In-Forest: Distributed In-Network Classification with Ensemble Models

Jiaye Lin[1,2], Qing Li[2], Guorui Xie[1,2], Yong Jiang[1,2], Zhenhui Yuan[3], Changlin Jiang[2], Yuan Yang[4]
[1]International Graduate School, Tsinghua University, Shenzhen, China
[2]Peng Cheng Laboratory, Shenzhen, China
[3]Department of Computer and Information Science, Northumbria University, Newcastle, United Kingdom
[4]Department of Computer Science and Technology, Tsinghua University, Beijing, China
Corresponding Author: Qing Li (liq@pcl.ac.cn)

2023 IEEE 31st International Conference on Network Protocols (ICNP) | 979-8-3503-0322-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICNP59255.2023.10355602

***Abstract*—A variety of model representation methods have been used in recent works to translate machine learning models into programmable switch rules to address network classification tasks at line-speed, i.e., in-network classification. These works generally deploy a complete but heavy model on a switch with limited hardware resources, causing both network-wide waste of resources and unsatisfactory accuracy. Therefore, we propose In-Forest, a general distributed in-network classification framework. Firstly, to improve accuracy with limited resources, we develop a Lightweight Ensemble Generic Optional Model (LEGO), which can be further enhanced into multiple enhanced base models with full functionality. Each switch only needs to deploy a simple base model, rather than the complete ensemble model. Thus, hardware resources required for both switches and the entire network can be significantly reduced. Secondly, as traffic traverses multiple switches, In-Forest aggregates the classification results from different enhanced base models for higher accuracy. Furthermore, we design a two-phase resource-aware model allocation strategy that assigns enhanced base models to switches under different scenarios. We use stable deep reinforcement learning to respond to dynamic traffic changes. Experimental results show that when compared to SwitchTree, Planter, and Netbeacon in two real network topologies, In-Forest can increase accuracy by up to 19.31%, while reducing the number of switch rules by 89.98%.**

***Index Terms*—distributed deployment, in-network classification, programmable data plane, deep reinforcement learning**

I. Introduction

Recently, network classification based on Machine Learning (ML) becomes widespread to support various needs, e.g., flow size prediction [1–4], traffic classification [5–8], and anomaly detection [9–12]. They simplify network management, boost network Quality of Service (QoS), or assure network security.

Typically, conventional network classification solutions off-path the traffic to remote GPU servers and use complex learning models, like Long Short Term Memory (LSTM) or Stacked AutoEncoder, to get better classification performance [3, 8, 13]. However, they have two problems: a) High latency. They need to transmit traffic to remote servers for analysis, introducing additional round-trip latency [14]. b) High cost. Powerful GPU servers are more expensive but with limited packet processing capacity compared to network forwarding devices (e.g., switches), especially when facing massive traffic of 100Gbps or even Tbps [15–17].

Some solutions deploy ML models on the Programmable Data Plane (PDP) to address network classification tasks at line-speed [14, 18–22], i.e., in-network classification. They translate models into rules of programmable switches (e.g., P4 switches [23]) to support on-path traffic processing. As a rule-based paradigm, tree-based models [24–26] are compatible with PDP's match-action architecture [27]. The existing solutions mainly use two model representation methods to deploy tree-based models, i.e., direct mapping and feature encoding. pForest [18] and SwitchTree [14] deploy the tree-based models by directly mapping every layer of the Decision Tree (DT) to a stage of the programmable switch. However, the number of stages (e.g., 12 for Tofino 1 [28]) limits the model depth, resulting in poor scalability and restricted accuracy. IIsy [19], Planter [20], and Netbeacon [22] use feature tables to encode features and a model table for decision-making. Although a stage can store multiple tables, the feature number must be limited due to the limitation in memory [29]. As model size increases, the surge in switch rules worsens scalability.

Above all, existing in-network classification solutions [14, 18–22] are centralized, focusing on deploying the complete but heavy model on one single switch, which causes two problems: a) Low accuracy. Due to the limited resources of one single switch, it is difficult to support the deployment for large-scale models that have high accuracy [29]. b) Overuse of resources. For a network-wide model deployment to cover all the traffic, the same model is deployed multiple times on different switches. It wastes a significant amount of network device resources to process traffic that has been processed by the first switch without offering any extra benefit.

In our view, it is more efficient to distribute the complete model into multiple switches and process the traffic by sub-models cooperatively. Distributed deployment enhances classification performance and reduces hardware resource consumption (detailed in Section III-A2). There are two potential ways to achieve this: a) We can split the model into layers, with each switch responsible for deploying one or more layers. Due to the fact that a tree-based model can only get the final results at the leaf nodes, a single sub-model cannot provide full functionality. b) We can also split the model after translating it into switch rules, and then assign rules to

different switches. However, each sub-model can only process part of the traffic and possesses limited functionality. The two ways both need to reschedule the traffic to pass through all the sub-models for full function processing. This introduces significant complexity in network routing and management. Our goal is to design a distributed framework that satisfies the following requirements: each lightweight sub-model provides full functionality, and the more sub-models the traffic passes through, the better classification performance we can get. In this way, we can achieve high accuracy without single-point resource limitation, network-wide redundant resource consumption, and traffic rescheduling.

Therefore, we propose In-Forest, a general distributed in-network classification framework that utilizes the available resources of multiple switches to deploy the large-scale model. We design a Lightweight Ensemble Generic Optional Model (LEGO) that can be transformed into multiple enhanced base models with full functionality and different classification knowledge. Through our designed model allocation strategy and model update mechanism, In-Forest enables flexible model distributed deployment under different resource scenarios and dynamic traffic changes. As traffic passes through switches with different enhanced base models, In-Forest uses ensemble learning to aggregate classification results, getting more comprehensive knowledge to correct the errors of individual models and achieve improved performance. In particular:

- We design a LEGO model, which consists of many simple but cooperative enhanced base models. Each enhanced base model is fully functional and can be deployed on a single switch. In-Forest aggregates the classification results from different enhanced base models by ensemble learning (e.g., majority voting) to get higher accuracy without single-point resource limitation.
- We propose a two-phase resource-aware model allocation strategy to determine the optimal allocation scheme of enhanced base models. Firstly, the offline topology-aware allocation selects models for switches, aiming to maximize model diversity across all paths within limited optional models and switch resources. Secondly, the online traffic-aware allocation based on Deep Reinforcement Learning (DRL) responds to traffic changes by tuning the deployed models to maximize accuracy. A stable learning mechanism is employed to ensure effectiveness.
- We devise a lightweight model update mechanism that ensures flexibility in adapting to different resource scenarios and dynamic traffic changes. The enhanced base model rules are assigned corresponding priorities for optimal model scaling in/out when the available resources change. In addition, the model can be updated to another one by changing only the rules when traffic changes.

In-Forest is evaluated by comparing three advanced in-network classification solutions, i.e., SwitchTree [14], Planter [20], and Netbeacon [22]. Our findings show that, in two real network topologies, In-Forest can increase accuracy by up to 19.31%, while reducing the number of switch rules by 89.98%.

II. Background and Related Work

A. Background

Programmable Data Plane. The rise of the Programmable Data Plane (PDP) improves network programmability. The PDP allows network managers to implement customized data plane algorithms on programmable forwarding devices (e.g., P4 switches [23]), catering to a variety of network applications [14, 20, 22, 30]. This makes in-network classification for line-speed traffic analysis a reality. However, the PDP has operational limits, supporting only simple operations such as shift, add, and boolean, while lacking support for multiplication, loop, or floating operations [27]. These limitations make it difficult to deploy complex models. Fortunately, tree-based models are well-suited for programmable switches as they are rule-based learning classifiers without many hard-to-implement operations. But each switch has limited resources (e.g., stage number, memory) [29], posing a challenge in deploying large-scale models to achieve high accuracy.

