Mousika: Enable General In-Network Intelligence in Programmable Switches by Knowledge Distillation

Guorui Xie*†, Qing Li†, Yutao Dong*†, Guanglin Duan*†, Yong Jiang*†, Jingpu Duan‡†
* International Graduate School, Tsinghua University, Shenzhen, China
† Peng Cheng Laboratory, Shenzhen, China
‡ Southern University of Science and Technology, Shenzhen, China

IEEE INFOCOM 2022 - IEEE Conference on Computer Communications | 978-1-6654-5822-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/INFOCOM48880.2022.9796936

***Abstract*—Given the power efficiency and Tbps throughput of packet processing, several works are proposed to offload the decision tree (DT) to programmable switches, i.e., in-network intelligence. Though the DT is suitable for the switches' match-action paradigm, it has several limitations. E.g., its range match rules may not be supported well due to the hardware diversity; and its implementation also consumes lots of switch resources (e.g., stages and memory). Moreover, as learning algorithms (particularly deep learning) have shown their superior performance, some more complicated learning models are emerging for networking. However, their high computational complexity and large storage requirement are cause challenges in the deployment on switches. Therefore, we propose Mousika, an in-network intelligence framework that addresses these drawbacks successfully. First, we modify the DT to the Binary Decision Tree (BDT). Compared with the DT, our BDT supports faster training, generates fewer rules, and satisfies switch constraints better. Second, we introduce the teacher-student knowledge distillation in Mousika, which enables the general translation from other learning models to the BDT. Through the translation, we can not only utilize the super learning capabilities of complicated models, but also avoid the computation/memory constraints when deploying them on switches directly for line-speed processing.**

***Index Terms*—In-network Intelligence; Decision Tree; Programmable Switch.**

I. Introduction

Recent years have witnessed an emerging trend of applying learning algorithms to many networking scenarios [1]. For instance, to improve the routing performance, in [2], the authors present a scheme of using machine learning models (e.g., Gaussian Process Regression and Neural Network) for flow size prediction. In [3]–[5], different deep learning models (e.g., Convolutional Neural Network, CNN and Recurrent Neural Network, RNN) are proposed for traffic classification to improve the network management and QoS provisioning. In [6], the authors also propose a neural network to detect malware traffic and guarantee network security. The current solution of implementing the learning model as a virtual network function (VNF) on X86 servers cannot provide satisfactory processing latency and capacity. However, deploying these learning models on the network devices directly for high-speed processing is a significant challenge to be addressed.

Compared with the traditional network devices, the modern programmable switches (e.g., P4 switches [7] of Tbps) enable the programmable logic. Therefore, some works are proposed to completely deploy the learning algorithms on switches' data plane for line-speed processing, i.e., in-network intelligence [8], [9]. In order to guarantee the high-speed processing, only simple instructions like integer additions and bit shifts are allowed. This limitation hinders the deployment of complicated models except for the decision tree (DT), a rule-based learning classifier [10]. In [11], the authors propose IIsy, which utilizes several programmable stages and lots of range match rules to offload a DT on the switch. Although the DT seems to fit the switches' match-action paradigm, it still has three challenges: 1) The range match used in the DT is not widely supported. 2) The implementation in [11] consumes multiple stages and contains thousands of rules, which may delay the packet processing and take up lots of the hardware resources. 3) The learning capability of the DT is not as powerful as other models, e.g., RNN and CNN.

Based on these, in this paper, we propose Mousika [1], a well-designed in-network intelligence framework for programmable switches that can successfully address the aforementioned drawbacks:

- The device compatibility. There are diverse software/hardware programmable switches that support different standards. Thus, the range match (e.g., defined in v1model.p4 [12]) used in the DT may not be widely supported by other switches [13]. In Mousika, we modify the DT to a new rule-based model, the *Binary Decision Tree*, whose classification rules are bits and thus can be effectively encoded into the widely-supported ternary match (defined in the standard core.p4 library).
- The model translation. Except for the DT, other learning-based models can hardly be deployed on the switches due to their complexity of computation and storage [14]. But their designs maintain lots of domain knowledge and usually have superior performance. Thus, in Mousika, we adopt a teacher-student knowledge distillation architecture to teach the BDT [15]. In other words, the knowledge of teachers (e.g., deep learning or ensemble machine learning models) is translated into the BDT through this architecture. As a result, the teacher models are deployed on switches indirectly, which avoids the computation and

Corresponding author: Qing Li (liq@pcl.ac.cn)

[1]Mousika is a Greek deity of learning.

memory constraints of switches. Furthermore, knowledge distillation can help to reduce the BDT training time, reduce the number of rules, and sometimes get a more accurate BDT.

- The resource limitations. Usually, programmable switches are compact and have critical limitations on computation and memory [11]. To share these resources with other network management tasks (e.g., access control list and routing), the program that implements the offloaded model should be lightweight. To tackle this, we delicately design a 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 for the cooperation with other tasks.

To sum up, in Mousika, we first leverage a teacher-student architecture to help the knowledge transfer from other models to a BDT; then we convert the BDT classification rules into ternary match table entries; these entries are finally installed on a P4 program for packet processing on the switch. Ideally, Mousika is compatible with diverse switches, and its implementation only takes up a small amount of hardware resources. Furthermore, Mousika is a general framework that supports the translation of multiple complex but powerful models. Through this translation, we can not only utilize the super learning capabilities of complicated models, but also avoid the computation/memory constraints when deploying them on switches.

