Distributed Multi-Task In-Network Classification on Programmable Switches by Ensemble Models

Qing Li, *Senior Member, IEEE*, Jiaye Lin, Guorui Xie, Zhongxu Guan, Zeyu Luan, Zhuyun Qi, Yong Jiang, *Member, IEEE*, and Zhenhui Yuan, *Member, IEEE*

***Abstract*—Offloading machine learning models for network classification on high-throughput programmable switches is a promising technology, enabling line-speed in-network classification. Existing solutions are centralized, deploying a complete but heavy model on a single switch with limited hardware resources, causing unsatisfactory accuracy, network-wide resource wastage, and non-generic single-task classification. Therefore, we propose In-Forest-M, a general distributed multi-task in-network classification framework. Firstly, we develop a Lightweight Ensemble Generic Optional Model (LEGO), which can be transformed into base models with full functionality. Each switch only needs to deploy lightweight base models rather than complete ensemble models. The significant reduction in resource consumption allows the deployment of larger models with higher accuracy and more models that support diverse tasks. We employ a fine-grained enhancement mechanism to enhance the classification performance of base models. As traffic traverses different switches, In-Forest-M aggregates the classification results of multiple enhanced base models to improve accuracy further. Secondly, we introduce a two-phase resource-aware model allocation strategy that assigns different task-specific enhanced base models to switches under resource constraints and task requirements. To respond to dynamic traffic changes, we design an optimization-driven reinforcement learning algorithm. Moreover, we propose a lightweight update mechanism for flexible model scaling. Comprehensive experiments reveal that, compared with state-of-the-art in-network classification solutions in three real network topologies, In-Forest-M achieves increased accuracy and reduced switch rules while exhibiting great generality in multi-task classification.**

***Index Terms*—Distributed deployment, in-network classification, programmable switch, deep reinforcement learning.**

Received 25 February 2024; revised 20 November 2024; accepted 24 June 2025; approved by IEEE TRANSACTIONS ON NETWORKING Editor K. Chowdhury. Date of publication 25 July 2025; date of current version 18 December 2025. This work was supported in part by the Project of the Department of Strategic and Advanced Interdisciplinary Research of Pengcheng Laboratory under Grant 2025QYA001, in part by the Major Key Major Key Project of Pengcheng Laboratory under Grant PCL2023A06, and in part by Shenzhen Key Laboratory of Software Defined Networking under Grant ZDSYS20140509172959989. *(Qing Li and Jiaye Lin contributed equally to this work.) (Corresponding author: Yong Jiang.)*

Qing Li, Zeyu Luan, and Zhuyun Qi are with the Pengcheng Laboratory, Shenzhen 518066, China (e-mail: liq@pcl.ac.cn; luanzy@pcl.ac.cn; qizy@pcl.ac.cn).

Jiaye Lin, Guorui Xie, Zhongxu Guan, and Yong Jiang are with the International Graduate School, Tsinghua University, Shenzhen 518055, China, and also with the Pengcheng Laboratory, Shenzhen 518066, China (e-mail: lin-jy22@mails.tsinghua.edu.cn; xgr19@mails.tsinghua.edu.cn; gzx23@mails.tsinghua.edu.cn; jiangy@sz.tsinghua.edu.cn).

Zhenhui Yuan is with the Department of Computer and Information Sciences, Northumbria University, NE1 8ST Newcastle upon Tyne, U.K. (e-mail: zhenhui.yuan@northumbria.ac.uk).

Digital Object Identifier 10.1109/TON.2025.3590275

I. INTRODUCTION

NETWORK classification based on Machine Learning (ML) gains increasing popularity for widely supporting various tasks, e.g., traffic type classification [1], [2], [3], [4], flow size prediction [5], [6], [7], [8], and anomaly attack detection [9], [10], [11], [12]. These classification results can be further exploited to boost network Quality of Service (QoS) [13], balance network load [14], or assure network security [15].

Conventional *network-assisted classification* solutions are implemented on the off-path servers, redirecting the incoming traffic to remote GPU servers and employing complex ML models (e.g., LSTM [7], Bert [16], and AutoEncoder [4]) for improved classification performance. However, they face two challenges: i) High latency. Transmitting traffic to remote servers introduces additional round-trip latency, which becomes problematic in latency-sensitive scenarios like anomaly attack detection [12], [17]. ii) Low throughput. GPU servers are more expensive but have limited packet processing capacity compared with specialized network forwarding devices (e.g., switches), particularly when handling high-speed traffic of 100Gbps or even Tbps [18], [19].

Recently, network devices (e.g., Intel Tofino switches [20]) have evolved to not only support nanosecond-level latency with Tbps-level throughput but also be programmable [13]. Several solutions offload ML models on Programmable Data Plane (PDP) [21], [22], [23], [24], [25], [26], [27], [28] to enable intelligent network classification at line-speed, i.e., *in-network classification*. They translate models into interpretable rules installed on programmable switches, facilitating on-path traffic analysis.

As rule-based classifiers, tree-based models [29], [30], [31], [32], [33], [34] fit naturally with the match-action architecture of PDP and eliminate the need for hard-to-implement operations like nonlinear activation functions [35]. Existing solutions mainly use three representation methods to deploy tree-based models, i.e., direct mapping, feature encoding, and model quantization. pForest [21] and SwitchTree [17] directly map every layer of the Decision Tree (DT) into a stage of the programmable switch. However, the limited stage number (e.g., 12 for Tofino 1 [20]) constrains the model depth, resulting in restricted classification accuracy. IIsy [22], Planter [24], and Netbeacon [36] utilize feature tables to encode features and another model table for decision-making. Although multiple tables can be stored in a stage, the feature number

must be constrained due to the memory limitation, hindering the scalability [37]. Mousika [26] and Mousikav2 [27] design a quantized DT for PDP, namely Binary Decision Tree (BDT). Despite utilizing the distillation method to transfer knowledge from complex teacher models to BDT, they encounter the challenge that the number of switch rules rises dramatically as the model size increases, leading to memory overflow [27], [36].

Crucially, all these solutions are centralized, deploying a complete but heavy model on a single switch, which causes three problems: i) Low accuracy. Due to the limited hardware resources of one single switch, it is difficult to deploy large-scale models with high accuracy [37]. ii) Redundant deployment. For network-wide deployment, the identical model is redundantly deployed on different switches to cover all the traffic. This wastes plenty of resources to repeatedly process traffic that has been processed by upstream switches without any extra benefit [38]. iii) Non-generic classification. Existing solutions solely consider single-task classification, which limits their generality in real network scenarios. For instance, to enable differentiated routing for boosted network QoS, it is essential to support tasks of traffic type classification and flow size prediction simultaneously [39].

To address the above limitations, we believe that distributing the complete model across different switches and processing traffic by multiple sub-models cooperatively is a more efficient way (discussed in § III-A). There are two potential methods: i) Layer splitting. The model can be split into layers, with each switch deploying one or more layers. ii) Rule splitting. The model can be split after being translated into switch rules, with subsets of rules assigned to different switches. However, they both require rescheduling the traffic to pass through all the sub-models for full-function analysis, introducing additional complexity in network routing and management.

In this paper, our goal is to achieve high-accuracy multi-task classification on PDP without single-point resource limitation, network-wide resource wastage, and traffic rescheduling. Therefore, we propose In-Forest-M, a general distributed multi-task in-network classification framework. Building upon our previous work,[1] In-Forest-M takes a further step towards enhancing classification performance while improving distributed deployment scalability and multi-task generality.

Firstly, we design an ensemble model that can be transformed into base models deployed on different switches, with each model providing full classification functionality. Distributed deployment eliminates the necessity of deploying a complete ensemble model on a switch with limited hardware resources, reducing the resource burden on every single switch and network-wide resource consumption. This allows for the deployment of larger models with higher accuracy and more models that support diverse tasks. In addition, it also avoids single-point failures that may cause the entire system to crash, reflecting robustness. We further employ a new fine-grained enhancement mechanism to improve the classification performance of base models. Secondly, through our designed model allocation strategy, we assign different task-specific enhanced base models to switches for multi-task classification. As traffic traverses a sequence of switches with multiple enhanced base models, In-Forest-M uses ensemble learning to aggregate classification results, gaining more comprehensive knowledge to correct the errors of individual models for improved accuracy. Moreover, we propose a lightweight update mechanism for flexible model scaling. As a whole, In-Forest-M benefits from the following key designs:

- We design an ensemble model for PDP, i.e., Lightweight Ensemble Generic Optional Model (LEGO), which consists of many simple but cooperative base models. The classification performance of base models is enhanced by a new fine-grained enhancement mechanism.
- We propose a two-phase resource-aware model allocation strategy that assigns different task-specific enhanced base models to switches, enabling multi-task classification. The offline topology-aware allocation maximizes the model diversity across switches under resource constraints and task requirements. In the online phase, we introduce a novel optimization-driven reinforcement learning algorithm that responds to dynamic traffic changes.
- We devise a lightweight update mechanism to ensure flexibility in model deployment. The rules of enhanced base models are assigned corresponding priorities, enabling optimal model scaling in/out when available resources change. In addition, each model can be updated with another one by simply modifying rules as traffic changes without restarting the switch.

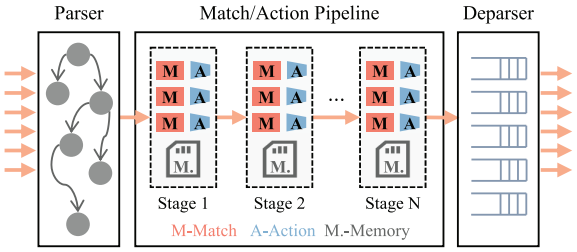


Fig. 1. The design of Protocol-Independent Switch Architecture (PISA).

II. Background

A. Programmable Data Plane

Recently, the rise of Programmable Data Plane (PDP) has further enhanced network programmability [40]. With PDP, network managers implement personalized data plane algorithms on programmable network devices through domain-specific languages (e.g., P4 [41]), supporting a wide range of high-speed network applications [28], [36]. PDP is centered on the abstract forwarding model, namely Protocol-Independent Switch Architecture (PISA) [42]. In PISA (Fig. 1), the programmable parser maps an incoming packet into a Packet Header Vector (PHV), which typically contains fields from packet headers (e.g., IP, VLAN, and TCP/UDP) and intrinsic metadata (e.g., ingress/egress ports) [13]. Then, the PHV traverses a series of pipeline stages that store match-action tables, performing lookups to match rules containing specific actions, such as forwarding, copying, dropping, and

[1]Our previous work, In-Forest, published in proc. of IEEE ICNP 2023 [38].

modifying packets. Finally, the PHV reaches the programmable deparser and is reconstructed along with the payload to form a complete packet. PDP makes in-network classification for line-speed traffic analysis a reality [15]. Despite its flexibility in programming, PDP has operational limitations. It only supports basic operations like shift, addition, and boolean, while lacking support for multiplication, looping, or floating operations [35]. Additionally, hardware resources (e.g., stage number and memory) are tight on each switch [37]. For instance, the Static Random Access Memory (SRAM) is only about 100MB on Tofino 1, while the Ternary Content Addressable Memory (TCAM) is less [36]. These limitations pose challenges in deploying large-scale models to enable in-network classification with high accuracy.

B. Ensemble Model

Ensemble models are ML algorithms that improve classification accuracy by combining multiple sub-models with relatively weak performance. They encompass a range of methods, e.g., bagging and boosting. Bagging methods, such as Random Forest (RF) [30], train sub-models in parallel on different subsets of the dataset. Sub-models possess diverse classification knowledge. The final decision of the ensemble model is determined by aggregating their results through majority voting. This process can correct the errors of sub-models, leading to improved accuracy. All sub-models are trained independently, without dependencies on each other. Boosting methods, such as Adaboost (ADB) [32] and Gradient Boosting Decision Tree (GBDT) [33], differ from bagging methods in that sub-models are trained in serial. Each sub-model focuses on correcting the misclassified samples of the preceding model, gradually improving classification performance. Thus, the construction of subsequent models depends on previous models. Ensemble models show superior performance in network classification tasks compared with individual models like DT [43]. However, the large resource consumption of ensemble models poses a challenge for in-network deployment. To address this problem, scaling down the models becomes necessary, albeit at the cost of classification performance [28].