Ensemble Model. Ensemble models encompass a range of methods, e.g., bagging and boosting. Bagging methods, such as Random Forest (RF) [25], generate an ensemble by independently training sub-models on different subsets of the dataset and combining their results. The performance of each sub-model may be relatively weak, but by majority voting, RF aggregates their results to obtain comprehensive classification knowledge, thus correcting misclassified samples, preventing overfitting, and achieving higher accuracy. Boosting methods, such as Adaboost (ADB) [31] and Gradient Boosting Decision Tree (GBDT) [32], successively train sub-models to generate an ensemble. Each sub-model improves classification accuracy on samples misclassified by the previous model. Ensemble models outperform individual models in network classification tasks [33, 34], but their complexity and large resource consumption hinder in-network deployment.

B. Existing Works

Deploying tree-based ML models on the PDP to achieve in-network classification is a promising technology. Table I provides a partial snapshot of the existing solutions.

NetWarden [17] and FlowLens [16] collect traffic information on the data plane and perform analysis on the control plane. Due to the communication latency [35], traffic analysis cannot be achieved at line-speed (not LS). Some solutions embed tree-based models into PDP's match-action tables by different representation methods. pForest [18] and SwitchTree [14] employ the direct mapping method, where every layer of the DT is deployed in a switch stage. But the model depth is limited by the stage number, making it difficult to accommodate larger and more accurate models. Another method used by IIsy [19], Planter [20], and Netbeacon [22] is feature encoding, which involves feature tables to encode features and a model table for decision-making. This method requires the dataset to have a limited number of features, otherwise, tables take up too much memory. Furthermore, both methods face a substantial increase in table rules with the larger model,

TABLE I
COMPARISON OF ADVANCED IN-NETWORK CLASSIFICATION SOLUTIONS

Work	• LS	• AR	• TA	• AN
NetWarden [17]	✘	✔	✔	✘
FlowLens [16]	✘	✔	✔	✘
pForest [18]	✔	✘	✔	✘
SwitchTree [14]	✔	✘	✔	✘
IIsy [19]	✔	✘	✘	✘
Planter [20]	✔	✘	✘	✘
Netbeacon [22]	✔	✘	✔	✘
Mousika [21]	✔	✔	✘	✘
In-Forest	✔	✔	✔	✔

• LS: Line-Speed Processing, AR: Accuracy Is Not Restrict by Hardware Resources, TA: Traffic Awareness, AN: Assess Network-Wide Performance

resulting in accuracy being restricted by hardware resources (not AR). Mousika [21] applies the distillation method to transfer knowledge from complex teacher models to Binary DT (BDT). But it lacks support for different network resource scenarios and dynamic traffic changes (not TA).

Existing solutions deploy the complete model on a single switch with limited resources. Extending these solutions to enable network-wide deployment causes significant resource waste (not AN). This is because the same model must be deployed on different switches to cover all the traffic, allowing the traffic that has been processed by the first switch continues to traverse the same pipeline without any extra benefit.

To address the challenges, we propose In-Forest, a general distributed deployment framework. By distributing the large-scale ensemble model efficiently across multiple programmable switches, In-Forest enables high accuracy and line-speed processing without single-point resource limitation. In-Forest achieves topology awareness and traffic awareness to determine network-wide model allocation schemes under different resource scenarios and dynamic traffic changes.

III. IN-DEPTH ANALYTICS-DRIVEN MOTIVATION

In this section, we use case studies to validate the necessities and challenges of deploying the ensemble model on the PDP, leading to our design goals for In-Forest.

A. Motivation

1) Ensemble models offer better performance than individual models in network classification tasks: We take the anomaly detection task as an example, evaluating models on pcap files with both malicious and benign traffic. Flow-level features are extracted to generate the dataset, which is then divided into 80% for training and 20% for testing. For classification, we utilize an ensemble model (RF) and an individual model (DT) trained by the widely-used ML framework scikit-learn [36]. Fig. 1 demonstrates the superiority of RF over DT in accuracy and F1 score. Both RF-10 and DT-10 have a maximum depth of 10, with RF consisting of 10 sub-models. We reduce the maximum depth of RF sub-models from 10 to 8, i.e., RF-8, to intentionally decrease their accuracy. The accuracy of RF, although compromised by this reduction, is still superior to DT. This toy case shows us that a) ensemble models outperform individual models in network classification tasks, and b) the performance of sub-models directly affects the performance of the ensemble model. Fig. 1 also shows the rule number of RF and DT under different model representation methods [14, 20, 22]. RF requires up to 17.66× more rules than DT, i.e., 5844 (RF-10) vs. 331 (DT-10), in SwitchTree. The huge hardware resource consumption hinders the deployment of ensemble models on a single switch.

2) Distributed deployment reduces hardware resource consumption while enhancing classification performance: We consider the network-wide model deployment of DT and RF. Fig. 2(a) shows an example topology with three subnets A∼C and six switches 1∼6. Traffic between subnet pairs is routed based on Open Shortest Path First (OSPF) protocol [37]. To cover all the traffic in the network, at least two models need to be deployed, e.g., DT on switches 1 and 5. The deployment, however, causes traffic transmitted from subnet B to A or C to react slowly. This issue can be resolved by deploying the additional model, e.g., DT on switch 3. In this way, two same models are used to process the traffic between subnet pairs A-B (as well as A-C and B-C), wasting resources and not improving accuracy. What's more, as the number of subnets increases, implementing just-in-time traffic analysis requires deploying the same model on almost switches, resulting in a huge waste of network-wide resources.

A more efficient solution is to deploy sub-models on different switches, enabling traffic to pass through multiple sub-models. By aggregating the classification results of sub-models, we can combine their strengths to correct the errors of individual models and prevent overfitting, resulting in higher accuracy with less single-point hardware resource consumption. In our preliminary experiments, we design a Lightweight Ensemble Generic Optional Model (LEGO), which consists of multiple enhanced base models with full functionality and different classification knowledge. Fig. 2(b) shows that LEGO outperforms DT, approaching or even surpassing RF's performance as the number of enhanced base models increases.

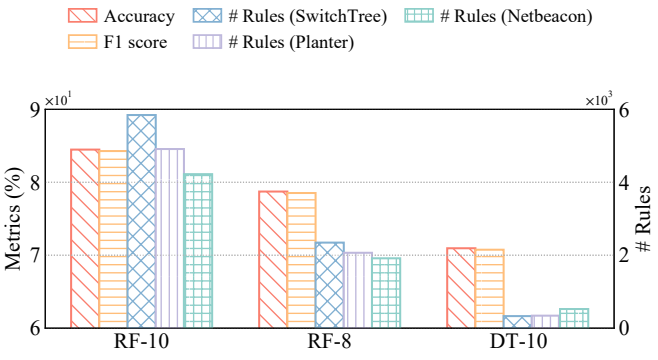
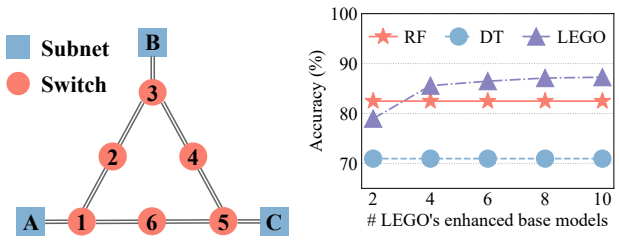


Fig. 1. The comparison of RF and DT in the anomaly detection task. RF-x and DT-x indicate that the maximum depth of RF and DT is x. RF and DT are translated into switch rules by three model representation methods.



(a) Example Topology (b) Accuracy Comparison

Fig. 2. Experimental analysis.

The reason for this improvement is that the accuracy of RF is restricted by the single-point resource limitation, whereas LEGO leverages the resources of multiple switches for deployment and achieves higher accuracy through aggregation. For different resource scenarios and dynamic traffic changes, we can flexibly adjust the allocation schemes of enhanced base models. For example, deploy the optimal three models on switches 1∼3 to provide the best service for traffic between subnets A and B with limited resources. Alternatively, deploy different models on all switches to guarantee network-wide traffic coverage and classification accuracy.