We conduct thorough experiments on three scenarios (flow size prediction, traffic type classification, and malware detection) to evaluate the performance of Mousika, according to the experimental results:

- The BDT after knowledge distillation usually has a better performance. For the flow size prediction task, the distilled BDT improves the accuracy of the DT by $\sim 3\%$ (94.95% vs. 92.23%), reduces the training time by ~ 156 times than the DT (276.6s vs. 43170.6s), and reduces the number of rules by ~ 4.9 times than the ordinary BDT (1216 vs. 5948).
- Due to the efficient processing performance of the hardware switch, loading Mousika has little impact on the packet forwarding. For traffic speeds of 40Gbps and 100Gbps, the traffic can still be transmitted at line rate with zero packet loss.
- Compared with IIsy proposed in [11], Mousika only occupies a small amount of resources on the hardware switch. For the traffic type classification task, Mousika takes up 2 tables, 2 stages, and 1.7% of the TCAM, but IIsy occupies 11 tables, 10 stages, and 63.5% of TCAM. In addition, for high-speed traffic (100Gbps), the packet processing of IIsy is slower than Mousika by 9 nanoseconds averagely.

II. Background and Motivation

A. P4 Switch and Its Constraints

As illustrated in Fig. 1, the data plane on the P4 switch has a Protocol Independent Switch Architecture (PISA). For an

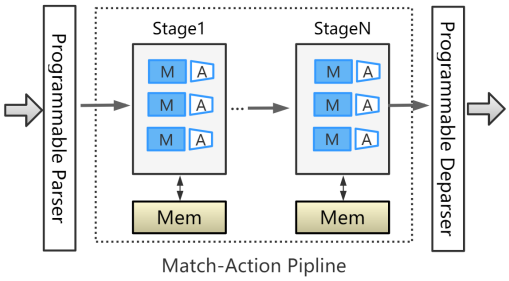


Fig. 1: The Protocol Independent Switch Architecture (PISA). PISA consists of a parser, a deparser, and a series of stages. A stage contains several match-action units to perform match (M) and action (A).

arriving packet, it is first mapped into a Packet Header Vector (PHV) by the parser. Then the PHV is passed to a pipeline. The pipeline consists of *match-action units* (MAUs) arranged in several stages. If a header field (e.g., the destination port) in the PHV matches (M) a given table, further processes in the action unit (A) associated with the matching table's entry are triggered. Finally, the processed PHV is reorganized into a packet by the deparser. In summary, PISA allows administrators to define customized processes (e.g., tables and actions) in P4 language and then instantiate them inside MAUs.

Matching constraints. For P4 language, the standard library (core.p4 [16]) defines three standard match kinds: 1) Exact match. The key has to match exactly with the field in the rule. 2) Ternary match. The key is AND with the mask associated with each rule, and then compared with the value for exact match. 3) Longest prefix match (LPM). Compared with ternary match, this case guarantees that the mask is a series of consecutive bits 1 followed by a series of consecutive bits 0 [13]. These are the standard match kinds that are supported widely by diverse devices. Other libraries (e.g., v1model.p4 [12]) may define additional match kinds such as range match, fuzzy match. But those are not available on many hardware targets [11].

Memory 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 [17]. 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 it [18]. Generally, this compact memory is shared by multiple advanced network management tasks such as access control list and routing. Hence, this constraint reminds us to design P4 programs with as few table entries as possible.

Processing constraints. To guarantee the high-speed processing, complex instructions such as multiplications, divisions, and floating-point operations (e.g., polynomials or loga-

rithms) are not allowed. Packet processing in MAUs is limited to very simple instructions like integer additions and bit shifts. For these allowed instructions, their number in a specific action is also constrained. In addition, the entire pipeline should occupy as few stages as possible, otherwise, the packet will be delayed (or even stalled) when it passes through the switch.

B. Learning on Programmable Switches

The main problem for in-network intelligence on programmable switches is the implementation complexity of the mathematical operations. Naturally, packet processing on the programmable switch is based on a series of predefined flow tables, i.e., the match-action paradigm, which may hinder multiple learning-based schemes [2]–[6]. To tackle this, the authors in [11] propose a framework, IIsy, which is used to map a rule-based classifier, the decision tree (DT), into table entries in the switch table. In IIsy, the number of stages implemented in the switch is equal to the number of features used as the DT input plus one. At each stage, it matches a feature with all its potential values. The matching result is encoded into the metadata field and indicates the branch taken in the tree. The last stage gets all the coded fields from the metadata and maps (matches) the values to the result leaf nodes. Except for IIsy, there also exist several works that deploy the DT on switches. In [8], the authors give a try to transfer deep learning models to a DT and then load the DT on the switch. In brief, they first reproduce a dataset that contains instances labeled by the deep models. Next, this dataset is resampled according to an advantage function [19]. Finally, the DT is trained on the reproduced dataset and deployed on the switch. But due to the complicated procedures, the authors leave the deployment on hardware switches to the future. In [9], the authors propose a two-phase learning scheme for flow size prediction. The model in the first phase is a DT that can be encoded to a set of rules executable by switches for identifying potential elephant flows. Then, in the second phase, the controller uses a more sophisticated stream mining model to detect true elephant flows from those candidates.

Although the DT seems to be a good fit for the match-action paradigm used by programmable switches, there remain some limitations: 1) The DT is implemented by range match which is not widely supported, given the diversity of switch hardware. Though range match can be translated into ternary match, the translated table entries will increase multiple times of resource consumption [13]. 2) Implementing the DT on switches takes up multiple stages which may delay the packet forwarding. 3) The unoptimized DT may generate thousands of rules, and these rules will take up a lot of memory on the compact switch. Thus, in Mousika, we redesign the DT algorithm, getting a new rule-based model, the *Binary Decision Tree* (BDT), which satisfies all mentioned limitations of switches.