C. In-Network Classification

For in-network classification, it is essential to offload ML models on network devices, e.g., programmable switches [13]. Since tree-based models have an architectural resemblance to PDP, most related solutions deploy them to enable intelligent traffic analysis. Table I provides a partial snapshot.

NetWarden [19] and FlowLens [44] collect traffic information on the data plane and conduct traffic analysis on the control plane. Due to the communication latency [45], traffic cannot be processed at line-speed (not LS). Besides, other works deploy models on the data plane, which embed tree-based models into PDP's match-action tables using three different representation methods, i.e., direct mapping, feature encoding, and model quantization. pForest [21] and SwitchTree [17] utilize the direct mapping method, where each layer of the DT is deployed in a stage of the switch.

TABLE I
COMPARISON OF IN-NETWORK CLASSIFICATION SOLUTIONS

Work	LS[§]	UA[§]	TA[§]	NW[§]	MT[§]
NetWarden [19]	✘	✔	✔	✘	✘
FlowLens [44]	✘	✔	✔	✘	✘
pForest [21]	✔	✘	✔	✘	✘
SwitchTree [17]	✔	✘	✔	✘	✘
IIsy [22]	✔	✘	✘	✘	✘
Planter [24]	✔	✘	✘	✘	✘
Netbeacon [36]	✔	✘	✔	✘	✘
Mousika [26]	✔	✔[¶]	✘	✘	✘
Mousikav2 [27]	✔	✔[¶]	✘	✘	✘
In-Forest-M (Ours)	✔	✔	✔	✔	✔

[§]LS: Line-Speed Processing, UA: Unrestricted Accuracy, TA: Traffic Awareness, NW: Network-Wide Assessment, MT: Multi-Task Classification.

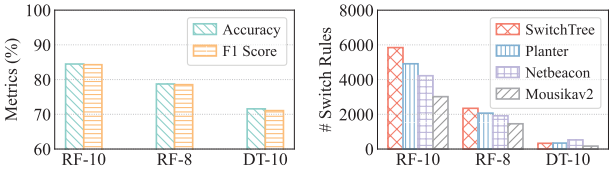
[¶]In some tasks, Mousika and Mousikav2 necessitate more nodes of BDT, which causes switch rules to overflow the limited memory [27], [36].

The model depth is limited by the stage number, making it challenging to accommodate larger models with higher accuracy. The method employed by IIsy [22], [23], Planter [24], [25], and Netbeacon [36] is feature encoding, which uses feature tables to encode features and another model table for decision-making. The feature number must be constrained, otherwise tables will take up excessive memory [46]. Both methods face a challenge in installing a huge number of switch rules translated from large-scale models due to limited hardware resources, resulting in restricted accuracy (not UA). Mousika [26] and Mousikav2 [27] design a quantized model for PDP, namely BDT. They apply the distillation method to transfer knowledge from complex teacher models to BDT (note that they require more switch rules in some tasks, leading to memory overflow [27], [36]). However, the same as IIsy and Planter, they lack support for dynamic traffic changes, hindering adaptability (not TA). Compared with the few closest works that are centralized, In-Forest-M takes a step further by introducing distributed deployment. Extending them for network-wide deployment leads to huge resource wastage as the identical model is required to be redundantly deployed on different switches to cover all the traffic (not NW). Traffic processed by upstream switches continues to traverse the same pipeline without any additional benefits [38]. Moreover, real network scenarios often involve various classification tasks simultaneously, such as traffic type classification and anomaly attack detection [46]. However, existing solutions concentrate on completing a single task and fail to fully utilize the available resources for multi-task classification (not MT).

III. IN-DEPTH ANALYTICS-DRIVEN MOTIVATION

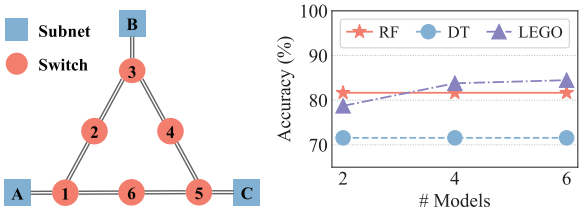
A. Motivation

1) Ensemble Models Outperform Individual Models But With Large Resource Consumption: Ensemble models have advantages in terms of classification performance and robustness. We take anomaly attack detection as an example and evaluate models using raw pcap files of network traffic [47]. Flow-level features are extracted to form the dataset, which is divided into 80% for training and 20% for testing. We



(a) Classification Performance (b) Number of Switch Rules

Fig. 2. The comparison of RF and DT in anomaly attack detection. RF-x and DT-x denote the maximum depth of RF and DT is x. Models are translated into switch rules by four existing in-network classification solutions.



(a) Example Topology (b) Model Comparison

Fig. 3. The preliminary experiments for network-wide deployment. Enhanced base models of LEGO are deployed on different switches.

employ an ensemble model (RF) and an individual model (DT) trained by the widely-used ML framework scikit-learn [48] to classify benign activity and specific malicious attacks. Fig. 2a demonstrates that RF outperforms DT in terms of accuracy. Both RF-10 and DT-10 have a maximum depth of 10, where RF comprises 10 sub-models. For sub-models of RF, we reduce the maximum depth from 10 to 8, decreasing their performance intentionally. Despite the reduction, RF-8 is still superior to DT-10. This case shows us that i) ensemble models outperform individual models in network classification tasks, and ii) the performance of sub-models directly affects the performance of the ensemble model. Fig. 2b presents the number of required switch rules to deploy RF and DT, where models are translated by existing in-network classification solutions, i.e., SwitchTree [17], Planter [24], Netbeacon [36], and Mousikav2 [27]. For instance, RF requires 17.66× more rules than DT, i.e., 5844 (RF-10) vs. 331 (DT-10), in SwitchTree. The huge resource consumption hinders the pratical deployment of large-scale ensemble models on a single switch.

2) Distributed Deployment Reduces Hardware Resource Consumption and Improves Classification Performance: We consider the network-wide deployment of DT and RF. Fig. 3a depicts an example topology with three subnets A∼C and six switches $1 \sim 6$. Traffic between subnet pairs is routed based on the Open Shortest Path First (OSPF) protocol [49]. To cover all the traffic, models need to be deployed on two switches, such as DT on switches 1 and 5. However, this deployment results in a slow reaction of traffic transmitted from subnet B to A (or B to C) [46]. Thus, an additional model needs to be deployed, such as DT on switch 3. Notably, two identical models are used to process traffic between subnet pair A-B (as well as B-C and A-C), wasting resources and not improving accuracy. Furthermore, as the number of subnets increases, it will lead to more huge resource wastage (Figure 4).

A more efficient solution is to deploy sub-models on different switches and allow traffic to traverse most of them.

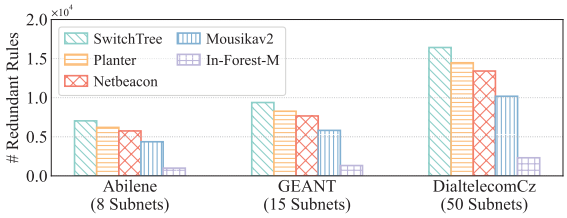


Fig. 4. The number of redundant paths in network topologies with different scales. RF is chosen as the model deployed by existing in-network classification solutions, while In-Forest-M utilizes LEGO for distributed deployment.

By aggregating the classification results of sub-models, we can combine their strengths to correct the errors of individual models, thereby improving accuracy without single-point resource limitation and network-wide resource wastage. We develop a Lightweight Ensemble Generic Optional Model (LEGO) consisting of multiple enhanced base models with full classification functionality. Enhanced base models are deployed on different switches for distributed deployment. Fig. 3b demonstrates that LEGO outperforms DT, approaching or even surpassing RF's performance as the number of enhanced base models increases. This improvement is attributed to the fact that the accuracy of RF is restricted by the limited single-point resources, whereas LEGO leverages the resources of multiple switches, achieving higher accuracy through aggregation.

Fig. 4 illustrates network-wide resource wastage in real network topologies. Abilene [50], GEANT [51], and DialtelecomCz [52] are connected to different numbers of subnet pairs. The number of redundant rules denotes the average number of identical rules traversed by each flow. RF is chosen as the model deployed by existing centralized solutions, while In-Forest-M utilizes LEGO for distributed deployment. Centralized solutions exhibit more redundant paths than In-Forest-M, leading to significant resource wastage. This issue becomes more prominent as the network topology scales up.

3) Distributed Deployment Empowers Flexible Adjustment and Multi-Task Classification: Under varying resource constraints and dynamic traffic changes, the allocation schemes of enhanced base models can be flexibly adjusted. As traffic traverses multiple switches with different enhanced base models, In-Forest-M uses ensemble learning to aggregate classification results. Compared with centralized deployment, distributed deployment eliminates the single-point resource limitation, enabling the deployment of larger, high-accuracy models. For example, we can deploy three optimal models on switches $1 \sim 3$ to best serve the traffic between subnet pair A-B with limited resources. Alternatively, we can also deploy different models on all switches to guarantee network-wide traffic coverage and classification accuracy. Additionally, In-Forest-M consumes significantly fewer resources, allowing the assignment of task-specific enhanced base models to switches for multi-task classification, i.e., different models tailored to different tasks. Multiple task-specific models can be deployed on a single switch, enabling efficient multi-task handling and resource optimization.

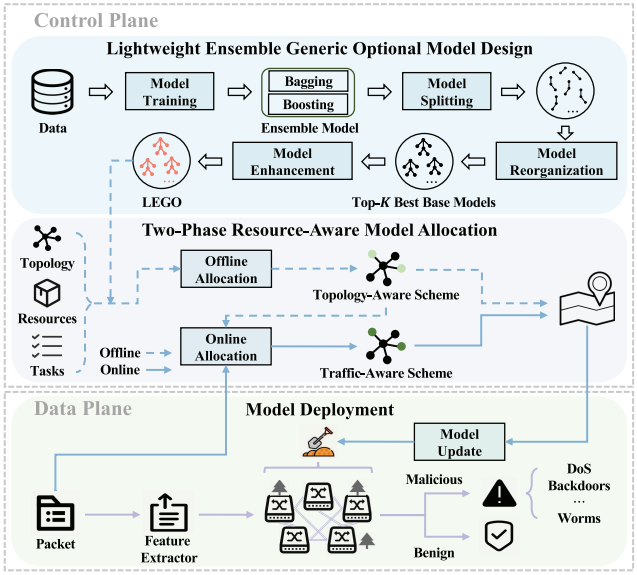


Fig. 5. The architecture of In-Forest-M, which consists of three modules.

B. Design Goals

Compared with existing solutions, we propose In-Forest-M, a distributed multi-task deployment framework that addresses the challenges of single-point resource limitation and network-wide resource wastage. Through cooperation, each switch can achieve high-accuracy classification by deploying only enhanced base models rather than complete ensemble models. In-Forest-M supports multi-task classification, which allows a single switch to handle different tasks simultaneously.

IV. The Design of In-Forest-M

A. System Overview

In-Forest-M is a general distributed multi-task in-network classification framework that leverages the available resources of multiple switches to deploy large-scale ensemble models. As Fig. 5 shows, on the control plane, we redesign conventional ensemble models into a Lightweight Ensemble Generic Optional Model (LEGO) for distributed deployment, which consists of multiple enhanced base models with full classification functionality (§ IV-B). To achieve optimal model allocation under various resource constraints, specific task requirements, and dynamic traffic changes, we employ a two-phase resource-aware model allocation strategy (§ IV-C). On the data plane, we translate enhanced base models into interpretable rules installed on programmable switches to enable line-speed traffic analysis and introduce a lightweight model update mechanism to ensure flexibility (§ IV-D).