B. Design Goals

In this paper, we propose In-Forest, a distributed deployment framework across multiple switches with the high-accuracy large-scale ensemble model, addressing the challenges of single-point resource limitation and network-wide redundant resource consumption. Each switch only needs to deploy a simple model, but with cooperation, it can achieve high accuracy. In-Forest needs to ensure flexibility in adapting to different resource scenarios. Furthermore, when dynamic traffic changes, models need to be updated to another one in time for higher classification accuracy. In-Forest does not require traffic rescheduling, ensuring its generality.

IV. The Design of In-Forest

A. System Overview

In-Forest is a general distributed in-network classification framework that leverages the available resources of multiple switches to deploy the large-scale ensemble model with high accuracy. Fig. 3 shows a high-level overview of In-Forest. On the control plane, we redesign conventional ensemble models into LEGO to better suit distributed deployment. LEGO consists of multiple enhanced base models with full functionality. A two-phase resource-aware model allocation strategy is employed to determine the optimal allocation schemes under different resource scenarios and dynamic traffic changes. On the data plane, each enhanced base model can be translated into interpretable rules and deployed on a single switch independently, addressing the network classification task at line-speed. We introduce a lightweight model update mechanism to ensure flexibility in model deployment. When traffic passes through multiple switches, In-Forest aggregates the classification results of models to get better performance.

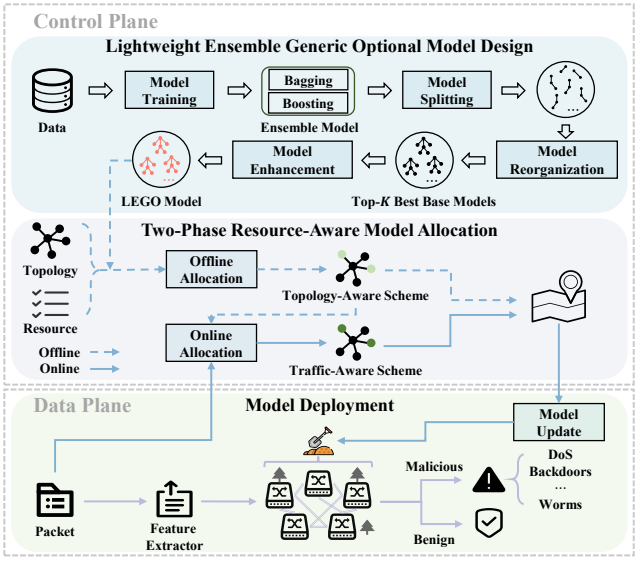


Fig. 3. The architecture of In-Forest.

B. LEGO Design Module

In-Forest utilizes a "splitting-reorganization-enhancement" mechanism to generate LEGO. Algorithm 1 shows the details.

Path-Based Model Splitting. For the raw packet set $\mathcal{C}$, flow ID h_i of each packet is defined by hashing the features (e.g., 5-tuple), and only the first F packets per flow are stored in the flow set $\mathcal{F}$ (lines 1∼5). Crucial flow-level features for network classification tasks [22], such as average/minimum/maximum packet lengths, are extracted to get the training set $\mathcal{X}$ (line 6). Redundant features are eliminated using a backward recursive method (line 7) to reduce model complexity [38]. This method iteratively removes the feature with the least effect based on the cross-validation score until the feature number reaches S. S is determined based on the feature importances [39], balancing complexity and accuracy improvement. The training set $\mathcal{X}$ and the selected feature set $\mathcal{U}'$ are used to generate the tree-based ensemble model $\mathcal{G}$ through bagging or boosting methods (line 8). When classifying a sample, it starts from the first internal node (i.e., root node) and traverses a sequence of internal nodes. Each internal node contains a classification feature and a threshold. The sample's feature is compared to the threshold, determining its direction towards the left or right branch. This branching process continues until a leaf node is reached, which contains the classification result of model $\mathcal{G}$. Suppose there is a path from the root node to a leaf node with the following knowledge: if $u_1 \leq 5, u_2 > 7$, then $class \leftarrow 1$. We can encode the knowledge as a classification path:

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

where a denotes the minimum of features u_1 and b denotes the maximum of feature u_2. The ensemble model $\mathcal{G}$ is split into multiple classification paths and stored in $PathPool$, as depicted in Fig. 4 (line 9).

Coarse-Grained Model Reorganization. Different classification paths correspond to distinct traffic subsets. We reorga-

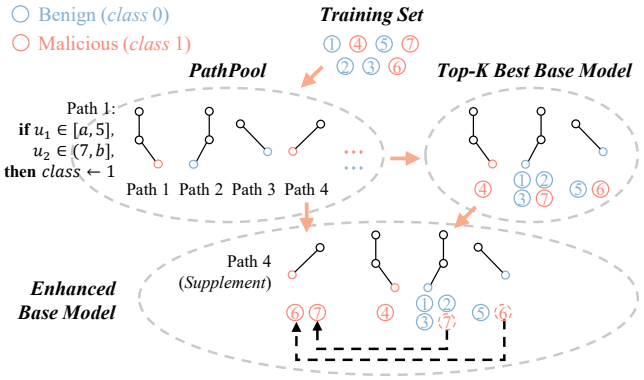


Fig. 4. An example of the LEGO design (with 1 enhanced base model, i.e., K=1). Leaf nodes in the paths output the predicted classification results (blue for benign, while red for malicious), with each corresponding to a subset of traffic. The enhancement mechanism introduces valuable paths to correct misclassifications (e.g., samples 6 and 7).

nize the paths in $PathPool$ into multiple base models $\mathcal{B}$, each of which can process all the traffic (line 10). For instance, the combination of paths 1∼3 in Fig. 4 forms a base model, which possesses the full functionality to address the classification task. Then, we apply the Top-K filtering method to select K base models with the best performance, i.e., $\mathcal{B}'$ (line 11). We define $K \triangleq \min(|\mathcal{B}|, H)$, where H is the maximum number of switches between subnet pairs. For Abilene [40] and GEANT [41] topologies, H is 6 and 8, respectively.

Fine-Grained Model Enhancement. Inspired by Section III-A1, we aim to improve the performance of each base model, ensuring better performance after aggregation. We sort the paths in descending order according to their priorities. The priority is determined as follows:

$$Pr_j \leftarrow \frac{\sum_{y \in \mathcal{X}_j} (y == \hat{y})}{|\mathcal{X}_j|}, \tag{2}$$

where $\mathcal{X}_j$ represents the traffic subset corresponding to path $PathPool_j$ (lines 12∼14). y and $\hat{y}$ denote the true class and the predicted class of the sample.

For classification paths whose combined models are not chosen by the Top-K filtering method, they remain valuable, e.g., path 4 in Fig. 4. Path 4 accurately classifies samples 6 and 7, which can serve as a $Supplement$ to the base model for correcting misclassifications. In this case, both paths 3 and 4 can process sample 6. We opt for the result of path 4 due to its higher priority, which achieves the correct classification of sample 6. The same applies to sample 7. We combine base models with their corresponding $Supplements$ to get the enhanced base models $\mathcal{B}''$ (lines 16∼26). The function CanInsert(·) is used to calculate whether the inserted path brings a performance improvement. LEGO is further obtained by aggregating enhanced base models to prevent overfitting [25] and achieve improved accuracy (line 27).

Notably, classification paths of the enhanced base model are translated into interpretable range match rules and installed in the programmable switch [21, 42] (detailed in Section IV-D).

Algorithm 1: LEGO Design Logic

Input: Raw packet set $\mathcal{C} = \{(\boldsymbol{c}_1, y_1), \ldots\}$; Flow-level feature set $\mathcal{U} = \{u_1, \ldots\}$; Maximum number of packets stored per flow F; Number of selected features S.