C. Learning on Networking

Except for the DT, other learning algorithms (especially deep learning) have been employed in every possible field to leverage their amazing power, e.g., computer vision and natural language process [20]–[23]. Networking has not escaped this trend, and several schemes are proposed to use learning algorithms for optimization and decision making [1]. In [2], the authors concern 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 [3], the authors focus on the problem of Internet traffic classification. They designed a deep learning-based system that can handle both the traffic characterization task and the application identification task. The system uses deep learning models such as Convolutional Neural Network (CNN) and Stacked Auto-Encoder (SAE). The works in [4], [5] also focus on employing the power of deep learning for traffic classification, the employed deep models are Recurrent Neural Network (RNN) and One-dimensional CNN respectively. In [6], the authors focus on malware detection by using an ensemble of neural networks called Auto-Encoders to collectively differentiate between normal and abnormal traffic patterns.

Now many learning-based works reach better accuracy and defeat their predecessors. However, as the switches only support match-action paradigm and some simple instructions, deploying them directly for in-network applications may be infeasible [14], [24]. Thus, in Mousika, we leverage the knowledge distillation architecture to transfer the knowledge from a complicated model to the BDT, and then deploy the BDT to the switch, avoiding the computational and memory constraints of directly deploying complicated models.

D. Knowledge Distillation

As known, due to the limited computation capacity and memory of the mobile devices, deploying deep models in these devices encounters great challenges. To address this problem, the idea of learning a small student model from a large teacher model is formally popularized as knowledge distillation [14]. In [15], the authors define the class probabilities output by the teacher model as the "knowledge" (i.e., soft label). Then the teacher model directly guides the training of the student model on a transfer dataset through the objective function:

$$L = -\sum y_i \log(p_i) \tag{1}$$

where for a training sample i, the soft label of the teacher is y_i, the student prediction output is p_i. Based on this, in [25], the authors present a tree-structure neural network, i.e., the Soft Decision Tree (SDT), as the student model. After the guided training, one can track the classification path of the SDT to understand its decision making. In [26], the authors propose the Rectified Decision Tree (ReDT), a knowledge distillation based decision tree rectification. In ReDT, the teacher knowledge participates the impurity calculation to determine the feature selection and node splitting. The great success in practice shows that the student model can mimic the teacher model and obtain a competitive or even a superior performance [27]–[29].

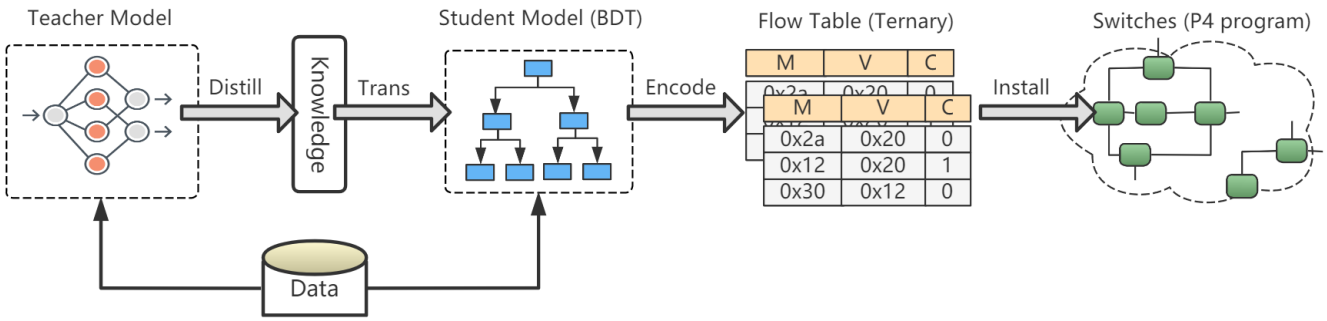


Fig. 2: The Mousika Framework. Mousika utilizes the teacher-student architecture to transfer the knowledge from other models to BDT. Next, the BDT rules are encoded into ternary-match table entries, according to Section III-C, a ternary match entry contains the mask (M), value (V), and class (C). Finally, these entries are installed on the P4 program for packet processing.

III. Methodology

A. Framework Overview

Our whole in-network intelligence framework, Mousika, is shown in Fig. 2. For a packet classification/prediction task, one could give a trained teacher model and a transfer dataset. Next, we utilize the teacher-student knowledge distillation architecture to translate the teacher model into the BDT. After the translation, we obtain a trained BDT with teacher knowledge embedded. Then, we encode the classification rules of the BDT into ternary match table entries, and finally install these entries on the P4 program of the switch. Now, for an arriving packet on the switch, the whole classification task is model-free, and the P4 program will look up the packet's header fields and find its matched table entry and the related class label.

Compared with the existing in-network schemes [8], [9], [11], Mousika can perfectly fit all P4 switch constraints. First, the rules of the BDT can be directly encoded into ternary match table entries. Ternary match is defined in the standard core.p4 library and supported by diverse switches (matching constraints). Second, compared with the DT, the BDT generates fewer rules, thus there are also fewer entries after the conversion (memory constraints). Third, in Mousika, we develop a P4 program that only requires two switch stages and uses the (ternary) match-action operation for packet classification/prediction (processing constraints). Moreover, in Mousika, existing deep learning or ensemble machine learning models can be the teacher of the BDT for knowledge distillation. The distilled BDT requires less training time, fewer rules, and sometimes may have a better accuracy. In the next Section III-B and III-C, we will introduce the algorithm and implementation of Mousika in detail.

B. BDT and Its Knowledge Distillation

BDT. Based on DT algorithm, we design Algorithm 1 that demonstrates the details of the BDT growth on the packet dataset D. For a packet $(x_i, y_i) \in D$, x_i is the selected part of the PHV in the form of bits (an instance of A), and y_i denotes the one-hot representation of the packet class (K-dimensional vector, where the value of the corresponding dimension of the category is 1, and the remaining are 0). The main codes of the BDT training are:

Algorithm 1 Binary Decision Tree Training

Input: Training set $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, Bit set $A = \{a_1, a_2, ..., a_m\}$ is the selected part of the packet header vector in the switch parser.