B. LEGO Design Module

In-Forest-M uses a "splitting-reorganization-enhancement" mechanism to generate LEGO, as depicted in Algorithm 1.

Path-Based Model Splitting. Given the raw packet set $\mathcal{C} = \{(\boldsymbol{c}_1, y_1), \ldots\}$, with y_i indicating the class of packet $\boldsymbol{c}_i$, we assign a flow ID h_i to each packet by hashing its features, such as the 5-tuple. The initial F packets per flow are stored in the set $\mathcal{F}$ (lines 1∼ 1). We extract crucial flow-level features for network classification [36] based on the given

Algorithm 1 LEGO Design Logic

```
   // ***** Model Splitting *****
1  F ← {[]};
2  for i = 1, ..., |C| do
3      h_i ← Hash(GetFiveTuple(c_i));
4      if F[h_i].length < F then F[h_i].append(c_i);
5  end
6  X ← FeatureExtraction(F, U);
7  U' ← BackwardFeatureSelection(X, U, S);
8  G ← Training(X, U');
9  PathPool ← ModelSplitting(G);
   // ***** Model Reorganization *****
10 B, B' ← ModelReorganization(PathPool), [];
11 for i = 1, ..., |B| do
12     if KBestFiltering(B_i) then B'.append(B_i);
13 end
   // ***** Model Enhancement *****
14 I_0 ← GetIndex(PathPool, B \ B');
15 Supplements ← {[]};
16 for k = 1, ..., |B'| do
17     I_k ← GetIndex(PathPool, B'_k);
18     PathInsert ← [];
19     for j ∈ I_0 do
20         for o ∈ I_k do
21             Pr_jo ← c_j + γ(c_j − c_o);
22             PathInsert.append((j, o, Pr_jo));
23         end
24     end
25     PathInsert.Sort(Pr, descending = True);
26     for i = 1, ..., |PathInsert| do
27     end
28     j, o = PathInsert[i][0], PathInsert[i][1];
29     if CanInsert(j, o) then
30         Supplements[k].append(P_j ∩ P_o);
31         P_o ← P_o \ (P_j ∩ P_o);
32     end
33 end
34 B'' ← Combine(B', Supplements);
35 LEGO ← ModelAggregation(B'');
```

feature set $\mathcal{U}$, forming the training set $\mathcal{X} = \{(\boldsymbol{x}_1, y_1), \ldots\}$, where $\boldsymbol{x}_i$ denotes the flow-level features of h_i (line 1). A backward recursive method [53] is used to prune redundant features (line 1), which iteratively removes the least impactful one based on cross-validation scores until the feature number reaches S. S denotes the number of selected features determined by feature importance [54], balancing model complexity and accuracy. The training set $\mathcal{X}$ and the selected feature set $\mathcal{U}'$ are used to train the tree-based ensemble model $\mathcal{G}$ through bagging or boosting methods (line 1). For classification, the process starts at the root node (i.e., the first internal node) and traverses a sequence of internal nodes. Each internal node contains a classification feature and a threshold. The sample's feature is compared with the threshold, determining its direction towards the left or right branch. This process continues until reaching a leaf node, which contains the classification result. Consider a process with the following

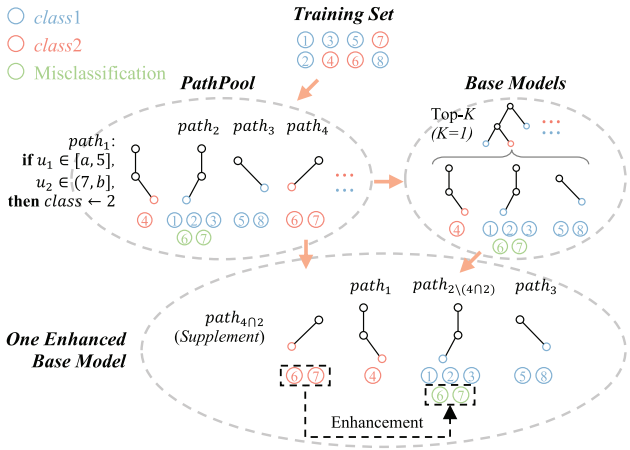


Fig. 6. An example of the LEGO design, where only one enhanced base model is shown, i.e., $K = 1$. Leaf nodes in paths output the predicted classification results (blue for $class1$ and red for $class2$), with each path corresponding to a subset of traffic. The enhancement mechanism inserts valuable paths to correct misclassifications (e.g., samples 6 and 7).

knowledge: if $u_1 \leq 5, u_2 > 7$, then $class \leftarrow 2$. We can encode it as a classification path:

$$\text{if } u_1 \in [a, 5], u_2 \in (7, b], \text{ then } class \leftarrow 2, \tag{1}$$

where a denotes the minimum value of feature u_1 and b denotes the maximum value of feature u_2. The ensemble model $\mathcal{G}$ is split into multiple classification paths and stored in *PathPool*, as depicted in Fig. 6 (line 1).

Coarse-Grained Model Reorganization. The classification paths correspond to distinct traffic subsets. We reorganize the paths in *PathPool* into multiple base models $\mathcal{B}$ (line 1), each of which possesses full classification functionality. Then, we apply the Top-*K* filtering method to select *K* base models with the best classification performance, i.e., $\mathcal{B}'$ (lines 1∼ 1). To align with network topologies, we define $K \triangleq \min(|\mathcal{B}|, H)$, where *H* is the maximum number of switches between subnet pairs. For Abilene [50], GEANT [51], and DialtelecomCz [52] topologies, *H* is 6, 8, and 15, respectively. Take Fig. 6 as an example, where we consider only one enhanced base model, i.e., $K = 1$. The combination of paths $1 \sim 3$ possesses full functionality to enable the classification task, which forms a base model (upper right ellipse). Then, it is picked as one of the Top-*K* best base models through the filtering method.

Fine-Grained Model Enhancement. Motivated by insights discussed in § III-A1, we aim to enhance base models for improved ensemble performance. Initially, we obtain indexes of the classification paths from $\mathcal{B} \setminus \mathcal{B}'$ (i.e., base models not chosen by the filtering method), denoted as $\mathcal{I}_0$ (line 1). Subsequently, for each base model $\mathcal{B}'_k \in \mathcal{B}'$, we selectively insert paths from $\mathcal{B} \setminus \mathcal{B}'$ for enhancement (lines 1∼ 1).

Specifically, we define indexes of the classification paths from $\mathcal{B}'_k$ as $\mathcal{I}_k$ (line 1). For each path $P_j \in \mathcal{B} \setminus \mathcal{B}'$ and the original path $P_o \in \mathcal{B}'_k$, where $j \in \mathcal{I}_0$ and $o \in \mathcal{I}_k$, we calculate the insertion priority as follows:

$$Pr_{jo} \leftarrow c_j + \gamma(c_j - c_o), \tag{2}$$

where c_j and c_o denote the numbers of correct classifications for P_j and P_o in $\mathcal{X}_j \cap \mathcal{X}_o$, respectively. $\mathcal{X}_j$ and $\mathcal{X}_o$ denote the corresponding traffic subsets of classification paths. γ is the enhancement rate. The indexes and the priority are stored in *PathInsert*. We sort *PathInsert* in descending order according to the priority (lines 1∼ 1), for ensuring the earlier inserted path brings a greater accuracy improvement. In § V-B, we demonstrate this and discuss the values of γ.

Then, we insert paths sequentially. The function CanInsert(·) is used to calculate c_j and c_o for checking whether the path should be inserted or not. If $c_j > c_o$, we add $P_j \cap P_o$ into the *Supplement* of B'_k. $P_j \cap P_o$ is the intersection path of P_j and P_o. Moreover, we modify the original path P_o to $P_o \backslash (P_j \cap P_o)$, making sure different classification paths from the same base model do not process the same traffic (lines 1∼ 1).

For greater clarity, we take an example to show the workflow of model enhancement. In Fig. 6, for classification paths whose base models are not chosen by the Top-*K* filtering method, e.g., $path_4$, they remain valuable. $path_4$ accurately classifies samples 6 and 7 and can serve as a *Supplement* to correct misclassifications (colored in green) of $path_2$. The number of correct classifications for $path_4$ in $\mathcal{X}_j \cap \mathcal{X}_o$ (i.e., samples 6 and 7) is 2, which is 0 for $path_2$. Due to $c_4 > c_2$, we can insert $path_4$. To avoid two different paths corresponding to the same traffic, we create a new path $path_{4\cap2}$ by taking the feature intersection of $path_4$ and $path_2$. Then, we modify $path_2$ to be the difference set $path_{2\backslash(4\cap2)}$. As a result, samples 6 and 7 are correctly classified by $path_{4\cap2}$ instead of the original $path_2$. After that, we combine base models $\mathcal{B}'$ with *Supplements* to get enhanced base models $\mathcal{B}''$ (line 1). Each enhanced base model comprises multiple classification paths (from base models $\mathcal{B}'$ and *Supplements*), which are sequences of feature discriminants, as shown in Equation (1). For each flow to be classified, different classification paths are traversed, and flow-level features are compared with the threshold to determine the final result. Notably, these classification paths are translated into range match rules and installed in programmable switches [26], [55] (detailed in § IV-D). LEGO is further obtained by aggregating $\mathcal{B}''$ to prevent overfitting [30] for improved accuracy (line 1).

In summary, our base model is a variant of DT. Compared with traditional DT, we utilize path-based model splitting, coarse-grained model reorganization, and fine-grained model enhancement to optimize the models, improving their suitability for distributed deployment with high accuracy. The suitability is characterized by four key features: i) Lightweight. Through feature selection, path splitting, and model filtering, LEGO reduces resource consumption while retaining the full classification functionality of each base model. ii) Ensemble/Enhanced. The accuracy of the ensemble model (i.e., LEGO) is improved by enhancing base models through inserting valuable paths. iii) Generic. The classification paths are obtained from bagging or boosting methods, and the number of insertion paths is flexibly adjusted under different resource constraints. iv) Optional. Depending on the task requirements and traffic changes, enhanced base models can be selectively aggregated to achieve superior classification performance.

It is worth noting that if adapting other types of models to LEGO, e.g., NN-based models, they can be converted into tree-based models using our previous work, Mousika [26]

TABLE II
SUMMARY OF VARIABLES IN THE MODEL ALLOCATION PROBLEM

Variable	Description
$Tasks$	The task requirement
V	The number of tasks in $Tasks$
K	The number of enhanced base models
W	The number of switches
N	The number of subnet pairs
M	The maximum number of deployed switches
E	The maximum number of network-wide rules
$\mathcal{B}_v''$	The classification model for task v
$\mathcal{D}v,k,w$	Whether to deploy the k-th model of $\mathcal{B}_v''$ on switch w
$\mathcal{O}_{v,k}$	The number of switch rules required for k-th model in $\mathcal{B}_v''$
$\mathcal{P}_{n,w}$	Whether switch w is on selected path between subnet pair n

and Mousikav2 [27], before applying LEGO for splitting and enhancement, which further demonstrates the generality.

C. Two-Phase Resource-Aware Model Allocation Module

When deploying LEGO across the entire network, metrics like traffic coverage, classification accuracy, and hardware resource consumption are vital in determining the allocation scheme of enhanced base models. However, directly modeling the correspondence between the scheme and network traffic is challenging. Due to the dynamic nature of traffic, the current allocation scheme may not adapt to future traffic. To overcome this, we employ a two-phase resource-aware model allocation strategy, including an offline phase and an online phase.

Offline Topology-Aware Allocation. In-Forest-M employs the offline phase to obtain topology-aware model allocation schemes under different resource constraints. Compared with our previous work [38], we additionally accommodate multi-task classification scenarios. Different task-specific models can be deployed on a single switch to enable simultaneous handling of multiple tasks, optimizing resource utilization.