1 $\mathcal{F} \leftarrow \{[\,]\}$;
2 **for** $i = 1, \ldots, |\mathcal{C}|$ **do**
3 $\quad h_i \leftarrow$ Hash(GetFiveTuple($\boldsymbol{c}_i$));
4 $\quad$ **if** $\mathcal{F}[h_i]$.length $< F$ **then** $\mathcal{F}[h_i]$.append($\boldsymbol{c}_i$);
5 **end**
6 Get the training set $\mathcal{X} = \{(\boldsymbol{x}_1, y_1), \ldots\}$ by extracting flow-level features from $\mathcal{F}$ based on $\mathcal{U}$;
7 $\mathcal{U}' \leftarrow$ BackwardFeatureSelection($\mathcal{X}, \mathcal{U}, S$);
8 $\mathcal{G} \leftarrow$ Training($\mathcal{X}, \mathcal{U}'$);
9 $PathPool \leftarrow$ ModelSplitting($\mathcal{G}$);
10 $\mathcal{B} \leftarrow$ ModelReorganization($PathPool$);
11 $\mathcal{B}' \leftarrow$ KBestFiltering ($\mathcal{B}$);
12 **for** $j = 1, \ldots, |PathPool|$ **do**
13 $\quad Pr_j \leftarrow \frac{\sum_{y \in \mathcal{X}_j} (y == \hat{y})}{|\mathcal{X}_j|}$;
14 **end**
15 $PathPool$.Sort($Pr, descending = True$);
16 $Supplements, j \leftarrow \{[\,]\}, 1$;
17 **while** $j \leq |PathPool|$ *and* $PathPool_j \notin \mathcal{B}'$ **do**
18 $\quad$ **for** $k = 1, \ldots, |\mathcal{B}'|$ **do**
19 $\quad\quad$ **if** CanInsert($PathPool_j, B'_k$) **then**
20 $\quad\quad\quad Supplements[k]$.append($PathPool_j$);
21 $\quad\quad\quad$ **Break**
22 $\quad\quad$ **end**
23 $\quad$ **end**
24 $\quad j \leftarrow j + 1$;
25 **end**
26 $\mathcal{B}'' \leftarrow$ Concat($\mathcal{B}', Supplements$);
27 $LEGO \leftarrow$ ModelAggregation($\mathcal{B}''$);

Output: $LEGO$ with K enhanced base models.

There are four characteristics of LEGO that make it highly suitable for distributed deployment: a) Lightweight. Through feature selection, path splitting, and model filtering, LEGO reduces hardware resource consumption while maintaining full functionality. b) Ensemble/Enhanced. Each base model is enhanced, further improving the accuracy of the ensemble model. c) Generic. Classification paths can be obtained from bagging or boosting methods, and $Supplements$ are flexibly adapted. d) Optional. Based on the network topology and dynamic traffic, enhanced base models can be selectively combined to achieve superior performance.

C. Two-Phase Resource-Aware Model Allocation Module

When deploying LEGO across the network, resource consumption, traffic coverage, and classification accuracy are important metrics in determining the optimal allocation scheme of enhanced base models. However, directly modeling the correspondence between the model allocation scheme and network traffic is challenging and unreliable. Due to the

dynamic nature of traffic, the current allocation scheme may not adapt effectively to future traffic. To overcome this, we employ a two-phase resource-aware model allocation strategy.

Offline Topology-Aware Allocation. In-Forest employs the offline phase to obtain topology-aware model allocation schemes under different resource scenarios, where traffic coverage is converted to path coverage for subnet pairs and classification accuracy is converted to the diversity of enhanced base models across all paths. Then, we simplify the model allocation problem as follows:

$$\begin{aligned}
\max_{\mathcal{D}} \quad & \alpha_1 \sum_{n=1}^{N} \text{Step}\left(\sum_{w=1}^{W} \boldsymbol{p}_{n_w} \sum_{b=1}^{B} \mathcal{D}_{w,b}\right) + \\
& \alpha_2 \sum_{n=1}^{N} \sum_{b=1}^{B} \text{Step}\left(\sum_{w=1}^{W} \boldsymbol{p}_{n_w} \mathcal{D}_{w,b}\right) - \\
& \alpha_3 \sum_{w=1}^{W} \sum_{b=1}^{B} \mathcal{D}_{w,b} e_b \qquad (3) \\
s.t. \quad & \sum_{w=1}^{W} \text{Step}\left(\sum_{b=1}^{B} \mathcal{D}_{w,b}\right) \leq D; \qquad (3a) \\
& \sum_{w=1}^{W} \sum_{b=1}^{B} \mathcal{D}_{w,b} e_b \leq E; \qquad (3b) \\
& \sum_{b=1}^{B} \mathcal{D}_{w,b} \leq 1, w = 1, \cdots, W; \qquad (3c) \\
& \mathcal{D}_{w,b} \in \{0,1\}, w = 1, \cdots, W, b = 1, \cdots, B. \qquad (3d)
\end{aligned}$$

Table II summarizes the variables. The objective function is to optimize path coverage and model diversity while minimizing the required switch rules. $\mathcal{D}$ represents the variable for determining the model allocation scheme. $\boldsymbol{p}_n$ is a one-dimensional vector of length W encoded with binary values, indicating whether switch w lies on the selected path between the subnet pair n. The function Step$(\cdot)$ converts values that are greater than 0 to 1 and others to 0. To ensure the comparability among three metrics in the objective function, we normalize them to the range of $[0, 1]$. α_1, α_2, and α_3 are weights to the metrics, with value of $\alpha_1 = 0.6$, $\alpha_2 = 0.2$, and $\alpha_3 = 0.2$. Model allocation schemes under different resource scenarios can be obtained by adjusting D and E.

The model allocation problem is computationally complex, resembling the multiple knapsack problem [43]. In the multiple knapsack problem, there are B items and W knapsacks. Items have different gains when assigned to different knapsacks and the goal is to maximize the overall gain. Our problem is more complex as there is no explicit gain function between the items (enhanced base models) and knapsacks (switches). Since the multiple knapsack problem is known as NP-hard, our problem is also NP-hard. To efficiently solve this, we employ a heuristic algorithm, i.e., Genetic Algorithm (GA) [44].

In practice, W is set to the number of switches in the real network topologies (i.e., 11 for Abilene [40] and 23 for GEANT [41]), and B is equal to the value of K set in Section IV-B. We randomly generate a population containing many $\mathcal{D}$ with different values and flatten each into a one-dimensional vector of length $W \times B$. Each vector represents a candidate scheme and is evaluated by the fitness (objective function in Equation (3)). We iteratively evolve the population and the schemes with higher fitness are more likely to reproduce in the next iteration. Offspring are generated through crossover and mutation, replacing partial schemes to maintain population size. The process iterates until the fitness converges. To ensure the satisfaction of constraints (3a)∼(3c), we penalize non-compliant schemes by assigning a large negative value to fitness. The scheme with the highest fitness is deemed the optimal model allocation scheme in the offline phase.

TABLE II
SUMMARY OF VARIABLES IN THE MODEL ALLOCATION PROBLEM

Variable	Description
W	Number of switches
B	Number of enhanced base models in $\mathcal{B}''$
N	Number of subnet pairs
D	Maximum number of deployed switches
E	Maximum number of rules stored in the network
$\mathcal{D}w, b$	Whether to deploy enhanced base model b on switch w
$\boldsymbol{p}_n$	Selected path between the subnet pair n
e_b	Number of rules for enhanced base model b

Online Traffic-Aware Allocation. To ensure In-Forest adapts to dynamic traffic changes, we introduce an online phase. By selecting the most suitable enhanced base models for classifying corresponding traffic, we can maximize the accuracy. We employ a model-free DRL approach to address the challenges in dynamic traffic modeling.