1: **function** BDTGENERATE(D, A)
2: Generate node N;
3: **if** $\forall y \in D, \text{Cls}(y) \equiv C$ **then**
4: $N.cls \leftarrow C$; **return**
5: **end if**
6: **if** $A = \emptyset$ **OR** $\forall x_1, x_2 \in D, A_{x1} \equiv A_{x2}$ **then**
7: $N.cls \leftarrow \text{CntMaxCls}(\forall y \in D)$; **return**
8: **end if**
9: Select the optimal bit a_* from A;
10: **for** $v \in \{0, 1\}$ **do**
11: Create a branch of node N as $N.brc$;
12: $D_v \leftarrow \{(x, y) \mid a_* = v,\ x \in D\}$;
13: **if** $\mid D_v \mid \leq min\_samples\_leaf$ **then**
14: $N.brc.cls \leftarrow \text{CntMaxCls}(\forall y \in D)$; **return**
15: **else**
16: $N.brc \leftarrow$ BDTGENERATE($D_v, A \setminus \{a_*\}$);
17: **end if**
18: **end for**
19: **end function**

Output: The binary decision tree N.

Lines 3 $\sim$ 5 indicate that if all samples in D belong to the same category, the BDT generation process stops. Here Cls(.) is a function to calculate the corresponding class of y_i:

$$C = \arg\max_j \{y_i^1, y_i^2, ..., y_i^K\} \tag{2}$$

where $j \in [0, K]$ is the index that has the maximum value (i.e., 1) in the one-hot vector y_i.

Lines 6 $\sim$ 8 are the second stop condition: if the current bit set A is empty (the bits in A are removed one by one during the generation process, see line 16) or all samples have the same value (0 or 1) on all bits, the class label of node N

is assigned by the return of CntMaxCls(.). CntMaxCls(.) is a function to generate a class label from all one-hot vectors $y \in D$:

$$P = (p_1, p_2, ..., p_K) = \frac{1}{\mid D \mid} \sum_{y \in D} y, \tag{3}$$

$$C = \arg\max_j P \tag{4}$$

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 9 selects the optimal branch bit by the Gini index criterion [10]:

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

$$a_* = \arg\min_{a \in A} \text{Gini}(D, a) \tag{6}$$

where D_v contains all samples whose value of bit a is v (line 12), P_{D_v} is calculated by Equation 3 denoting the class probability vector of subset D_v, and the optimal branch bit a_* is the one that has the minimum Gini index.

Lines 10 $\sim$ 18 are the loop for tree branching. For each value $v \in \{0, 1\}$ of the optimal a_*, if the corresponding subset $\mid D_v \mid$ contains samples less than a predefined threshold (e.g., $min\_samples\_leaf = 5$), the tree generation stops (lines 13 $\sim$ 14). Otherwise, the tree will grow on the new subset D_v (line 16).

Compared with the DT, there are several advantages of the BDT. First, the input of the BDT is bits, which reduces the search space of a_*, makes the Gini index calculation quite efficient, and therefore saves training time. For example, if an attribute has m bits, the complexity of Gini calculation in the DT is $O(2^m)$, while it is $O(m)$ in the BDT (see Equation 5). Second, according to line 10 in Algorithm 1, the BDT uses zeros and ones as branch conditions. Hence, from the root to each leaf, there is a classification rule consisting of zeros and ones, which is a good fit for ternary match entries on the switch (see Section III-C for the table entry encoding). Furthermore, we will see below that using one-hot vector to represent the packet class is compatible with knowledge distillation, and only a minor alteration is required to participate in the teacher-student knowledge distillation architecture, while the DT can not be distilled.

Distilled BDT. As there have been several more complicated learning-based schemes for networking, in Mousika, we also consider a teacher-student architecture to help translating their knowledge into the BDT. Generally, for a neural network, it performs a series of complex operations (e.g., float point multiplication, pooling, and activation) on the input, getting a class probability vector. Then the input is assigned with the class label having the maximum probability in that vector. The class probability vector is also known as the soft label. Soft labels carry a lot of information summarized by the neural network. For instance, during the teacher inference, some incorrect classes may also be assigned low probabilities, which indicates that the teacher thinks they are a little similar to the correct class. This information (i.e., knowledge) has been proved to be useful in training the student model [15], [25], [26].

Thus, for a K-class classification problem, we define the class probabilities output of the teacher model as the soft label (knowledge). For all samples x_i in the transfer dataset, it is first fed to the trained teacher model to output the soft label $\hat{y_i} \in \mathbb{R}^K$, then these pairs $\{(x_1, \hat{y_1}), ..., (x_n, \hat{y_n})\}$ are used to generate the distilled BDT according to Algorithm 1. Though the alteration is straightforward, the soft label deeply affects the growth of BDT. Formally, the class probability distribution in a dataset is changed from Equation 3 to

$$\hat{P} = (\hat{p_1}, \hat{p_2}, ..., \hat{p_K}) = \frac{1}{\mid D \mid} \sum_{\hat{y} \in D} \hat{y}. \tag{7}$$

Accordingly, the calculation of Equation 4 $\sim$ 6 is changed. In other words, the knowledge from the teacher can teach the BDT where to stop branching (Line 6 $\sim$ 8 in Algorithm 1 with changed Equation 3 and 4) and which is the optimal branch bit (Line 9 with changed Equation 5 and 6). This model translation helps the BDT to mimic the teacher model and obtain a competitive performance. In addition, Mousika supports a variety of models as the teacher, provided that they have good performance and can output class probabilities.