Table II summarizes the variables. The task requirement is denoted as *Tasks*, consisting of *V* different tasks specified by network managers. For each task *v*, the corresponding classification model is denoted as $\mathcal{B}_v''$. By converting traffic coverage to path coverage between subnet pairs and classification accuracy to model diversity across switches, we formulate the objective function for task *v* as follows:

$$\begin{aligned} obj_v = \alpha_1 \sum_{n=1}^{N} \text{Step}\left(\sum_{w=1}^{W} \mathcal{P}_{n,w} \sum_{k=1}^{K} \mathcal{D}_{v,k,w}\right) \\ + \alpha_2 \sum_{n=1}^{N} \sum_{k=1}^{K} \text{Step}\left(\sum_{w=1}^{W} \mathcal{P}_{n,w} \mathcal{D}_{v,k,w}\right) \\ - \alpha_3 \sum_{k=1}^{K} \sum_{w=1}^{W} \mathcal{D}_{v,k,w} \mathcal{O}_{v,k}. \end{aligned} \tag{3}$$

We aim to maximize the objective function, i.e., maximize path coverage (term 1) and model diversity (term 2), while minimizing the total number of required switch rules (term 3). The model allocation scheme is denoted as $\mathcal{D} \in \mathbb{R}^{V \times K \times W}$. The number of enhanced base models i.e., *K*, is defined in § IV-B. *W* is set to the number of switches in real network topologies, i.e., 11 for Abilene [50], 23 for GEANT [51], and 106 for DialtelecomCz [52]. The binary-encoded matrix $\mathcal{P} \in \mathbb{R}^{N \times W}$ indicates whether switches lie on the selected routing path, where *N* is the number of subnet pairs. The function Step(·) converts values that are greater than 0 to 1 and others to 0. To ensure comparability among the three terms of metrics, we normalize them into the range of $[0, 1]$, with α_1, α_2, and α_3 representing the respective weights. Then, we define the multi-task model allocation problem as follows:

$$\max_{\mathcal{D}} \quad \sum_{v=1}^{V} obj_v \tag{4}$$

$$s.t. \quad \sum_{w=1}^{W} \text{Step}\left(\sum_{v=1}^{V}\sum_{k=1}^{K} \mathcal{D}_{v,k,w}\right) \le M; \tag{4a}$$

$$\sum_{v=1}^{V}\sum_{k=1}^{K}\sum_{w=1}^{W} \mathcal{D}_{v,k,w}\mathcal{O}_{v,k} \le E; \tag{4b}$$

$$\sum_{k=1}^{K} \mathcal{D}_{v,k,w} \le 1, \ \forall v, \ \forall w; \tag{4c}$$

$$\mathcal{D}_{v,k,w} \in \{0, 1\}, \ \forall v, \ \forall k, \ \forall w. \tag{4d}$$

Equations (4a) and (4b) are employed to enforce resource limitations on the model allocation scheme. *M* and *E* denote the maximum number of deployed switches and the maximum number of network-wide rules, corresponding to distinct resource constraints. For each task, we restrict the deployment of at most one enhanced base model per switch (Equation (4c)). Notably, a single switch can accommodate multiple models that support different tasks for multi-task classification.

The multi-task model allocation problem has a similar definition to the multiple knapsack problem [56]: When assigning an enhanced base model (item) to different switches (knapsacks), a corresponding gain is yielded, and the objective is to maximize the overall gain. The multiple knapsack problem is known as NP-hard, and hence our problem is also NP-hard. Since existing studies [57], [58], [59], [60] employ a heuristic algorithm, i.e., Genetic Algorithm (GA), for solving the multiple knapsack problem, we share the same design.

We generate a diverse initial population by randomly creating multiple instances of $\mathcal{D}$ with different values. Each instance is then flattened into $\tilde{\mathcal{D}} \in \mathbb{R}^{1 \times Q}$, where $Q = V \times K \times W$. These instances represent candidate model allocation schemes, which are evaluated by the fitness (i.e., the objective function in Equation (4)). Through iterative evolution, schemes with higher fitness are more likely to reproduce in subsequent iterations. Offspring are generated through crossover and mutation, replacing partial schemes to maintain the population size. This process continues until fitness converges. To ensure the satisfaction of constraints (4a)∼ (4d), we penalize non-compliant schemes by assigning a large negative value to the fitness. The scheme with the highest fitness is deemed the optimal model allocation scheme in the offline phase.

Online Traffic-Aware Allocation. To improve the adaptability of In-Forest-M to dynamic traffic changes, we introduce an online phase. By selecting suitable enhanced base models to classify corresponding traffic, we can further enhance the accuracy. GA faces limitations in the online phase due to its high time complexity and the inability to anticipate long-term returns. To overcome these challenges, we employ a model-free Deep Reinforcement Learning (DRL) approach [61]. DRL excels at solving online decision-making problems by optimizing both current and future rewards, enabling fast adaptation to dynamic environments [50], [62].

We formulate the traffic transmission as a Markov Decision Process (MDP). The MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T})$ [50], where $\mathcal{S}$ represents the state space, $\mathcal{A}$ represents the action space, $\mathcal{R}$ represents the reward function, and $\mathcal{T}$ represents the state transition probability. In-Forest-M receives network information $\boldsymbol{s} \in \mathcal{S}$ (for notational simplicity, we omit the subscript t) and determines the model allocation scheme $\boldsymbol{a} \in \mathcal{A}$, which then gets a reward r. The MDP aim to find an optimal policy $\pi_{\boldsymbol{\theta}}$ that maximizes the objective function $\mathbf{J}(\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ denotes the policy parameters.

We adopt the Proximal Policy Optimization (PPO) algorithm to update $\pi_{\boldsymbol{\theta}}$ [61] due to its enhanced sampling efficiency and reduced training variance. PPO leverages importance sampling to efficiently utilize the sampled data based on the old policy parameters $\boldsymbol{\theta}'$. The update of the policy $\pi_{\boldsymbol{\theta}}$ can be expressed as follows:

$$\nabla_{\boldsymbol{\theta}} \mathbf{J}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\tau} \sim \mathbf{P}_{\boldsymbol{\theta}'}(\boldsymbol{\tau})}[\mathbf{W}(\boldsymbol{\theta}') \mathbf{A}(\boldsymbol{s}, \boldsymbol{a}) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})], \quad (5)$$

where $\boldsymbol{\tau}$ is the state-action pair $(\boldsymbol{s}, \boldsymbol{a})$ sampled from $\mathbf{P}_{\boldsymbol{\theta}'}(\boldsymbol{\tau})$. The advantage function $\mathbf{A}(\cdot)$ quantifies the superiority of the selected action compared to others within a given state [63]. The importance weight, denoted as $\mathbf{W}(\boldsymbol{\theta}') = \frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\boldsymbol{\theta}'}(\boldsymbol{a}|\boldsymbol{s})}$.

It is crucial to emphasize that the new policy parameters $\boldsymbol{\theta}$ and the old policy parameters $\boldsymbol{\theta}'$ should not deviate significantly. We modify the objective function to limit the gap between them. The updated objective function is:

$$\begin{aligned} &\mathbf{J}^{CLIP}(\boldsymbol{\theta}) \\ &= \mathbb{E}_{\boldsymbol{\tau} \sim \mathbf{P}_{\boldsymbol{\theta}'}(\boldsymbol{\tau})}[\min(\mathbf{W}(\boldsymbol{\theta}') \mathbf{A}(\boldsymbol{s}, \boldsymbol{a}), \mathbf{W}'(\boldsymbol{\theta}') \mathbf{A}(\boldsymbol{s}, \boldsymbol{a}))], \end{aligned} \quad (6)$$

where $\mathbf{W}'(\boldsymbol{\theta}') = \mathrm{Clip}(\mathbf{W}(\boldsymbol{\theta}'), 1-\epsilon, 1+\epsilon)$ is used to prevent a significant increase (or decrease) in the probability of good (or bad) actions [61]. Here, ϵ is set to 0.2.

We introduce the Covered Flow Accuracy (CFA) metric as the reward function to assess the model allocation scheme:

$$CFA \triangleq \frac{\sum_{l=1}^{L} m_l (y_l == \hat{y}_l)}{L}, \quad (7)$$

where m_l is a binary variable indicating whether flow l is covered by any enhanced base models and L is the number of flows. y_l and $\hat{y}_l$ are the true and predicted classes, respectively.

In the state space $\mathcal{S}$, we consider the flow feature (5)-tuple of each flow), the traffic distribution (flow number in links), and the current model allocation scheme. Regarding the action space $\mathcal{A}_v \in \mathcal{A}$ for task v, the action output from each switch is a vector of $K+1$ dimensions. The first K dimensions denote the deployment probability of each enhanced base model, while the last dimension represents the non-deployment probability. To determine the model allocation scheme in the online phase, we select the action with the highest probability on each switch and combine them as the global action.

Fig. 7 shows the pipeline of online traffic-aware allocation. Following [50], we assume a discrete-time model where time is partitioned into consecutive timesteps, i.e., $t = 1, 2, \ldots$. The t-th flow arrives at the beginning of timestep t and the model allocation scheme is updated every 100 timesteps. To ensure the convergence of the online phase, we employ an optimization-driven mechanism. We determine the deployed switches through the offline phase and then use the online phase to tune the enhanced base models deployed on them. This can reduce the dimensions of action space for faster and more stable convergence. The optimal allocation scheme in the offline phase serves as an initial value for PPO, which explores and refines the scheme. If the action output does not outperform the offline phase, the model allocation scheme remains unchanged. The scheme in the online phase is also required to meet the constraints. We set a penalty value (i.e., -1) as the reward for schemes with lower performance than the offline phase or that do not satisfy the constraints (4a)∼(4d). The checked online action interacts with the environment, and then samples are collected to update the policy.

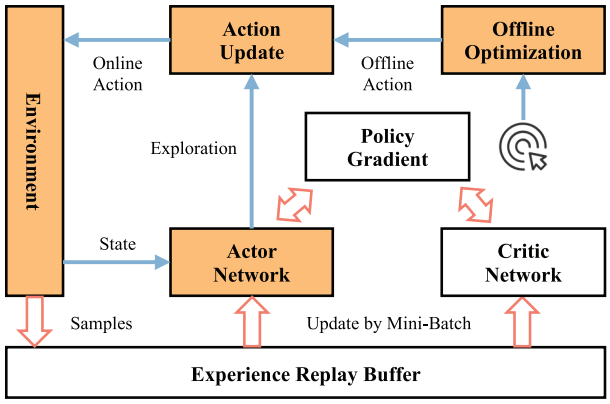


Fig. 7. The pipeline of online traffic-aware allocation. Colored rectangles indicate the optimization-driven mechanism.

All in all, our two-phase resource-aware model allocation strategy consists of an offline phase and an online phase. The offline phase provides topology-aware model allocation schemes under different resource constraints, serving as a solid base. The online phase is followed to fine-tune the scheme for enhanced adaptability. The offline phase is topology-aware, employing a heuristic algorithm GA to address the multi-task model allocation problem, and the online phase is traffic-aware, utilizing DRL to solve online decision-making problems by optimizing both current and future rewards. The base scheme from the offline phase aids faster and more stable convergence in the online phase, while the online phase achieves higher accuracy by adapting to dynamic traffic.

D. Model Deployment Module

In-Forest-M deploys enhanced base models on different programmable switches to enable distributed in-network classification. Each downstream switch aggregates the classification results of models deployed on upstream switches, improving accuracy. Furthermore, we devise a lightweight model update mechanism to ensure flexibility.