We formulate the traffic transmission as a Markov Decision Process (MDP) to effectively utilize DRL. The MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T})$ [45], where $\mathcal{S}$ represents the state space, $\mathcal{A}$ represents the action space, $\mathcal{R}$ represents the reward space, and $\mathcal{T}$ represents the state transition probability. At time step t, In-Forest receives network information $\boldsymbol{s}_t$ and determines the model allocation scheme $\boldsymbol{a}_t$, which then gets a reward r_t. The MDP aims to find an optimal policy $\pi_{\boldsymbol{\theta}}$ that maximizes the objective function $\mathbf{J}(\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ represents the policy parameters. Policy Gradient (PG) [46] methods are commonly employed to update $\pi_{\boldsymbol{\theta}}$, but they suffer from inefficient data sampling [40]. To address this problem, we utilize the Proximal Policy Optimization (PPO) algorithm [47], which is known for its improved sampling efficiency and reduced training variance. PPO leverages importance sampling to efficiently make use of the data sampled by the old policy parameters $\boldsymbol{\theta}'$. The update of policy $\pi_{\boldsymbol{\theta}}$ follows the form:

$$\nabla_{\boldsymbol{\theta}} \mathbf{J}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\tau} \sim \mathbf{P}_{\boldsymbol{\theta}'}(\boldsymbol{\tau})}[\mathbf{W}(\boldsymbol{\theta}') \mathbf{A}(\boldsymbol{s}, \boldsymbol{a}) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})], \quad (4)$$

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 in a given state [48]. The importance weight is denoted as $\mathbf{W}(\boldsymbol{\theta}') = \frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\boldsymbol{\theta}'}(\boldsymbol{a}|\boldsymbol{s})}$.

It is important to note that the new policy parameters $\boldsymbol{\theta}$ and old policy parameters $\boldsymbol{\theta}'$ should not differ significantly. Following [47], we adjust the objective function to limit the gap between them. The new objective function is:

$$\mathbf{J}^{CLIP}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\tau} \sim \mathbf{P}_{\boldsymbol{\theta}'}(\boldsymbol{\tau})}[\min(\mathbf{W}(\boldsymbol{\theta}') \mathbf{A}(\boldsymbol{s}, \boldsymbol{a}), \mathbf{W}'(\boldsymbol{\theta}') \mathbf{A}(\boldsymbol{s}, \boldsymbol{a}))], \quad (5)$$

where $\mathbf{W}'(\boldsymbol{\theta}') = \text{Clip}(\mathbf{W}(\boldsymbol{\theta}'), 1-\epsilon, 1+\epsilon)$ is used to ensure the probability of the good (or bad) action does not increase (or decrease) substantially [47]. We set ϵ to 0.2.

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

$$CFA \triangleq \frac{\sum_{l=1}^{L} m_l(y_l == \hat{y}_l)}{L}, \tag{6}$$

where m_l represents whether flow l is covered by enhanced base models. If models are deployed on the routing path of flow l, we set $m_l = 1$, otherwise, $m_l = 0$. y_l and $\hat{y}_l$ are the true class and the predicted class, respectively.

In the state space, we consider the flow feature (5-tuple of each flow), the traffic distribution (flow number in links), and the current model allocation scheme. The action output for each switch is a vector of length $B+1$. The first B 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 of each switch and combine them as the global action.

To ensure the effectiveness of the online phase, we employ a stable learning mechanism. We determine the deployed switches through the offline phase and then use the online phase to tune the enhanced base models on these switches. This can reduce the dimensions of action space for faster convergence. In addition, the allocation scheme from the offline phase serves as the initial value of PPO, which explores and refines the scheme further. 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 satisfy the constraints. We add a penalty value to the reward for schemes with lower performance than the offline phase or that do not satisfy the constraints (3a)$\sim$(3c).

D. Model Deployment Module

In-Forest deploys enhanced base models on different programmable switches to achieve in-network classification. Each switch aggregates the classification results of enhanced base models deployed on preceding switches, improving accuracy. Furthermore, a lightweight model update mechanism is designed to maintain flexibility in model deployment.

Feature Extraction. In-Forest uses flow-level features for classification, as described in [22]. These features are obtained by combining attributes from other packets within the same flow. To balance memory consumption and classification accuracy, we extract features by the first F packets per flow, where F is set to 4. Features, e.g., average/minimum/maximum packet lengths, are taken into account.

Model Representation. Once the features are extracted, their values are used as input for classification. Each enhanced base model can be deployed as a single match-action table matching all the features. Each classification path of the enhanced base model, as shown in Equation (1), is translated into a range match rule and stored in the table. This model representation method ensures flexibility in adding or deleting switch rules for model updating.

Classification Result Aggregation. We use a packet header field to record the classification results of models deployed on the preceding switches. Specifically, a variable $prob$ is initialized with the value of H, where H represents the maximum number of switches between subnet pairs. In the anomaly detection task, if an enhanced base model classifies the incoming flow as malicious, we increment $prob$ by 1. Subsequently, we check the value of $prob$. If $prob$ exceeds H, it indicates that at least one model predicts the flow as malicious, and a larger $prob$ signifies higher confidence. Conversely, if $prob$ is less than H, it implies that at least two preceding models classify the flow as benign, which means that the result of the current model is probably wrong and the result will be corrected by majority voting. When $prob$ equals H, forwarding of the flow continues, indicating that the flow's class cannot be determined at that point. The comparison of $prob$ can be implemented by range match rules [42].

Lightweight Model Update Mechanism. To enhance the flexibility in model deployment, we introduce a lightweight model update mechanism. Benefiting from the model representation method we use, each switch rule is obtained from the translation of a classification path. The switch rule obtained from the classification path with higher priority in Equation (2) also has higher priority. Switch rules with higher priority are given preference for matching PHVs. When the available resources change, we can achieve optimal model scaling in (or out) by adding (or deleting) switch rules with higher (or lower) priority. In the online phase, it is crucial to ensure timely model updates to respond to dynamic traffic changes. For the enhanced base model on each switch, we can update it with another one that has the same number of rules. This updating can be accomplished by changing the rules alone, eliminating the need for switch restarts.

V. Evaluation

Our evaluation of In-Forest focuses on (a) the lightweight and enhancement of LEGO in comparison to conventional ensemble models; (b) the effectiveness and superiority of network-wide model distributed deployment under different resource scenarios and dynamic traffic changes compared to baselines; and (c) the lower hardware resource consumption.

A. Experiment Setup

We simulate traffic transmission by building two real network topologies, i.e., Abilene [40] with 11 switches and 14 bidirectional links as well as GEANT [41] with 23 switches and 37 bidirectional links, on Linux servers. The servers are equipped with NVIDIA GeForce RTX 2080Ti GPUs and Intel Xeon Gold 6230R CPUs. The tree-based models are trained by the commonly used ML framework scikit-learn [36]. For the online traffic-aware model allocation, we implement the PPO [47] algorithm in PyTorch [49]. Model deployment is implemented on Tofino 1 switches, i.e., H3C S9850-32H[1].

