classification with recursively regularized deep graph-cnn," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, 2018, pp. 1063–1072.

[22] 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.

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

[24] T. Pan, N. Yu, C. Jia, J. Pi, L. Xu, Y. Qiao, Z. Li, K. Liu, J. Lu, J. Lu, E. Song, J. Zhang, T. Huang, and S. Zhu, "Sailfish: accelerating cloud-scale multi-tenant multi-service gateways with programmable switches," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2021, pp. 194–206.

[25] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. M. Caulfield, E. S. Chung, H. K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair,

D. Bansal, D. Burger, K. Vaid, D. A. Maltz, and A. G. Greenberg, "Azure accelerated networking: Smartnics in the public cloud," in *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation*, 2018, pp. 51–66.

[26] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. V. Ramos, and A. Madeira, "Flowlens: Enabling efficient flow classification for ml-based network security applications," in *Proceedings of the 28th Annual Network and Distributed System Security Symposium*, 2021.

[27] P4 Language Consortium, "core.p4," Website, https://github.com/p4lang/p4c/blob/main/p4include/core.p4, accessed: 2022-09-13.

[28] H. Liu, "Efficient mapping of range classifier into ternary-cam," in *Proceedings of the 10th Annual IEEE Symposium on High Performance Interconnects*, 2002, pp. 95–100.

[29] G. J. Narlikar, A. Basu, and F. Zane, "Coolcams: Power-efficient tcams for forwarding engines," in *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, 2003, pp. 42–52.

[30] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *Proceedings of the Conference of the 2017 ACM Special Interest Group on Data Communication*, 2017, pp. 15–28.

[31] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth, 1984.

[32] C. Busse-Grawitz, R. Meier, A. Dietmüller, T. Bühler, and L. Vanbever, "pforest: In-network inference with random forests," *arXiv preprint arXiv:1909.05680*, 2019.

[33] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[34] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[35] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[36] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[37] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[38] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2014, pp. 2654–2662.

[39] G. Urban, K. J. Geras, S. E. Kahou, Ö. Aslan, S. Wang, A. Mohamed, M. Philipose, M. Richardson, and R. Caruana, "Do deep convolutional nets really need to be deep and convolutional?" in *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[40] J. Bai, Y. Li, J. Li, X. Yang, Y. Jiang, and S.-T. Xia, "Multinomial random forest," *Pattern Recognition*, vol. 122, p. 108331, 2022.

[41] N. Frosst and G. E. Hinton, "Distilling a neural network into a soft decision tree," in *Proceedings of the 1st International Workshop on Comprehensibility and Explanation in AI and ML 2017 co-located with 16th International Conference of the Italian Association for Artificial Intelligence*, vol. 2071, 2017.

[42] J. Bai, Y. Li, J. Li, Y. Jiang, and S. Xia, "Rectified decision trees: Towards interpretability, compression and empirical soundness," *arXiv preprint arXiv:1903.05965*, 2019.

[43] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the Tenth conference on Internet measurement*, 2010, pp. 267–280.

[44] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related features," in *Proceedings of the 2nd International Conference on Information Systems Security and Privacy*, 2016, pp. 407–414.

[45] 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.

[46] R. Bolla, R. Bruschi, C. Lombardo, and D. Suino, "Evaluating the energy-awareness of future internet devices," in *Proceedings of the 12th IEEE International Conference on High Performance Switching and Routing*, 2011, pp. 36–43.

[47] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[48] 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.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vander-Plas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[50] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2019, pp. 8024–8035.

[51] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè, "Toward effective mobile encrypted traffic classification through deep learning," *Neurocomputing*, vol. 409, pp. 306–315, 2020.

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.

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.

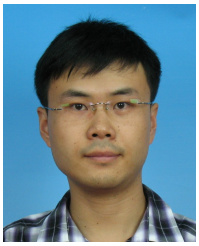
Guanglin Duan received his B.S. degree from Tsinghua University, Beijing, China in 2020. He is pursuing an M.S. degree in Tsinghua Shenzhen International Graduate School. His research interests include programmable data planes, software-defined programmable network security, etc.

Jiaye Lin is currently pursuing the M.E. degree in the International Graduate School, Tsinghua University, Shenzhen, China. He received his B.E. degree (2022) in the School of Intelligent Systems Engineering, Sun Yat-sen University, Guangzhou, China. His research interests include in-network intelligence, P4 program development, and reinforcement learning.

Yutao Dong received the B.S. degree (2020) from Jilin University, Changchun, China. Currently, he is an M.S. student in Tsinghua Shenzhen International Graduate School. His research interests include anomaly detection in the network environment and AI for networks, etc.

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.

Dan Zhao is currently an assistant researcher with Peng Cheng Laboratory, Shenzhen, China. She was a Postdoctoral Researcher with School of Electronic Engineering, Dublin City University and with School of Computing, National College of Ireland. Dr Dan Zhao obtained her bachelor's degree from Beijing University of Posts and Telecommunications in 2011, majored in Telecommunications. In 2015, she graduated from Queen Mary University of London with a Ph.D. degree in Electronic Engineering.



Yuan Yang received the B.Sc., M.Sc., and Ph.D. degrees from Tsinghua University. He was a Visiting Ph.D. Student with The Hong Kong Polytechnic University. He is currently an Assistant Researcher with the Department of Computer Science and Technology, Tsinghua University. His major research interests include computer network architecture and routing protocols.