Model Representation. In-Forest-M employs flow-level features (e.g., average/minimum/maximum packet lengths) for classification, as detailed in [36]. These features are derived by combining attributes from a sequence of packets within the same flow. To balance memory consumption and classification accuracy, we extract features of the first F packets per flow, where F is set to 4 [64]. After extracting the features, their values are utilized as input to the models. Each enhanced base model can be deployed as a single match-action table that

```
1  // ***** Task1 *****
2  action Set_Task1_Cls1(result){
3      meta.res = result;
4      meta.prob1 = meta.prob1 + 1;
5      meta.prob2 = meta.prob2 - 1;}
6  action Set_Task1_Cls2(result){
7      meta.res = result;
8      meta.prob2 = meta.prob2 + 1;
9      meta.prob1 = meta.prob1 - 1;}
10 table Get_Task1_Res{
11     key = {
12         meta.f1: range;
13         meta.f2: range;
14         ...}
15     actions = {
16         Set_Task1_Cls1;
17         Set_Task1_Cls2;}}
18 // ***** Task2 *****
19 action Set_Task2_Cls1(result){...}
20 action Set_Task2_Cls2(result){...}
21 table Get_Task2_Res{...}
22 ...
```

Listing 1. P4 pseudocode for model deployment.

matches all the features. The classification paths, as shown in Equation (1), are translated into range match rules and stored in the table (lines $10 \sim 17$ in Listing 1). If there is a matched rule in table Get_Task1_Res, the corresponding action is triggered, e.g., Set_Task1_Cls1 (lines $2 \sim 5$ in Listing 1).

Result Aggregation. We utilize diverse packet header fields to record classification results, where the results from upstream switches are aggregated on each downstream switch. For a classification task with Z distinct classes, we define Z variables, namely $prob1, \ldots, probZ$. When the model outputs a result of class z, we increment the z-th variable by 1 while decrementing the other variables by 1.

Suppose a task involves two classes of flows, namely $class1$ and $class2$. We define two actions, i.e., Set_Task1_Cls1 and Set_Task1_Cls2, along with two variables, i.e., $prob1$ and $prob2$. Variables are initialized with the value of H, where H denotes the maximum number of switches between subnet pairs. If the model classifies a flow as $class1$, the action Set_Task1_Cls1 is triggered, resulting in an increment of $prob1$ by 1 and a decrement of $prob2$ by 1. Subsequently, we place the aggregate table in an additional switch stage (our P4 program requires 2 stages in total). The aggregation operation can be implemented by range match rules [55], which look up variables with values in $[H+1, 2H]$. If $prob1$ exceeds H, it indicates that at least one model classifies the flow as $class1$, and a larger $prob1$ signifies higher confidence. When $prob1$ and $prob2$ both equal H, the flow continues to be forwarded, indicating that the class cannot be determined at this point. Note that this aggregation operation does not introduce an excessive number of switch rules (only Z more rules). In addition, we implement corresponding processing solutions for abnormal traffic. By aggregating the results of enhanced base models, we can achieve high-confidence classification. Identified abnormal traffic is reported to the control plane for secondary processing. While 100% accuracy is hard to guarantee, for a small amount of misclassified traffic, we will intervene through an allow list, which can be installed on switches through range match rules.

Actually, we only provide an example of model deployment. Network managers can set different numbers of variables for specific tasks or add more tables to support multi-task classification (e.g., lines $18 \sim 22$ in Listing 1).

Lightweight Model Update Mechanism. To enhance flexibility in model deployment, we introduce a lightweight model update mechanism. Benefiting from the model representation method we use, each switch rule is derived from a classification path. Classification paths with higher priority in Equation (2) will be translated into switch rules with higher priority in match-action tables. Rules with higher priority are given preference for matching PHVs. Our designed priority ensures that the earlier inserted path brings a better accuracy improvement. Thus, optimal model scaling out (or in) can be achieved by adding (or deleting) switch rules with higher (or lower) priority when available resources change. The addition and deletion of paths can be implemented with a modification of *Supplements*, ensuring each enhanced base model still has full classification functionality. During the online phase, it is crucial to ensure timely model updates that respond to dynamic traffic changes. For each enhanced base model, we can update it with another one that has the same number of switch rules. This updating can be accomplished by changing only the rules, eliminating the need for switch restarts.

V. Evaluation

In-Forest-M is evaluated based on the i) lightweight and enhancement of LEGO (§ V-B); ii) superiority and scalability of network-wide distributed deployment (§ V-C); iii) generality of multi-task in-network classification (§ V-D); and iv) low resource consumption with high throughput (§ V-E).

A. Experiment Setup

We build three real network topologies on Linux servers to simulate traffic transmission. Abilene [50] is a small-scale topology with 11 switches and 14 bidirectional links. GEANT [51] is a medium-scale topology with 23 switches and 37 bidirectional links. DialtelecomCz [52] is a large-scale topology with 106 switches and 119 bidirectional links. The high-performance servers are equipped with NVIDIA GeForce RTX 3080 GPUs and Intel Xeon 4210R CPUs. We use the popular ML framework scikit-learn [48] to train tree-based models. For the online traffic-aware model allocation, we implement the PPO algorithm in PyTorch [65]. Besides, model deployment is implemented on two commodity P4 switches, i.e., H3C S9850-32H[2] and OpenMesh BF-48X6Z[3]

We use real-world network traffic from three datasets (i.e., UNSW-NB15 [47], BoT-IoT [66], and CIC-IDS [67]), which have been widely used for evaluating recent in-network classification solutions [13], [36], to construct three network classification tasks with distinct numbers of classes. In-Forest-M is employed to classify specific target classes within these tasks. Table III provides the relevant details. All datasets

[2]https://www.h3c.com/cn/Products___Technology/Products/Switches/Data_Center_Switch/S9800/S9850/

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

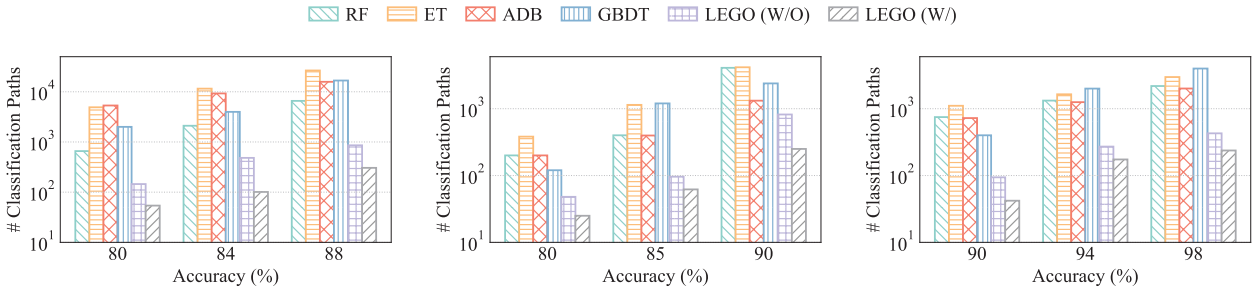


Fig. 8. The comparison in the number of classification paths required to achieve the same accuracy on three network classification tasks.

TABLE III
SETTING OF NETWORK CLASSIFICATION TASKS

Task Dataset	Training (Flows)	Testing (Flows)	# Classes
UNSW-NB15 [47]	500K	125K	10
BoT-IoT [66]	1108K	277K	6
CIC-IDS [67]	1420K	355K	9

contain raw pcap files of network traffic[4], [5], [6]. As suggested in [36] and [64], we set the maximum number of packets stored per flow, i.e., F, to 4 and extract flow-level features for classification. Specifically, we consider the following features: i) Average/minimum/maximum packet lengths of F packets. ii) IP flag (MF and DF) counts of F packets. iii) TCP flag (SYN, ACK, PSH, FIN, RST, and ECN) counts of F packets. iv) Average/minimum/maximum TCP window sizes of F packets. Flows in each task are divided into two parts, including 80% for training and 20% for testing.

B. LEGO Performance

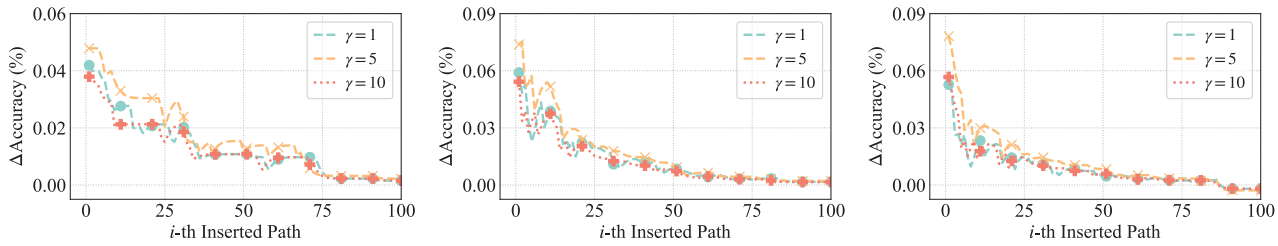
LEGO is the core of In-Forest-M, and we first compare it with four conventional ensemble models: i) Random Forest (RF) [30] is a bagging-based model, while Extremely Randomized Trees (ET) [31] is a variant of RF. ii) Adaboost (ADB) [32] and Gradient Boosting Decision Tree (GBDT) [33] are boosting-based models. Since RF, ET, ADB, and GBDT consist of multiple base models (i.e., DTs), accordingly, we consider DTs as base models and enhance them through our LEGO design module (detailed in § IV-B), i.e., our enhanced base model is a variant of DT. We fairly compare LEGO with these DT-based ensemble models to verify its effectiveness. LEGO is not supported by scikit-learn, so we implement its design logic from scratch, where classification paths are obtained from RF. The number of selected features, i.e., S, is 8 for LEGO, and the same for other models [38].

Fig. 8 compares the number of classification paths required to achieve the same accuracy. Two versions of LEGO are evaluated, i.e., LEGO (W/O), a trimmed version without fine-grained model enhancement, and LEGO (W/), the full version. We control the performance of each model by changing the maximum depth while keeping the other hyperparameters unchanged. Categories of the x-axis are chosen to showcase that LEGO (W/O) and LEGO (W/) require fewer classification paths than other models to achieve varying classification performances. The results demonstrate that LEGO is lightweight and performance-enhanced, achieving comparable classification performance with only part of the paths. For instance, in Fig. 8a, LEGO (W/O) reduces paths by an average of 84.11% (from 3116 to 495) with the same accuracy compared to RF, showcasing the contribution of path-based model splitting and coarse-grained model reorganization in forming lightweight base models. LEGO (W/) enhances base models by inserting valuable paths, which results in a larger reduction of paths by an average of 95.09% (from 3116 to 153). Besides, conventional ensemble models suffer from heavy sizes, hindering their single-point deployment, whereas LEGO is not constrained by this limitation.

To further demonstrate the effectiveness of fine-grained model enhancement, we show the accuracy improvement of LEGO by the i-th inserted path under different enhancement rates γ in Fig. 9. We use LEGO (W/O) with the best classification performance on each of the three tasks and insert paths sequentially. The insertion order is determined by the priority in Equation (2) (detailed in Section § IV-B). ΔAccuracy represents the difference in accuracy between after and before inserting a path. The results show that the earlier inserted path brings a greater improvement and it is cumulative, facilitating flexible model scaling in/out (detailed in § IV-D). As $\gamma = 5$ has the highest ΔAccuracy, we use this setting for all experiments. After inserting 100 paths (only a portion of the total classification paths), under $\gamma = 5$, it leads to an accuracy improvement of 1.64% on UNSW-NB15, 1.59% on BoT-IoT, and 1.21% on CIC-IDS, demonstrating notable enhancement. It is worth noting that the insertion of low-priority paths may lead to accuracy degradation, e.g., ΔAccuracy <0 in Fig. 9c. We attribute this to overfitting. Thus, we determine the maximum number of insertion paths through 5-fold cross-validation [68] during model training.