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

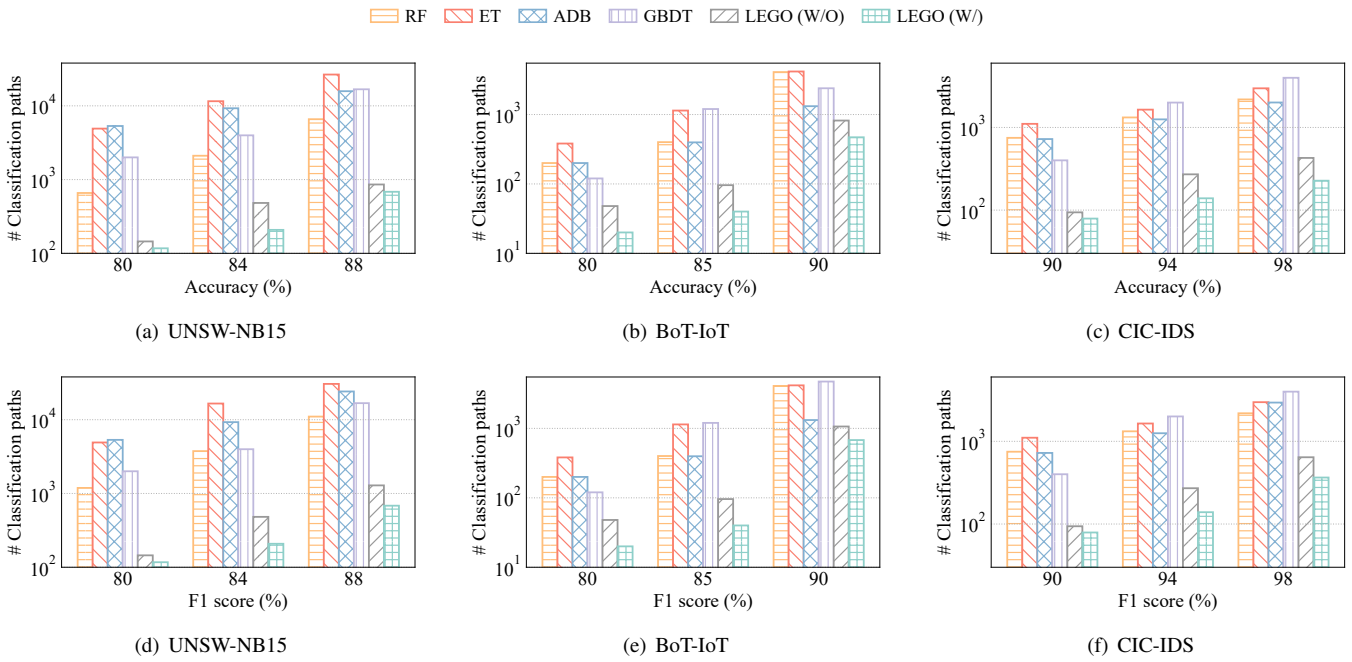


Fig. 5. The comparison in the number of classification paths required by LEGO and conventional ensemble models (RF, ET, ADB, and GBDT) to achieve the same performance (accuracy and F1 score) under three anomaly detection datasets, i.e., UNSW-NB15, BoT-IoT, and CIC-IDS.

B. LEGO Performance

LEGO is a core component of In-Forest, so we first compare it to four conventional ensemble models, i.e., Random Forest (RF) [25], Extra Trees (ET) [50], Adaboost (ADB) [31], and Gradient Boosting Decision Tree (GBDT) [32].

The comparison focuses on the number of classification paths required to achieve the same accuracy and F1 score. As the most popular and important task for in-network classification, various recent works highlight the extensive usage and significance of the three datasets (UNSW-NB15 [51], BoT-IoT [52], and CIC-IDS [53]) in anomaly detection [14, 18–21, 54–56]. We conduct experiments on the datasets to identify whether the real-world flows are malicious or benign. Table III provides the relevant details. Each dataset is divided into two parts, i.e., 80% for training and 20% for testing. The conventional ensemble models are trained by scikit-learn. Since scikit-learn does not support generating LEGO, we implement the design logic (detailed in Section IV-B) from scratch, splitting the classification paths from RF. As suggested by [22], we use the flow-level features for training, including average/minimum/maximum packet lengths. We set the number of selected features, i.e., S, in LEGO to 8 and all other models use the same number of features.

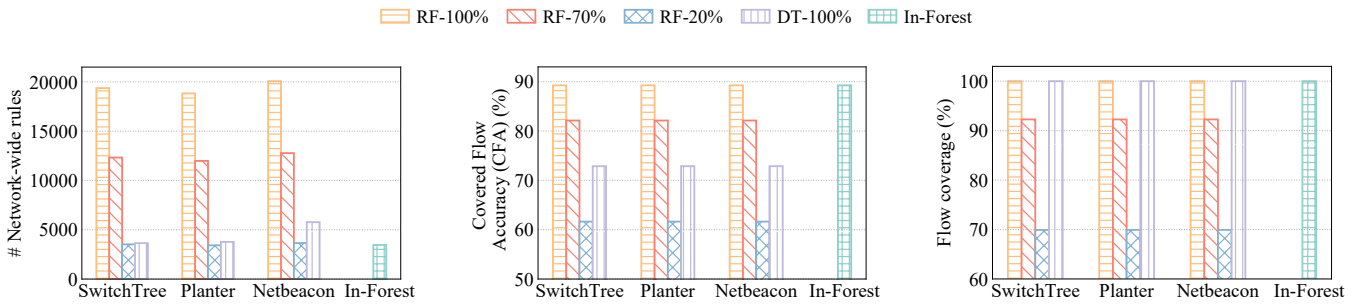
Fig. 5 illustrates that LEGO is lightweight and enhanced, achieving the same performance with fewer classification paths. We control the performance of each model by changing depth while maintaining the other parameters unchanged. LEGO (W/O) represents a trimmed version without fine-grained model enhancement, while LEGO (W/) is the full version. Compared to RF, both versions of LEGO achieve the same accuracy while notably decreasing the number of classification paths. LEGO (W/O) reduces paths by 77.10% (from 2105 to 482) and LEGO (W/) reduces paths by 90.12% (from 2105 to 208), in UNSW-NB15. LEGO (W/O) showcases the contribution of path-based model splitting and coarse-grained model reorganization in forming lightweight base models, eliminating redundant information. The fine-grained model enhancement in LEGO (W/) further enhances base models by inserting valuable paths as *Supplements* to ensure the capability of classification paths. In contrast, conventional ensemble models suffer from heavy sizes, e.g., 30563 classification paths of ET in UNSW-NB15, limiting their practical deployment. They need to compromise between model size and performance, while LEGO is not restricted by this limitation. Moreover, only the enhanced base model needs to be deployed on a switch, which requires fewer resources.

C. Model Distributed Deployment Performance

We extend LEGO for network-wide deployment to show the effectiveness and superiority of In-Forest. In-Forest is compared to three advanced in-network classification solutions, i.e., SwitchTree [14], Planter [20], and Netbeacon [22]. These solutions adopt different model representation meth-

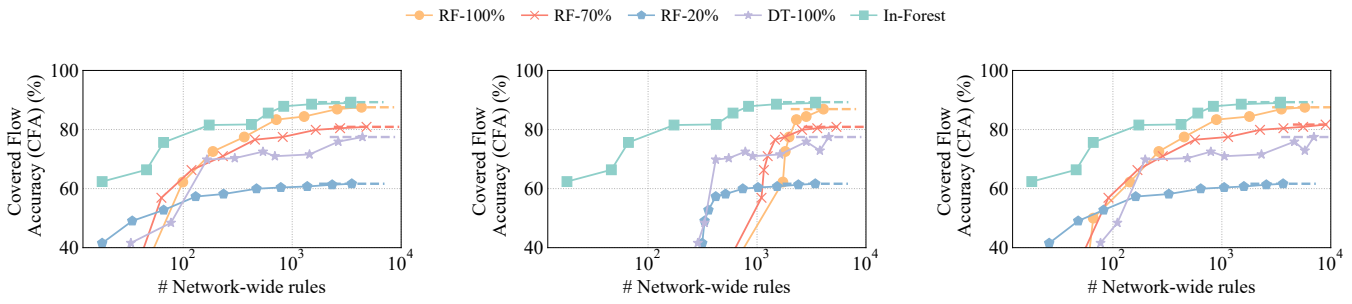
TABLE III
EXPERIMENTAL DETAILS

Dataset	Training (flows)	Testing (flows)	Class
UNSW-NB15 [51]	500K	125K	9 attacks and 1 benign
BoT-IoT [52]	1108K	277K	5 attacks and 1 benign
CIC-IDS [53]	1420K	355K	8 attacks and 1 benign



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

Fig. 6. The network-wide model deployment performance of In-Forest and existing solutions (SwitchTree, Planter, and Netbeacon) in Abilene. To compare with the distributed deployment of In-Forest, we design four baselines (RF-100%, RF-70%, RF-20%, and DT-100%) for the existing solutions.