C. Hardware Implementation

Ternary match encoding. According to Algorithm 1, for a trained/distilled BDT, its classification rules consist of zeros and ones. Each rule represents a path from the root node to the leaf node in the BDT. Starting from the root, a bit feature is checked to be one or zero and the corresponding branch is selected. This procedure is repeated until a final leaf is reached, which represents the classification target. For any path in a BDT, the number of nodes does not exceed the dimension of the input bit features. In other words, we only need to check the feature's value in specific positions according to the nodes, which is similar to the ternary match and makes it possible to transfer a BDT rule to a ternary match entry. For example, let $x = x_0 x_1 \cdots x_7$ denotes the input feature which is an 8-bit number. A rule output by the BDT is:

$$\text{If } x_0 = 1 \; and \; x_3 = 0 \; and \; x_6 = 0 \text{ Then } class \leftarrow 1.$$

As feature values in positions 0, 3, and 6 need to be considered, the ternary mask (M) is $0\text{b}10010010$ and the corresponding ternary value (V) is $0\text{b}10000000$. Given an input feature x, we just need to check if x AND $mask$ equals the ternary value to valid rule matching. If it does match, the class label (C) will be used as a parameter in the table action. Therefore, we fit the sequential decision process of a BDT into the ternary match and action of switches.

P4 program. As shown in Fig. 2, after encoding the BDT classification rules as ternary match table entries, we develop a P4 program on the switch to utilize these entries for packet classification/prediction. The P4 program follows the PISA in Fig. 1, using about 100 lines of $P4_{16}$ code. Its main parts are:

Parser. In the parser, we first select a set of features such as IP protocol, time to live, and packet length from the packet and store them in the PHV. As suggested by [5], [30], some bias features (e.g., IP/MAC address) and meaningless features (e.g., checksum) are not considered here. Then we denote a fixed-length key $bin\_feature$ to later store the above features in the form of bits. It would be desirable to initialize $bin\_feature$ with the bits in PHV, but this can not be achieved on the hardware switch. We find that concatenating multiple features is so complex and causes compile errors in the Parser. To resolve the operation restriction in Parser, we move the initialization to one pipeline stage.

Table for $bin\_feature$ initialization in the first pipeline stage. As shown in Listing 1, table $tb\_concat\_feature$ uses default action $ac\_concat\_feature$ to initialize $bin\_feature$. We declare $bin\_feature$ in the metadata which can be carried over across the pipeline stages. Since the data is stored in the form of bits in PHV, a feature can be assigned to the corresponding position of $meta.bin\_feature$. For example, the IP protocol field of one IPv4 packet can be assigned to $meta.bin\_feature$'s lower 8 bits directly (line 3 in Listing 1). To speed up the pipeline operations, programmable switches allow separate tables in the same stage have simultaneity. But we find that $bin\_feature$ initialization and packet classification can not be finished in parallel at the same stage, as these operations have a dependence on the $bin\_feature$, i.e., $bin\_feature$ should be first initialized and then used as the key for classification input. To resolve the data dependency of sequential operations, we place packet classification in a new table tb_packet_cls at the second pipeline stage.

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      /* similar code is omitted */
5  }
6  // stage 1: feature concatenation
7  table tb_concat_feature{
8      actions = {
9          ac_parse_bin_feature;
10     }
11     default_action = ac_parse_bin_feature;
12 }
```

Table for packet classification. With the table entries encoded from a BDT, the classification process is transformed into the ternary match-action of tb_packet_cls (line 8 in Listing 2). If there is a hit, the corresponding action $ac\_packet\_forward$ (line 2∼4 in Listing 2) will be triggered: We map classes to forwarding ports of the switch, and packets of different classes will be forwarded to different ports by the switch. For example, for a coming packet, if its $bin\_feature$ = 0x0000 0022 0210 0000 0000, it will match (i.e., $bin\_feature$ 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. Actually, we only provide an example of operations for packet classification here, and other operations can be easily adjusted according to the specific scenario (e.g., forwarding packets of different classes according to the predefined QoS). As for the deparser, we retain its default function defined in PISA, which is to assemble output packets according to changes in the pipeline.

Table I: 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 packet 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 packet classification
6  table tb_packet_cls {
7      key = {
8          meta.bin_feature: ternary;
9      }
10     actions = {
11         ac_packet_forward;
12     }
13     size = TABLE_SIZE;
14 }
```

IV. Evaluation

A. Networking Scenarios

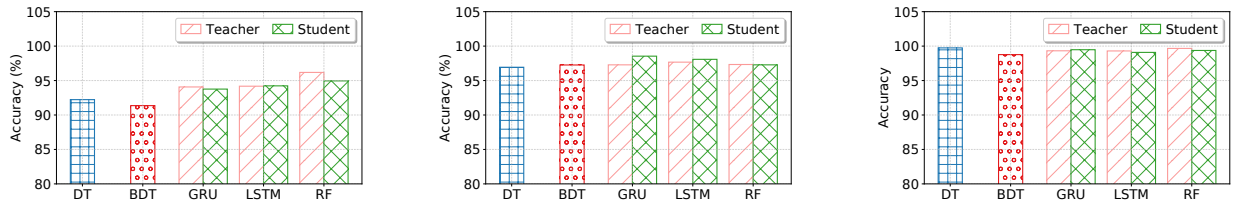
To demonstrate the performance of Mousika, we build it to identify specific target classes within the context of three networking scenarios:

- Flow size prediction. This task involves the problem of predicting the size of a flow and detecting elephant flows (very large flows whose bytes exceed a predefined threshold). Here we utilize the UNIV1 dataset made available in [31]. 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 in UNIV1 as elephants, while the other packets are mice.
- Traffic type classification. This task classifies the traffic according to the types, e.g., VoIP (Voice over Internet Protocol) and file transferring (uploading or downloading). We leverage the ISCX dataset [32] in this work and classify the packets into six types (Email, Chat, Streaming, File Transfer, VoIP, P2P).
- Malware detection. It aims to distinguish legitimate traffic from various malware attacks (such as DDoS and service scanning). In this work, we use the Bot-IoT dataset [33] and detect the packets are legitimate or malicious.