Moreover, we compare the performance of LEGO and DT. For a fair comparison, we set $K = 1$, i.e., LEGO consists of only one enhanced base model. The value within () represents the maximum number of classification paths. Fig. 10 shows that the enhanced base model outperforms DT under different model resource settings. For instance, LEGO (100) achieves a 7.51% $\uparrow$ higher accuracy than DT (100) on BoT-IoT. In

[4]https://research.unsw.edu.au/projects/unsw-nb15-dataset
[5]https://research.unsw.edu.au/projects/bot-iot-dataset
[6]https://www.unb.ca/cic/datasets/ids-2017.html



(a) UNSW-NB15 (b) BoT-IoT (c) CIC-IDS

Fig. 9. The accuracy improvement of LEGO by the i-th inserted path on three network classification tasks. γ is the enhancement rate.

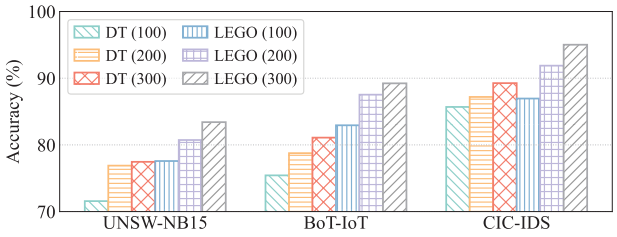


Fig. 10. The comparison of DT and LEGO under different model resource settings. (x) represents the maximum number of classification paths is x.

general, our proposed enhancement mechanism effectively improves the classification performance of base models.

C. Model Distributed Deployment Performance

Next, we consider the network-wide deployment. In-Forest-M is compared with state-of-the-art in-network classification solutions that adopt different model representation methods: i) SwitchTree [17] utilizes direct mapping. ii) Planter [24] and Netbeacon [36] use feature encoding. iii) Mousikav2 [27] employs model quantization. Since existing solutions are centralized and lack a network-wide deployment mechanism, we extend them by introducing four baselines, i.e., RF-100%, RF-70%, RF-20%, and DT-100%:

- RF-100%. Deploy RF on all switches.
- RF-70/20%. Deploy RF randomly on 70/20% switches.
- DT-100%. Deploy DT on all switches.

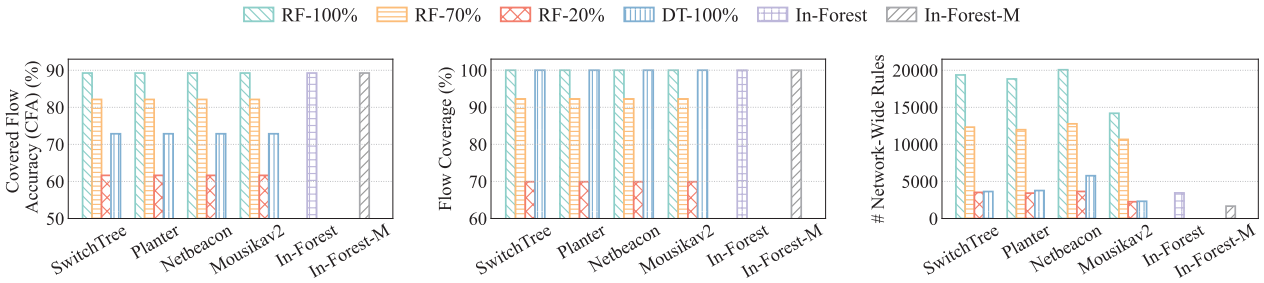
RF is selected as the ensemble model deployed by existing solutions because it is widely supported by each of them [28]. We also consider the deployment of DT to verify whether the model for distributed deployment can outperform the individual model through ensemble learning. For each baseline, the complete RF or DT is deployed on single switches by existing solutions. The maximum depth of models is set to 10, as deeper trees are impractical for in-network deployment [21]. RF is configured with only 3 sub-models due to the single-point memory limitation [28]. In-Forest and In-Forest-M employ offline model allocation to determine the allocation scheme of enhanced base models. We use the grid search to find the optimal weights in Equation (3), i.e., $\alpha_1 = 0.6$, $\alpha_2 = 0.2$, and $\alpha_3 = 0.2$. We use UNSW-NB15 as the flow dataset and Abilene as the network topology connected to 8 subnets. For assigning each flow to a subnet pair, we use a hash function based on the 5-tuple. The routing path is determined by the OSPF [49] protocol.

Fig. 11 compares the number of network-wide switch rules required to achieve equal CFA (Equation (7)) and flow coverage (percentage of flows with models deployed on the routing path). The number is calculated by summing the number of rules required on different switches. Compared with RF-100%, In-Forest-M achieves the same CFA and flow coverage with reduced switch rules. In Fig. 11c, even only with the partial deployment of RF, i.e., RF-70% and RF-20%, the rule number of existing solutions remains higher than In-Forest-M. Notably, RF-70% and RF-20% do not guarantee flow coverage (Fig. 11b), which requires traffic rescheduling, leading to complex network routing and management. Compared with DT-100%, In-Forest-M achieves a 16.39% higher CFA on average. Since the models are identical, flows do not obtain any accuracy improvement after traversing multiple DTs. In contrast, In-Forest-M aggregates the classification results of enhanced base models to improve accuracy.

Fig. 12 performs a comparison under different resource scenarios by limiting the maximum number of network-wide rules. In-Forest-M achieves flexible adjustment of the model allocation scheme by changing M and E (detailed in § IV-C). Simultaneously, the model depth is adjusted and the optimal model scaling in/out is enabled (detailed in § IV-D). Larger M and E indicate that the entire network has more available resources for model deployment, resulting in a higher CFA. The other four baselines do not have a similar mechanism, which adapts by only adjusting the model depth. The results show that In-Forest-M has a superior CFA. For instance, compared with RF-100% in Fig. 12b, In-Forest-M improves accuracy by 19.31% (from 62.21% to 81.52%) while reducing the rule number by 94.96% (from 1727 to 87).

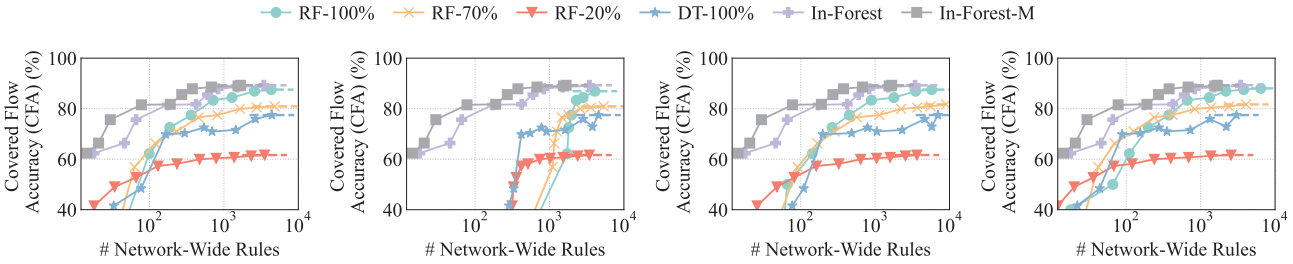
To present the scalability of In-Forest-M, we evaluate the network-wide deployment performance in larger topologies. GEANT is connected to 15 subnets and DialtelecomCz is connected to 50 subnets. As shown in Fig. 13 and Fig. 14, In-Forest-M still achieves the highest CFA. The advantages of distributed deployment become even more apparent as the topology scales up, due to the increasing resource waste of centralized in-network classification solutions.

Furthermore, we demonstrate the traffic awareness of In-Forest-M in topologies with different H settings. H depends on the topology scale, ensuring the diversity of enhanced base models across switches. Specifically, H is 6 in Abilene, 8 in GEANT, and 15 in DialtelecomCz. We set the flow duration, which is 10 timesteps in Abilene, 15 timesteps in GEANT, and 50 timesteps in DialtelecomCz [50], allowing the traffic



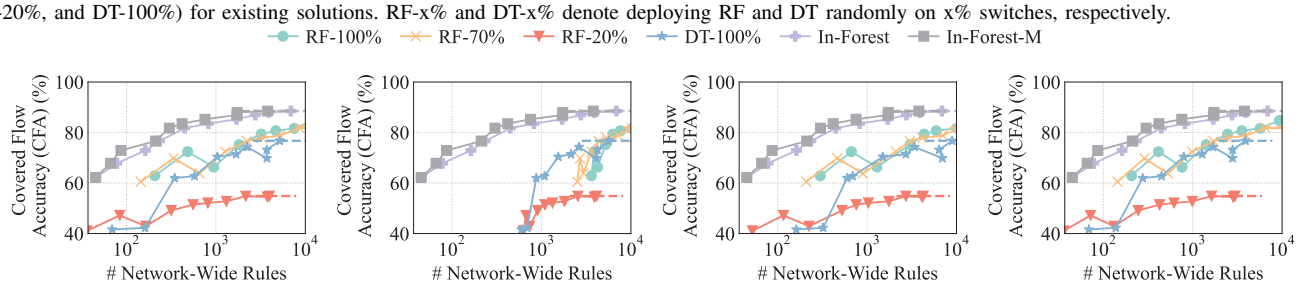
(a) Covered Flow Accuracy (CFA) (b) FLow Coverage (c) Number of Network-Wide Rules

Fig. 11. The network-wide deployment performance in Abilene. We design four baselines (RF-100%, RF-70%, RF-20%, and DT-100%) for existing solutions. RF-x% and DT-x% denote deploying RF and DT randomly on x% switches, respectively.



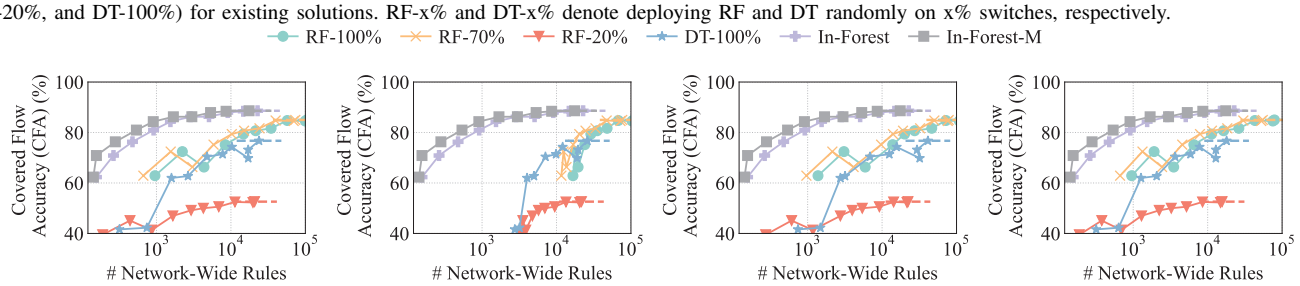
(a) SwitchTree (b) Planter (c) Netbeacon (d) Mousikav2

Fig. 12. The Covered Flow Accuracy (CFA) under different numbers of network-wide rules in Abilene. We design four baselines (RF-100%, RF-70%, RF-20%, and DT-100%) for existing solutions. RF-x% and DT-x% denote deploying RF and DT randomly on x% switches, respectively.



(a) SwitchTree (b) Planter (c) Netbeacon (d) Mousikav2

Fig. 13. The Covered Flow Accuracy (CFA) under different numbers of network-wide rules in GEANT. We design four baselines (RF-100%, RF-70%, RF-20%, and DT-100%) for existing solutions. RF-x% and DT-x% denote deploying RF and DT randomly on x% switches, respectively.