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

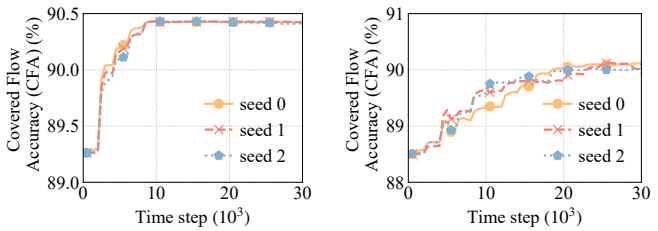
Fig. 7. The CFA of In-Forest and existing solutions (SwitchTree, Planter, and Netbeacon) under different numbers of network-wide rules in Abilene. To compare with the distributed deployment of In-Forest, we design four baselines (RF-100%, RF-70%, RF-20%, and DT-100%) for the existing solutions.

ods. Specifically, SwitchTree utilizes direct mapping, while Planter and Netbeacon use feature encoding. The comparison relies on the number of network-wide switch rules needed to achieve equal CFA (defined in Equation (6)) and flow coverage (percentage of flows with models deployed on the routing path). The number is calculated by summing up the rules required on each switch. We use UNSW-NB15 as the flow dataset and Abilene as the network topology to simulate traffic transmission. Eight subnets are connected to different switches. We use a hash function based on the 5-tuple to assign each flow to a subnet pair, and the flow routing is determined by OSPF [37] protocol. RF is selected as the ensemble model for the deployment of existing solutions because it is supported by each of them. Deploying RF requires a limit on the size due to the single-point resource limitation. In addition, we consider the deployment of DT to verify if the distributed deployment of In-Forest can outperform the individual model through ensemble learning. Since the existing solutions lack a network-wide distributed deployment mechanism, we extend them by adding four deployment methods (RF-100%, RF-70%, RF-20%, and DT-100%), and use them as baselines:

- RF-100%. Deploy RF on all switches.
- RF-70% and RF-20%. Deploy RF randomly on 70% or 20% switches.
- DT-100%. Deploy DT on all switches.
- In-Forest. Employ the offline model allocation to choose the optimal allocation scheme of enhanced base models.

In RF-100%, RF-70%, RF-20%, and DT-100%, the complete RF or DT is deployed on every single switch by the existing solutions (SwitchTree, Planter, and Netbeacon). The model depth is set to 10, as deeper trees are impractical for switch deployment [18, 30]. Similar to [30], RF is configured as 3 sub-models. Fig. 6 shows that In-Forest achieves the same CFA and flow coverage compared to RF-100%, while significantly reducing the number of rules, e.g., from 19371 (SwitchTree) to 3454. In Fig. 6(a), we can find that, even with the partial deployment of RF (RF-70% and RF-20%), the rule number in SwitchTree remains higher than In-Forest. Notably, reducing rules by using RF-70% and RF-20% does not guarantee flow coverage (Fig. 6(c)), and traffic rescheduling is required which leads to complex network routing and management. Compared to DT-100%, In-Forest achieves a 16.39% higher CFA (89.26% vs. 72.87%). Since the models are the same, a flow does not get any improvement after passing through multiple DTs. In contrast, In-Forest can aggregate the classification results of different enhanced base models, leading to improved accuracy, while covering all the traffic without rescheduling.

Additionally, we perform a comprehensive comparison under different resource scenarios. We limit the number of rules stored across the entire network. In-Forest adapts to different resource scenarios by adjusting D and E for optimal model allocation (detailed in Section IV-C). Simultaneously, LEGO's model depth is adjusted and the optimal model scaling in/out is achieved through the lightweight model update mechanism



(a) Abilene ($H = 6$) (b) Geant ($H = 8$)

Fig. 8. The CFA under dynamic traffic changes in two topologies. We set three different random seeds (i.e. seeds 0/1/2) to simulate diverse changes.

TABLE IV
AVERAGE CFA IMPROVEMENT IN THE ONLINE PHASE

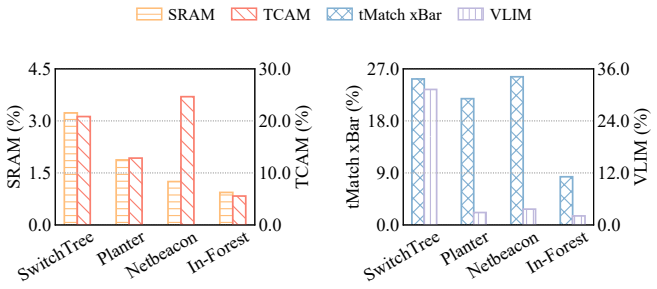
Network Topology	Average CFA (%)		Improvement
	Offline	Offline + Online	
Abilene [40]	89.26%	90.41%	1.15% ↑
GEANT [41]	88.50%	90.07%	1.57% ↑

(detailed in Section IV-D). Larger D and E indicate that the entire network has more hardware resources available for model deployment, resulting in higher CFA. Conversely, when resources are limited, network managers set smaller D and E. The other four baselines do not have a similar mechanism and adapt by only adjusting the model depth. Fig. 7 shows that In-Forest has the highest CFA. Compared to RF-100%, In-Forest can improve the accuracy by 19.31% (from 62.21% to 81.52%) while reducing the number of switch rules by 89.98% (from 1727 to 173), as shown in Fig. 7(b). Furthermore, when they have the same CFA (i.e., 81.52%), the number of rules for In-Forest is reduced by 92.47% (from 2299 to 173).

Next, we demonstrate In-Forest's traffic awareness in Abilene and GEANT topologies with different H settings. H depends on the topology scale, ensuring the diversity of enhanced base models to improve the classification performance after aggregation. Specifically, H is 6 in Abilene and 8 in GEANT. We set the survival time for each flow, which is 10 time steps in Abilene and 15 time steps in GEANT, allowing the traffic to change over time [45]. To simulate diverse traffic changes, we add a random number from 0 to 3 into the survival time and set three different random seeds, i.e., 0, 1, and 2. We choose the model allocation scheme in the offline phase as a good initial value and employ the online phase to sense the dynamic traffic changes. Fig. 8 demonstrates the effectiveness of the online phase, which converges in both network topologies and brings a better CFA compared to the offline phase. We average the results under different random seeds to avoid evaluation noise. Table IV shows the average CFA improvement, i.e., 1.15% ↑ in Abilene and 1.57% ↑ in GEANT, respectively.

D. Hardware Resource Performance

We now compare In-Forest with SwitchTree, Planter, and Netbeacon in single-point switch resource consumption. In our setting, SwitchTree uses direct mapping to deploy RF,



(a) Memory Resources (b) Computational Resources

Fig. 9. The resource consumption of different solutions on the H3C switch.

while Planter and Netbeacon use feature encoding. In-Forest deploys the enhanced base model. The models are all from the task of UNSW-NB15 in Section V-C, 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 or range match, and VLIM is used for actions [21].

Fig. 9 illustrates that In-Forest consumes the lowest switch resources. For instance, in Fig. 9(a), the SRAM usage for SwitchTree and In-Forest is respectively 3.23% and 0.94% (i.e., 70.90% ↓) and the TCAM usage for Netbeacon and In-Forest is respectively 24.65% and 5.55% (i.e., 77.48% ↓). In Fig. 9(b), In-Forest occupies the least percentage of both tMatch xBar and VLIW, e.g., 31.25% (SwitchTree), 2.86% (Planter), 3.65% (Netbeacon), and 2.08% (In-Forest) of VLIW.

VI. CONCLUSION

In this paper, we propose In-Forest, a general distributed in-network classification framework. Firstly, we design a Lightweight Ensemble Generic Optional Model (LEGO). LEGO can be transformed into multiple enhanced base models, each of which can enable line-speed network classification on a single switch. Secondly, we propose a two-phase resource-aware model allocation strategy to optimize the allocation schemes. In addition, we devise a lightweight model update mechanism to ensure flexibility in adapting to different resource scenarios and dynamic traffic changes. Experimental results show that In-Forest outperforms existing in-network classification solutions in accuracy and hardware resource consumption under network-wide model deployment.