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

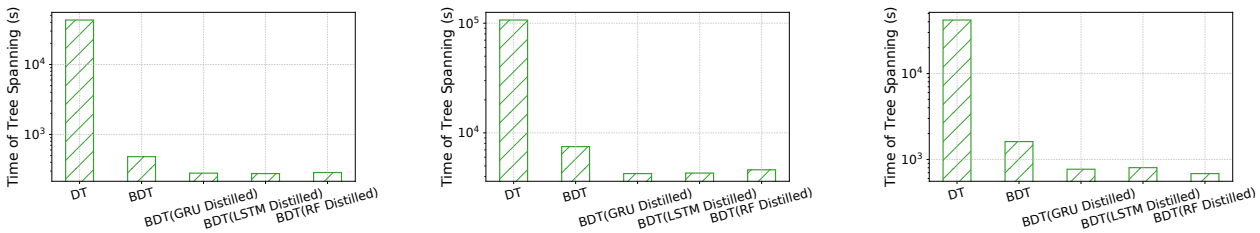
B. BDT Performance

As the BDT is a core component of Mousika, we first evaluate its performance after traditional training and distil-



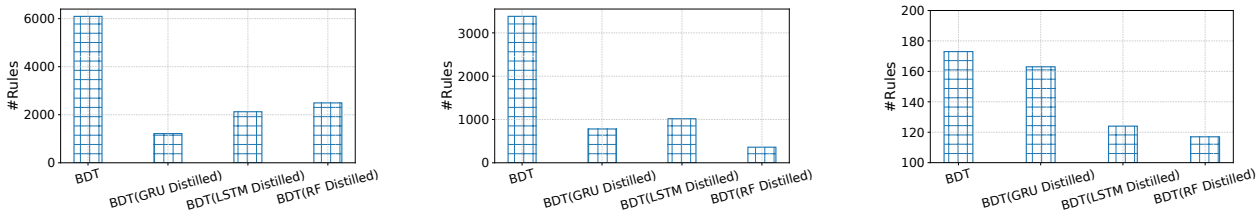
(a) Flow Size Prediction (b) Traffic Type Classification (c) Malware Detection

Fig. 3: The classification accuracy of different tree-based models on the three tasks. For knowledge distillation, we hire GRU, LSTM, and RF as the teacher and guide the BDT student training respectively.



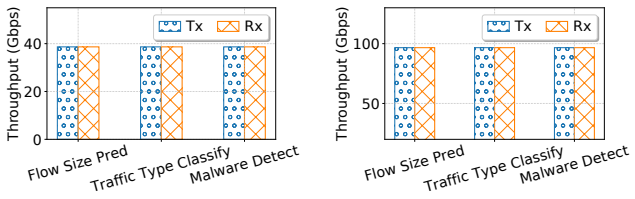
(a) Flow Size Prediction (b) Traffic Type Classification (c) Malware Detection

Fig. 4: The training time of different tree-based models on the three tasks. Content in the brackets is the name of the teacher used in the knowledge distillation architecture. For example, BDT (GRU distilled) denotes the BDT distilled from GRU.



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

Fig. 5: The classification rules of different tree-based models on the three tasks. Content in the brackets is the name of the teacher used in the knowledge distillation architecture. For example, BDT (GRU distilled) denotes the BDT distilled from GRU.



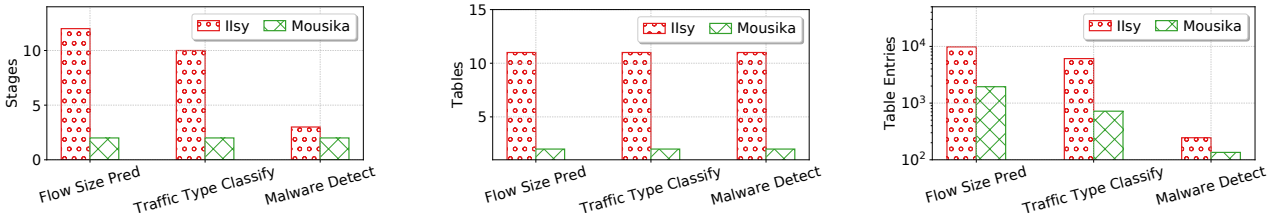
(a) 40Gbps (b) 100Gbps

Fig. 6: The receive throughput (Rx) and transmit throughput (Tx) of the switch after loading the P4 program of Mousika under the traffic speeds of 40Gbps and 100Gbps.

lation respectively. During the distillation, we mainly hire three models as the teacher respectively, including LSTM [34], GRU [35] (two deep learning models) and RF [36] (an ensemble machine learning model). Both LSTM and GRU are implemented by PyTorch[2] and run on a graphic card of GeForce RTX 2080 Ti, RF is implemented by scikit-learn[3]. We assess the quality of the DT and the BDT using the following set of metrics: 1) Accuracy, i.e., the percentage of test samples that are correctly classified in their class; 2) Training time, i.e., the time it takes for trees to grow on the training dataset; 3) Rules, i.e., the number of classification rules of the trees (not switch table entries). To be fair, both the DT and the BDT are written by ourselves in Python3 and run on a server with a CPU of Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz.

The accuracy results are depicted in Fig. 3. Among the three tasks, the DT is slightly better than the BDT on flow size prediction (92% vs. 91% in Fig. 3a) and malware detection

[2]https://pytorch.org/
[3]https://scikit-learn.org/stable/



(a) Number of Stages (b) Number of Tables (c) Number of Table Entries

Fig. 7: The program resource usage of IIsy [11] and Mousika on the same hardware switch.