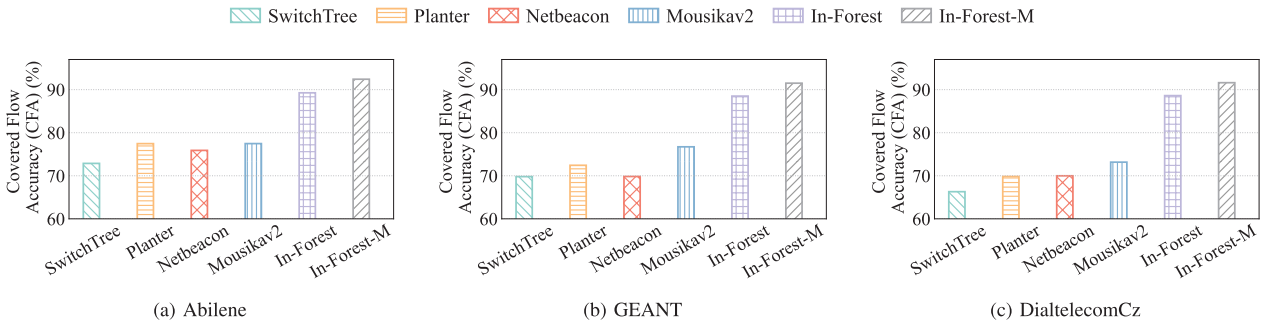
(a) SwitchTree (b) Planter (c) Netbeacon (d) Mousikav2

Fig. 14. The Covered Flow Accuracy (CFA) under different numbers of network-wide rules in DialtelecomCz. We design four baselines (RF-100%, RF-70%, RF-20%, and DT-100%) for existing solutions. RF-x% and DT-x% denote deploying RF and DT randomly on x% switches, respectively.

to change over time. To simulate diverse changes, we add a random number $\sigma \in [0, 3]$ to the duration of each flow. In addition, we set three random seeds (i.e., 0, 1, and 2) for σ and average the obtained results, avoiding evaluation bias. We choose the model allocation scheme in the offline phase as the initial value of PPO and employ the online phase to detect dynamic traffic changes. Table IV shows the average CFA improvement, i.e., $1.15\%\uparrow$ in Abilene, $1.57\%\uparrow$ in GEANT, and $1.52\% \uparrow$ in DialtelecomCz, respectively. The results demonstrate the effectiveness of traffic awareness, which brings a better CFA compared with the offline phase.

D. Multi-Task Classification Performance

For In-Forest-M, we evaluate the generality of multi-task in-network classification. The flows from three network classification tasks are mixed to simulate real network demands. We conduct experiments in diverse network topologies where mixed flows are assigned to subnet pairs. In-Forest-M uses the

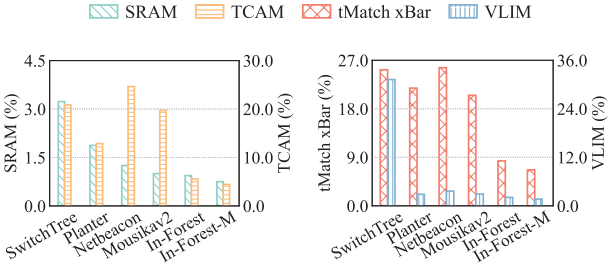


(a) Abilene (b) GEANT (c) DialtelecomCz

Fig. 15. The multi-task classification performance in diverse topologies. The flows from three network classification tasks are mixed to simulate real network demands. SwitchTree, Planter, Netbeacon, Mousikav2, and In-Forest employ the optimal single-task allocation schemes while randomly deploying models on switches for three tasks. In-Forest-M assigns task-specific enhanced base models to different switches to enable multi-task classification.

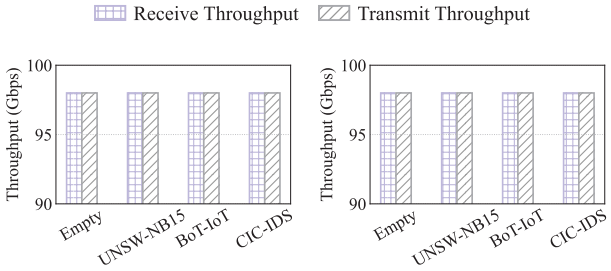
TABLE IV
AVERAGE CFA IMPROVEMENT OF THE ONLINE PHASE

Network Topology	H	Average CFA (%)	
		Offline	Offline + Online
Abilene [50]	6	89.26%	90.41% (1.15%↑)
GEANT [51]	8	88.50%	90.07% (1.57%↑)
DialtelecomCz [52]	15	88.61%	90.13% (1.52%↑)



(a) Memory Resources (b) Computational Resources

Fig. 16. The resource consumption of different solutions on the H3C switch.



(a) H3C S9850-32H (b) OpenMesh BF-48X6Z

Fig. 17. The switch throughput on three network classification tasks.

two-phase resource-aware model allocation strategy to assign different task-specific enhanced base models to switches, enabling them to handle different tasks simultaneously. In contrast, SwitchTree, Planter, Netbeacon, Mousikav2, and In-Forest are specifically designed for a single task. Consequently, they employ the optimal single-task allocation schemes in Fig. 12, Fig. 13, and Fig. 14, respectively, while randomly deploying models on switches for three tasks.

Fig. 15 demonstrates that In-Forest-M outperforms other solutions. Existing centralized in-network classification solutions require the deployment of complete models, which makes it challenging to deploy multiple high-accuracy models on a single-point switch for different tasks, resulting in compromised network-wide classification performance. Moreover, compared with In-Forest, In-Forest-M effectively utilizes available resources to identify a better multi-task deployment scheme for the entire network. For instance, in Abilene, In-Forest-M has a 3.16% higher CFA than In-Forest.

E. Hardware Performance

Fig. 16 shows the single-point resource consumption on the H3C S9850-32H switch. SwitchTree, Planter, Netbeacon, and Mousikav2 deploy RF, while In-Forest and In-Forest-M deploy the enhanced base model. The models are all from Fig. 11, which achieve the same CFA and flow coverage. We compare memory resources, i.e., the percentage of used SRAM and TCAM, and computational resources, i.e., the percentage of used tMatch xBar and VLIW. SRAM is used to store exact match rules, while TCAM is used to store ternary match, longest prefix match, and range match rules. tMatch xBar is used to perform ternary match and range match, while VLIM is used for actions [26]. The results illustrate that In-Forest-M consumes the lowest switch resources. For instance, the SRAM usage for SwitchTree and In-Forest-M is 3.23% and 0.75% in Fig. 16a, while the tMatch xBar usage for Netbeacon and In-Forest-M is 25.63% and 6.67% in Fig. 16b.

For the three network classification tasks, we deploy the optimal enhanced base model on two commodity P4 switches. Fig. 17 shows switch throughput under high-speed traffic of 100Gbps. As shown, after loading the P4 program and installing rules from the enhanced base model, the switch throughput remains virtually unchanged.

F. In-Forest-M Deep Dive

We further explore various aspects of In-Forest-M to demonstrate its performance in different network scenarios.

Reconfiguration Process. In-Forest-M employs a two-phase resource-aware model allocation strategy: The offline phase provides topology-aware model allocation schemes under different resource constraints, serving as solid bases, followed by an online traffic-aware phase that fine-tunes schemes to adapt to dynamic traffic. Although reconfiguration and deployment (e.g., offline topology-aware allocation) are inevitable when network topology changes, such changes are generally much less frequent in real-world applications

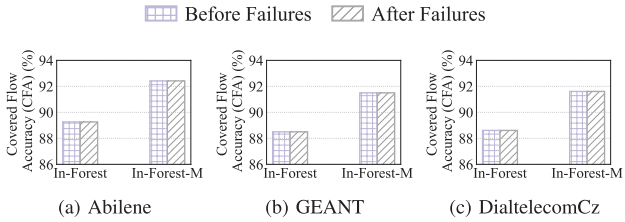


Fig. 18. The multi-task classification performance in diverse topologies before and after network failures (i.e., 10% of switches fail randomly).

than traffic fluctuations. Therefore, compared with the online phase, the offline phase has lower requirements for computing latency. In other words, the reconfiguration process and the distributed deployment caused by topology changes will not significantly affect the overall framework's performance. To demonstrate this, we evaluate the time overhead due to network topology changes, which is 5.03s from Abilene to GEANT and 8.47s from GEANT to DialtelecomCz. All experimental settings align with those in Fig. 15. Switching between larger topologies takes more time but can still be completed quickly. We also validate the cost of changing some links in DialtelecomCz, i.e., randomly modifying 10% of 119 bidirectional links. In-Forest-M adapts to such minor changes based on existing allocation schemes, requiring less time than a complete topology alteration. Statistical results show that reconfiguration and deployment are fully updated within 1.15s.

Error Handling Mechanism. Distributed deployment splits the ensemble model into enhanced base models across multiple switches, avoiding single-point failures that may cause the entire system to crash. In response to dynamic traffic changes and network failures, we apply traffic-aware model allocation and a lightweight model update mechanism to mitigate the degradation caused by traffic distribution deviation, which can maintain classification accuracy. In addition, we also implement corresponding processing solutions for abnormal traffic. By aggregating the results of enhanced base models from multiple switches, we achieve high-confidence classification, and the identified abnormal traffic is reported to the control plane for secondary processing. While 100% accuracy is hard to guarantee, for a small number of misclassified traffic, we will intervene through an allow list to further optimize. We refer to all the settings in Fig. 15 and simulate realistic network conditions like network failures and abnormal traffic. As shown in Table III and Fig. 15, flows from three network classification tasks are mixed to simulate real network demands, including abnormal traffic. We configure three network topologies with 10% of switches randomly failing and evaluate multi-task classification performance post-reconfiguration. Fig. 18 shows that through the reconfiguration process and error handling mechanism, the multi-task classification performance before and after the failure can remain consistent, which demonstrates the robustness of our framework in real-world applications.

VI. Discussion and Future Work

In-Forest-M exhibits promising potential for distributed in-network classification with powerful performance. Although we focus on deploying it on programmable switches in this paper, our framework can also be deployed on standard network hardware with only minor adjustments. Specifically, we still adopt a similar distributed deployment solution, splitting the ensemble model into various enhanced base models and deploying them as table entries on different switches according to offline topology-aware allocation. We then modify the model update mechanism, e.g., online traffic-aware allocation. Our distributed framework allows the classification performance to be optimized by replacing only some switches, i.e., core switches that occupy key positions in network topologies, to match dynamic traffic. We periodically replace table entries on certain switches to achieve performance improvements with as little overhead as possible. In our future work, we will explore the application of In-Forest-M in more network hardware, such as Smart Network Interface Cards (SmartNICs).

Besides, we aim to explore more fine-grained distributed deployment strategies, such as deployment based on subsets of switch rules, and plan to leverage heterogeneous resources to further enhance the applicability of In-Forest-M. Current experimental results fully validate the feasibility of In-Forest-M in prototype deployment. We are developing a large-scale testbed using commodity P4 switches based on real network topologies and will further explore In-Forest-M's potential in our future work.

VII. Conclusion

In this paper, we propose In-Forest-M, a general distributed multi-task in-network classification framework, achieving high-accuracy network traffic classification on programmable switches. Firstly, we design a Lightweight Ensemble Generic Optional Model (LEGO), which can be transformed into multiple enhanced base models, each enabling line-speed classification on a single switch. Secondly, we propose a two-phase resource-aware model allocation strategy to assign different task-specific enhanced base models to switches for multi-task classification. In addition, we devise a lightweight model update mechanism to ensure flexibility in adapting to different resource constraints and dynamic traffic changes. Comprehensive experiments reveal that, compared with state-of-the-art centralized solutions, In-Forest-M outperforms in terms of accuracy and hardware resource consumption under network-wide deployment while presenting great generality in multi-task classification.

References

[1] A. Nascita, A. Montieri, G. Aceto, D. Ciuonzo, V. Persico, and A. Pescapé, "XAI meets mobile traffic classification: Understanding and improving multimodal deep learning architectures," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 4, pp. 4225–4246, Dec. 2021.

[2] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, "FS-Net: A flow sequence network for encrypted traffic classification," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2019, pp. 1171–1179.