VII. ACKNOWLEDGMENTS

We thank our shepherd Kuai Xu and the anonymous reviewers for their helpful feedback. This work is supported in part by the National Key R&D Program of China under Grant No. 2022YFB3105000, the National Natural Science Foundation of China under Grant No. 61972189, the Major Key Project of PCL under Grant No. PCL2023AS5-1, and the Shenzhen Key Lab of Software Defined Networking under Grant No. ZDSYS20140509172959989.

REFERENCES

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

[2] T. Panayiotou, M. Michalopoulou, and G. Ellinas, "Survey on machine learning for traffic-driven service provisioning in optical networks," *IEEE Communications Surveys & Tutorials*, 2023.

[3] Y. Yan, F. Li, W. Wang, and X. Wang, "Talentsketch: Lstm-based sketch for adaptive and high-precision network measurement," in *IEEE International Conference on Network Protocols (ICNP)*, 2022, pp. 1–12.

[4] X. Zhang, L. Cui, F. P. Tso, and W. Jia, "pheavy: Predicting heavy flows in the programmable data plane," *IEEE Transactions on Network and Service Management (TNSM)*, pp. 4353–4364, 2021.

[5] 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 Transactions on Network and Service Management (TNSM)*, pp. 4225–4246, 2021.

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

[7] Z. Wu, Y. Dong, X. Qiu, and J. Jin, "Online multimedia traffic classification from the qos perspective using deep learning," *Computer Networks (CN)*, p. 108716, 2022.

[8] O. Aouedi, K. Piamrat, and D. Bagadthey, "A semi-supervised stacked autoencoder approach for network traffic classification," in *IEEE International Conference on Network Protocols (ICNP)*, 2020, pp. 1–6.

[9] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *ISOC Network and Distributed System Security (NDSS) Symposium*, 2018, pp. 1–15.

[10] B. M. Xavier, R. S. Guimarães, G. Comarela, and M. Martinello, "Programmable switches for in-networking classification," in *IEEE International Conference on Computer Communications (INFOCOM)*, 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 *IEEE International Conference on Network Protocols (ICNP)*, 2021, pp. 1–11.

[12] T. Zheng and B. Li, "Poisoning attacks on deep learning based wireless traffic prediction," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2022, pp. 660–669.

[13] M. E. Kanakis, R. Khalili, and L. Wang, "Machine learning for computer systems and networking: A survey," *Computing Surveys (CSUR)*, pp. 1–36, 2022.

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

[15] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric ddos attacks with programmable switches," in *ISOC Network and Distributed System Security (NDSS) Symposium*, 2020, pp. 1–18.

[16] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. Ramos, and A. Madeira, "Flowlens: Enabling efficient flow classification for ml-based network security applications." in *ISOC Network and Distributed System Security (NDSS) Symposium*, 2021, pp. 1–18.

[17] J. Xing, Q. Kang, and A. Chen, "Netwarden: mitigating network covert channels while preserving performance," in *Usenix Security Symposium (USENIX Security)*, 2020, pp. 2039–2056.

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

[19] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," in *ACM Workshop on Hot Topics in Networks (HotNets)*, 2019, pp. 25–33.

[20] C. Zheng and N. Zilberman, "Planter: seeding trees within switches," in *ACM SIGCOMM Poster*, 2021, pp. 12–14.

[21] 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 *IEEE International Conference on Computer Communications (INFOCOM)*, 2022, pp. 1938–1947.

[22] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *Usenix Security Symposium (USENIX Security)*, 2023, pp. 1–18.

[23] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, and G. Varghese, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review (CCR)*, pp. 87–95, 2014.

[24] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, "Classification and regression trees," *Biometrics*, p. 358, 1984.

[25] L. Breiman, "Random forests," *Machine Learning*, pp. 5–32, 2001.

[26] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *ACM Knowledge Discovery and Data Mining (SIGKDD)*, 2016, pp. 785–794.

[27] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven streaming network telemetry," in *ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 2018, pp. 357–371.

[28] Barefoot Networks, "Tofino switch," Website, https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series/tofino.html.

[29] 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 *ACM International Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 2017, pp. 15–28.

[30] A. T. Akem, B. Bütün, M. Gucciardo, and M. Fiore, "Henna: hierarchical machine learning inference in programmable switches," in *International Workshop on Native Network Intelligence (NativeNI)*, 2022, pp. 1–7.

[31] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences (JCSS)*, pp. 119–139, 1997.

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

[33] Y. Li, J. Sun, W. Huang, and X. Tian, "Detecting anomaly in large-scale network using mobile crowdsourcing," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2019, pp. 2179–2187.

[34] X. Ji, B. Han, R. Li, C. Xu, Y. Li, and J. Su, "Access: Adaptive qos-aware congestion control for multipath tcp," in *IEEE/ACM International Symposium on Quality of Service (IWQoS)*, 2022, pp. 1–10.

[35] S. Wang, C. Sun, Z. Meng, M. Wang, J. Cao, M. Xu, J. Bi, Q. Huang, M. Moshref, and T. Yang, "Martini: Bridging the gap between network measurement and control using switching asics," in *IEEE International Conference on Network Protocols (ICNP)*, 2020, pp. 1–12.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, and

V. Dubourg, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research (JMLR)*, pp. 2825–2830, 2011.
[37] B. Fortz and M. Thorup, "Optimizing ospf/is-is weights in a changing world," *IEEE Journal of Selected Areas in Communications (JSAC)*, pp. 756–767, 2002.
[38] 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*, 1994, pp. 403–413.
[39] B. H. Menze, B. M. Kelm, R. Masuch, U. Himmelreich, P. Bachert, W. Petrich, and F. A. Hamprecht, "A comparison of random forest and its gini importance with standard chemometric methods for the feature selection and classification of spectral data," *BMC Bioinformatics*, pp. 1–16, 2009.
[40] C. Liu, M. Xu, Y. Yang, and N. Geng, "Drl-or: Deep reinforcement learning-based online routing for multi-type service requirements," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2021, pp. 1–10.
[41] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM Computer Communication Review (CCR)*, pp. 83–86, 2006.
[42] H. Siddique, M. Neves, C. Kuzniar, and I. Haque, "Towards network-accelerated ml-based distributed computer vision systems," in *International Conference on Parallel and Distributed Systems (ICPADS)*, 2021, pp. 122–129.
[43] "Solving multiple knapsack problems by cutting planes," *SIAM Journal on Optimization (SIOPT)*, pp. 858–877, 1996.
[44] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, pp. 65–85, 1994.
[45] C. Liu, M. Xu, Y. Yang, and N. Geng, "Drl-or: Deep reinforcement learning-based online routing for multi-type service requirements," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2021, pp. 1–10.
[46] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning (ICML)*, 2015, pp. 1889–1897.
[47] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
[48] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.
[49] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, and L. Antiga, "Pytorch: An imperative style, high-performance deep learning library," *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pp. 1–12, 2019.
[50] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, pp. 3–42, 2006.
[51] N. Moustafa and J. Slay, "Unsw-nb15: A comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *Military Communications and Information Systems Conference (MilCIS)*, 2015, pp. 1–6.
[52] 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 Generation Computer Systems (FGCS)*, pp. 779–796, 2019.
[53] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *International Conference on Information Systems Security and Privacy (ICISSP)*, pp. 108–116, 2018.
[54] C. Zheng, M. Zang, X. Hong, R. Bensoussane, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "Automating in-network machine learning," *arXiv preprint arXiv:2205.08824*, 2022.
[55] C. Zheng, Z. Xiong, T. T. Bui, S. Kaupmees, R. Bensoussane, A. Bernabeu, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "Iisy: Practical in-network classification," *arXiv preprint arXiv:2205.08243*, 2022.
[56] B. M. Xavier, R. S. Guimarães, G. Comarela, and M. Martinello, "Map4: A pragmatic framework for in-network machine learning traffic classification," *IEEE Transactions on Network and Service Management (TNSM)*, pp. 4176–4188, 2022.