(99% vs. 98% in Fig. 3c). The main reason is that the integer input of the DT is more informative than the bit input used in the BDT. But this drawback is made up by the teacher-student knowledge distillation architecture. In other words, after the knowledge is transferred from the superior teacher models to the BDT, the BDT outperforms the DT. For example, in Fig. 3a, the BDT distilled from RF improves the accuracy by 3% (from 91% to 94.95%). The same results are also revealed in Fig. 3b and 3c.

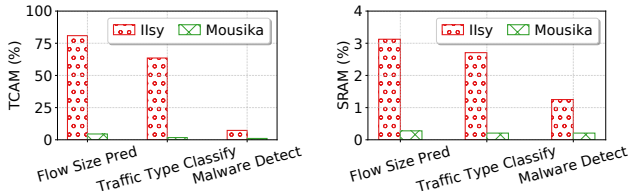
The training time [4] is shown in Fig. 4. According to Section III-B, the BDT dramatically reduces the Gini calculation complexity of the DT (from $O(2^m)$ to $O(m)$), so the BDT spends very little time on training. For example, in Fig. 4a, the training time of the BDT is 482.7s which is 89x faster than that of the DT (43170.6s). Moreover, the BDT distilled from LSTM is even fast (276.6s).The training time in Fig. 4b and 4c shows the same result, i.e., training/distilling a BDT is more efficient than a DT.

Fig. 5 shows the number of classification rules of the ordinary BDT and its knowledge distillation variant. Both Fig. 5a and 5b show that knowledge distillation helps to dramatically reduce the number of rules by several times (e.g., the BDT distilled from GRU reduces rules by $\sim$ 4.9x in flow size prediction). Since the ordinary BDT in Fig. 5c already has the least rules, the rules of the distilled BDT do not show such a significant reduction. The main reason behind this reduction is that knowledge distillation transfers the information of existing models to the BDT, which can teach the BDT to effectively find the optimal classification boundary and reduce the unnecessary rules.

C. Hardware Performance

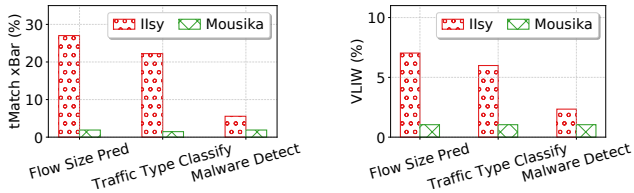
Once the BDT is trained or distilled, Mousika will encode this whole tree into several ternary match table entries. In this section, we install the table entries and the P4 program on the hardware switch (EdgeCore wedge 100BF-65X) and evaluate the efficiency by several metrics: 1) Throughput, i.e., the receive throughput (Rx) and transmit throughput (Tx) of the switch after loading the program; 2) Packet loss, i.e., the packet loss rate of the switch under a high-speed traffic (e.g., 40Gbps or 100Gbps). Here we use a traffic generator (KEYSIGHT XGS12-SDL) for high-speed traffic simulation.

Fig. 6 shows the switch throughput under the traffic speeds of 40Gbps and 100Gbps. As shown, after loading the P4 program for the three in-network tasks, the switch throughput changes little. As for the packet loss, we find that no matter how we change the traffic speed (e.g., from 40Gbps to 100Gbps), the loss is always zero. These phenomenons are mainly due to the excellent hardware characteristics of the programmable switch (e.g., the PISA in Section II-A and the ASIC Tofino chip [37]), which show the rationality and prospects of in-network intelligence.



(a) TCAM Usage (b) SRAM Usage

Fig. 8: The memory resource usage of different frameworks on the hardware switch.



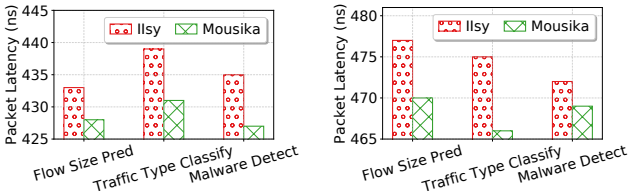
(a) tMatch xBar Usage (b) VLIW Usage

Fig. 9: The computational resource usage of IIsy [11] and Mousika on the same hardware switch. tMatch xBar denotes the ternary match crossbar, VLIW denotes the very long instruction word.

D. Compare With IIsy

To further demonstrate the performance of Mousika, we now consider the hardware resource consumption and packet

[4]For model translation, the teacher is usually pre-trained, so we mainly focus on the DT and BDT training time.



(a) 40Gbps (b) 100Gbps

Fig. 10: The time that packets are processed by IIsy [11] and Mousika on the same switch under traffic speeds of 40Gbps and 100Gbps.

processing latency between the P4 programs of Mousika and IIsy [5] (a framework that installs the DT on switches, see Section II-B). For the resource consumption, we mainly focus on the following three aspects: 1) Program resources, i.e., the number of stages, tables, and table entries; 2) Memory resources, i.e., the percentage of used TCAM and SRAM; 3) Computational resources, i.e., the percentage of used tMatch xBar and VLIW. As for packet processing latency, it refers to the time that packets are processed by the switch.

The program resource consumption is depicted in Fig. 7. Here we consider three commonly used resources, i.e., the stages (Fig. 7a), the tables (Fig. 7b), and the table entries (Fig. 7c). As shown, for each task, Mousika only takes up two stages. But IIsy occupies different numbers of stages: for the flow size prediction task, it uses eight stages; for the traffic type classification task, it uses 10 stages; for the malware detection task, it uses three stages. For the table resource, Mousika uses two tables, but IIsy uses 11 tables. For table entries, IIsy also takes up them significantly more than Mousika. For example, in Fig. 7c, the number of entries in the traffic type classification of IIsy is 6093, while the number of entries in Mousika is only 720.