[3] Z. Wu, Y.-N. Dong, X. Qiu, and J. Jin, "Online multimedia traffic classification from the QoS perspective using deep learning," *Comput. Netw.*, vol. 204, Feb. 2022, Art. no. 108716.

[4] O. Aouedi, K. Piamrat, and D. Bagadthey, "A semi-supervised stacked autoencoder approach for network traffic classification," in *Proc. IEEE 28th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2020, pp. 1–6.

[5] P. Poupart et al., "Online flow size prediction for improved network routing," in *Proc. IEEE 24th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2016, pp. 1–6.

[6] T. Panayiotou, M. Michalopoulou, and G. Ellinas, "Survey on machine learning for traffic-driven service provisioning in optical networks," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 2, pp. 1412–1443, 2nd Quart., 2023.

[7] Y. Yan, F. Li, W. Wang, and X. Wang, "TalentSketch: LSTM-based sketch for adaptive and high-precision network measurement," in *Proc. IEEE 30th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2022, pp. 1–12.

[8] X. Zhang, L. Cui, F. P. Tso, and W. Jia, "PHeavy: Predicting heavy flows in the programmable data plane," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 4, pp. 4353–4364, Dec. 2021.

[9] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–15.

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

[11] Y. Dong, Q. Li, R. O. Sinnott, Y. Jiang, and S. Xia, "ISP self-operated BGP anomaly detection based on weakly supervised learning," in *Proc. IEEE 29th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2021, pp. 1–11.

[12] T. Zheng and B. Li, "Poisoning attacks on deep learning based wireless traffic prediction," in *Proc. IEEE Conf. Comput. Commun.*, May 2022, pp. 660–669.

[13] C. Zheng, X. Hong, D. Ding, S. Vargaftik, Y. Ben-Itzhak, andN. Zilberman, "In-network machine learning using programmable network devices: A survey," *IEEE Commun. Surveys Tuts.*, vol. 26, no. 2, pp. 1–35, 2nd Quart., 2024.

[14] C. Zheng, B. Rienecker, and N. Zilberman, "QCMP: Load balancing via in-network reinforcement learning," in *Proc. 2nd ACM SIGCOMM Workshop Future Internet Routing Addressing*, Sep. 2023, pp. 35–40.

[15] R. Parizotto, B. L. Coelho, D. C. Nunes, I. Haque, and A. Schaeffer-Filho, "Offloading machine learning to programmable data planes: A systematic survey," *ACM Comput. Surv.*, vol. 56, no. 1, pp. 1–33, Jan. 2024.

[16] X. Lin, G. Xiong, G. Gou, Z. Li, J. Shi, and J. Yu, "ET-BERT: A contextualized datagram representation with pre-training transformers for encrypted traffic classification," in *Proc. ACM WWW*, Apr. 2022, pp. 633–642.

[17] J.-H. Lee and K. Singh, "SwitchTree: In-network computing and traffic analyses with random forests," *Neural Comput. Appl.*, 2020.

[18] M. Zhang et al., "Poseidon: Mitigating volumetric DDoS attacks with programmable switches," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–18.

[19] J. Xing, Q. Kang, and A. Chen, "NetWarden: Mitigating network covert channels while preserving performance," in *Proc. USENIX Secur.*, Jan. 2020, pp. 2039–2056.

[20] Barefoot Netw. *Tofino Switch*. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series/tofino.html

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

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

[23] C. Zheng et al., "IIsy: Practical in-network classification," 2022, *arXiv:2205.08243*.

[24] C. Zheng and N. Zilberman, "Planter: Seeding trees within switches," in *Proc. SIGCOMM Poster*, Aug. 2021, pp. 12–14.

[25] C. Zheng et al., "Automating in-network machine learning," 2022, *arXiv:2205.08824*.

[26] 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 *Proc. IEEE Conf. Comput. Commun.*, May 2022, pp. 1938–1947.

[27] G. Xie et al., "Empowering in-network classification in programmable switches by binary decision tree and knowledge distillation," *IEEE/ACM Trans. Netw.*, vol. 32, no. 1, pp. 382–395, Jan. 2023.

[28] A. T.-J. Akem, B. Bütün, M. Gucciardo, and M. Fiore, "Henna: Hierarchical machine learning inference in programmable switches," in *Proc. 1st Int. Workshop Native Netw. Intell.*, Dec. 2022, pp. 1–7.

[29] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Boca Raton, FL, USA: CRC Press, 2017.

[30] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, Oct. 2001.

[31] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Mach. Learn.*, vol. 63, no. 1, pp. 3–42, Apr. 2006.

[32] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997.

[33] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001.

[34] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 785–794.

[35] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven streaming network telemetry," in *Proc. ACM SIGCOMM*, Aug. 2018, pp. 357–371.

[36] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *Proc. 32nd USENIX Sec. Symp.*, 2023, pp. 1–18.

[37] 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 *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 15–28.

[38] J. Lin et al., "In-forest: Distributed in-network classification with ensemble models," in *Proc. IEEE 31st Int. Conf. Netw. Protocols (ICNP)*, Oct. 2023, pp. 1–12.

[39] G. Xie et al., "Efficient flow recording with InheritSketch on programmable switches," in *Proc. IEEE 43rd Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2023, pp. 1–11.

[40] B. A. A. Nunes, M. S. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," in *Proc. IEEE Commun. Surveys & Tuts.*, vol. 16, Jan. 2014, pp. 1617–1634.

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

[42] N. McKeown, "Pisa: Protocol independent switch architecture," in *Proc. P4 Workshop*, vol. 516, 2015.

[43] Y. Li, J. Sun, W. Huang, and X. Tian, "Detecting anomaly in large-scale network using mobile crowdsourcing," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2019, pp. 2179–2187.

[44] 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 *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–18.

[45] S. Wang et al., "Martini: Bridging the gap between network measurement and control using switching ASICs," in *Proc. IEEE 28th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2020, pp. 1–12.

[46] X. Jia, F. Li, S. Chen, C. Gao, P. Wang, and X. Wang, "RED: Distributed program deployment for resource-aware programmable switches," in *Proc. IEEE Conf. Comput. Commun.*, May 2023, pp. 1–10.

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

[48] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011.

[49] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS weights in a changing world," *IEEE J. Sel. Areas Commun.*, vol. 20, no. 4, pp. 756–767, May 2002.

[50] C. Liu, M. Xu, Y. Yang, and N. Geng, "DRL-OR: Deep reinforcement learning-based online routing for multi-type service requirements," in *Proc. IEEE Conf. Comput. Commun.*, May 2021, pp. 1–10.

[51] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 83–86, Jan. 2006.

[52] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

[53] F. J. Ferri, P. Pudil, M. Hatef, and J. Kittler, "Comparative study of techniques for large-scale feature selection," in *Machine Intelligence and Pattern Recognition*, vol. 16. Amsterdam, The Netherlands: North Holland,, 1994, pp. 403–413.

[54] B. H. Menze et al., "A comparison of random forest and its Gini importance with standard chemometric methods for the feature selection and classification of spectral data," *BMC Bioinf.*, vol. 10, no. 1, pp. 1–16, Dec. 2009.

[55] H. Siddique, M. Neves, C. Kuzniar, and I. Haque, "Towards network-accelerated ML-based distributed computer vision systems," in *Proc. IEEE 27th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2021, pp. 122–129.

[56] C. E. Ferreira, A. Martin, and R. Weismantel, "Solving multiple knapsack problems by cutting planes," *SIAM J. Optim.*, vol. 6, no. 3, pp. 858–877, Aug. 1996.
[57] C. Bodnar, B. Day, and P. Lió, "Proximal distilled evolutionary reinforcement learning," in *Proc. AAAI Conf. Artif. Intell. (AAAI)*, vol. 34, no. 4, Feb. 2020, pp. 3283–3290.
[58] S. Khadka and K. Tumer, "Evolutionary reinforcement learning," 2018, *arXiv:1805.07917*.
[59] E. Marchesini and A. Farinelli, "Genetic deep reinforcement learning for mapless navigation," in *Proc. AAMAS*, Cham, Switzerland. Springer, May 2020, pp. 1919–1921.
[60] Y. Song, L. Wei, Q. Yang, J. Wu, L. Xing, and Y. Chen, "RL-GA: A reinforcement learning-based genetic algorithm for electromagnetic detection satellite scheduling problem," *Swarm Evol. Comput.*, vol. 77, Mar. 2023, Art. no. 101236.
[61] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
[62] B. Khurshid, S. Maqsood, Y. Khurshid, K. Naeem, and Q. S. Khalid, "A hybridization of evolution strategies with iterated greedy algorithm for no-wait flow shop scheduling problems," *Sci. Rep.*, vol. 14, no. 1, p. 2376, Jan. 2024.
[63] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015, *arXiv:1506.02438*.
[64] G. Xie, Q. Li, H. Yan, D. Zhao, G. Antichi, and Y. Jiang, "Efficient attack detection with multi-latency neural models on heterogeneous network devices," in *Proc. IEEE 31st Int. Conf. Netw. Protocols (ICNP)*, Oct. 2023, pp. 1–12.
[65] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. NIPS*, 2019, pp. 1–12.
[66] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset," *Future Gener. Comput. Syst.*, vol. 100, pp. 779–796, Nov. 2019.
[67] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy (ICISSP)*, 2018, pp. 108–116.
[68] M. W. Browne, "Cross-validation methods," *J. Math. Psychol.*, vol. 44, no. 1, pp. 108–132, Mar. 2000.

Qing Li (Senior Member, IEEE) received the B.S. degree in computer science and technology from Dalian University of Technology, Dalian, China, in 2008, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2013. He is currently an Associate Researcher at the Pengcheng Laboratory, Shenzhen, China. His research interests include reliable and scalable routing of the internet, network function virtualization, and in-network caching/computing.

Jiaye Lin received the B.S. degree from Sun Yat-sen University, Guangzhou, China, in 2022. He is currently pursuing the M.S. degree with the International Graduate School, Tsinghua University, Shenzhen, China. His research interests include in-network classification, P4 program development, and reinforcement learning.

Guorui Xie received the B.S. degree from Sun Yat-sen University, Guangzhou, China, in 2019. He is currently pursuing the Ph.D. degree in computer science with the International Graduate School, Tsinghua University, Shenzhen, China. His research interests include in-network intelligence, programmable switch development, and network classification tasks based on machine learning.

Zhongxu Guan received the B.S. degree from Harbin Institute of Technology, Harbin, China, in 2021. He is currently pursuing the M.S. degree with the International Graduate School, Tsinghua University, Shenzhen, China. His research interests include in-network intelligence and interpretable machine learning.

Zeyu Luan received the B.S. and M.S. degrees from Tianjin University, Tianjin, China, in 2015 and 2018, respectively. He is currently pursuing the Ph.D. degree in computer science with the International Graduate School, Tsinghua University, Shenzhen, China. His research interests include software defined networking, traffic engineering, and reinforcement learning.

Zhuyun Qi received the B.S. degree from Beijing Technology and Business University, Beijing, China, in 2006, and the M.S. degree from China Academy of Launch Vehicle Technology, Beijing, in 2009. She is currently a Senior Engineer at the Pengcheng Laboratory, Shenzhen, China. Her research interests include software defined networking, named data networking, and blockchain.

Yong Jiang (Member, IEEE) received the B.S. and Ph.D. degrees in computer science and technology from Tsinghua University, Beijing, China, in 1998 and 2002, respectively. He is currently a Full Professor at the International Graduate School, Tsinghua University, Shenzhen, China. His research interests include future network architecture, software defined networking, and network function virtualization.

Zhenhui Yuan (Member, IEEE) received the B.S. degree from Wuhan University, Wuhan, China, in 2008, and the Ph.D. degree from Dublin City University, Dublin, Ireland, in 2012. He is currently a Senior Lecturer at Northumbria University, Newcastle upon Tyne, U.K. His research interests include software defined networking and wireless communication systems for robots and vehicles.