The memory resource consumption is shown in Fig. 8. Since the match conditions are mainly stored in TCAM, we see that the usage of TCAM of IIsy and Mousika far exceeds the usage of SRAM. But the same kind of memory usage is completely different between IIsy and Mousika. For instance, in the traffic type classification task, the TCAM usage of IIsy and Mousika is respectively 63.5% and 1.7%; the SRAM usage of IIsy and Mousika is respectively 2.7% and 0.2%. The significant difference in memory usage between IIsy and Mousika is caused by the following two reasons: First, IIsy uses range match that is more TCAM consuming than ternary match; Second, IIsy trains the DT alone without considering the experience of the existing superior models, which results in a DT of thousands of rules.

As for the computational resources, here we consider the tMatch xBar and VLIW. tMatch xBar denotes the ternary match crossbar, VLIW denotes the very long instruction word. tMatch xBar is used to perform ternary or range match, and VLIW is used for actions. The consumption is demonstrated in Fig. 9. As shown, compared with Mousika, IIsy occupies much percentage of both tMatch xBar and VLIW. This reveals that range match used in IIsy not only consumes much TCAM but also takes up a lot of computational resources. In summary, IIsy consumes lots of hardware resources, but Mousika is more lightweight which makes it feasible to cooperate with other network management tasks (e.g., routing) on a compact switch.

The packet processing latency is depicted in Fig. 10. The packet processing latency indicates the time it takes for different frameworks to output the classification/prediction result for each packet on average. As shown, the packet processing of IIsy is slower than Mousika. For instance, at the traffic speed of 100Gbps, IIsy spends 475ns on traffic type classification per packet, but Mousika spends 466ns (faster by 9ns).

[5]As the development of in-network intelligence is at an early age, we find that IIsy is the only solution with open-source hardware implementation.

V. Conclusion and Further Discussion

In this paper, we propose Mousika, a novel in-network intelligence framework. Mousika 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 in the form of bits, which can be effectively encoded into the widely-supported ternary match. Second, we design a P4 program to use the classification rules of the BDT. This program only uses a small amount of the program resources, which is lightweight enough on the compact switch. Moreover, given the complexity of existing superior models, in Mousika, 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 complicated deployments on switches.

Though Mousika shows superior performance on the experiments, it also has several limitations. First, its input features are simply the bits of the PHV from the parser. To support other flow-level features like packet size and inter-packet frequency distributions [2], [38], several complex modifications are required, such as allocating and maintaining registers in the P4 program. In addition, knowledge distillation may not always achieve a competitive performance from the teacher model. As different models are proposed for different tasks, it is necessary to exam several models for specific tasks to ensure that teacher knowledge can have a positive impact on BDT. We leave the optimization of these limitations to future work. The authors have provided public access to their code and/or data at https://github.com/xgr19/Mousika.

VI. Acknowledgment

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 No. 61902171, and the Shenzhen Key Lab of Software Defined Networking under grant No. ZDSYS20140509172959989.

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] 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*. IEEE Computer Society, 2016, pp. 1–6.

[3] 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.

[4] R. Li, X. Xiao, S. Ni, H. Zheng, and S. Xia, "Byte segment neural network for network traffic classification," in *Proceedings of the Twenty-Sixth International Symposium on Quality of Service*. New York, USA: ACM, 2018, pp. 1–10.

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

[6] 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, 2018*. The Internet Society, 2018.

[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] Z. Meng, M. Wang, J. Bai, M. Xu, H. Mao, and H. Hu, "Interpreting deep learning-based networking systems," in *Proceedings of the 2020 Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, H. Schulzrinne and V. Misra, Eds. ACM, 2020, pp. 154–171.

[9] S. Chao, K. C. Lin, and M. Chen, "Flow classification for software-defined data centers using stream mining," *IEEE Transactions on Services Computing*, vol. 12, no. 1, pp. 105–116, 2019.

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

[11] 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*. ACM, 2019, pp. 25–33.

[12] P4 Language Consortium, "v1model.p4," Website, https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4, accessed: 2021-07-13.

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

[14] 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.

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

[16] P4 Language Consortium, "core.p4," Website, https://github.com/p4lang/p4c/blob/main/p4include/core.p4, accessed: 2021-07-13.

[17] 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*. IEEE Computer Society, 2003, pp. 42–52.

[18] 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*. ACM, 2017, pp. 15–28.

[19] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," in *Proceedings of the Annual Conference on Neural Information Processing Systems 2018*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 2499–2509.

[20] 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*, Y. Bengio and Y. LeCun, Eds., 2015.

[21] 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*, P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., 2012, pp. 1106–1114.

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

[23] 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, WWW 2018, Lyon, France, April 23-27, 2018*. ACM, 2018, pp. 1063–1072.

[24] 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.

[25] N. Frosst and G. E. Hinton, "Distilling a neural network into a soft decision tree," in *Proceedings of the First International Workshop on Comprehensibility and Explanation in AI and ML 2017 co-located with 16th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2017)*, ser. CEUR Workshop Proceedings, T. R. Besold and O. Kutz, Eds., vol. 2071. CEUR-WS.org, 2017.

[26] 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.

[27] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Proceedings of the 2014 Annual Conference on Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., 2014, pp. 2654–2662.

[28] 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*. OpenReview.net, 2017.

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

[30] 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.

[31] 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*. ACM, 2010, pp. 267–280.

[32] 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*. SciTePress, 2016, pp. 407–414.

[33] 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.

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

[35] 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.

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

[37] Barefoot Networks, "Tofino switch," Website, https://www.barefootnetworks.com/products/brief-tofino/, accessed: 2021-07-13.

[38] 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*. The Internet Society, 2021.