LINC: Enabling Low-Resource In-network Classification and Incremental Model Update

Haolin Yan[1,2], Qing Li[2], Guorui Xie[1,2], Gareth Tyson[3], Yong Jiang[1,2]
[1] International Graduate School, Tsinghua University, Shenzhen, China [2] Peng Cheng Laboratory, Shenzhen, China
[3] The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China

2024 IEEE 32nd International Conference on Network Protocols (ICNP) | 979-8-3503-5171-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICNP61940.2024.10858585

***Abstract*—Deploying machine learning models within the data plane can facilitate traffic analysis at line rate. Nevertheless, existing schemes suffer from performance degradation when faced with hardware resource constraints. They further cannot accommodate incremental model updates as the underlying traffic distributions change. This paper presents LINC, a novel framework for low-resource and incrementally updatable in-network classification. To circumvent the complicated P4 program that consumes extensive resources, LINC generalizes explicit deployable classification rules by interpreting a customized neural network. For incremental updates with minimal new training data, we design a divide-and-conquer strategy that decomposes the update task into binary classification tasks within distinct subspaces to simplify the task. We then update the model locally within each subspace to improve accuracy for new categories without compromising the accuracy of existing categories. Experimental results reveal that, compared to state-of-the-art solutions, LINC significantly reduces switch hardware resources by up to $15.6\times$ and achieves up to 11.63% higher classification accuracy. During the incremental model updates, LINC not only improves the accuracy by 26.1% but also decreases the table entries by $6\times$.**

I. INTRODUCTION

The advent of programmable network devices, such as programmable switches [1] and smart network interface cards (NICs) have enabled the development of more adaptable networking functions. These include monitoring [2]–[4], caching [5]–[7], failure localization [8]–[11], distributed coordination [12]–[14], and load balancing [15]–[17]. One particularly important example is in-network classification [18], which aims to identify a flow's traffic category within the data plane. This is helpful for various applications, such as Quality of Service (QoS) aware prioritization, implementing security measures, and enabling targeted network optimization.

In contrast to traditional solutions (e.g., empirically derived traffic filters [19]), there have been recent efforts [18], [20]–[25] to deploy popular machine learning (ML) models in the data plane to improve in-network classification performance, while maintaining line speed inference (aka in-network ML [20]). Compared to server-based solutions [26]–[28], which deploy the ML models in back-end servers, in-network ML bypasses traffic redirection to dedicated servers, thus reducing response delay. This is essential for some latency-sensitive applications such as DDoS mitigation. However, these prior in-network ML solutions lack *two* core requirements.

The *first* requirement is that any solution must be deployable within low-resource environments while retaining high model performance. Yet, current solutions fall short of this requirement as they have to sacrifice model performance in order to adapt to the resource constraints of network devices. For example, P4 switches (e.g., with Intel Tofino ASIC [29]) have a limited number of Match-Action (MA) stages and memory resources such as SRAM and TCAM. In addition, a native switch needs to allocate some (or most) of its resources to basic networking functions such as forwarding [18]. Thus, in-network classification must be performed in a very resource-efficient manner. Although some resource-efficient schemes [21]–[23] have been proposed, compared to previous schemes [30], [31], they still have to trade-off model complexity (i.e., accuracy) in low-resource scenarios. For example, Planter [21] reduces the number of features and tree estimators to prevent an excessive number of MA tables, and Mousikav2 [23] reduces the amount of training data to mitigate the problem of combinatorial explosion [22], [24]. In both cases, this reduces model accuracy. **Therefore, designing a more resource-efficient scheme for in-network classification remains a key challenge.**

The *second* requirement is that any solution must be incrementally updateable [32]–[34]. This is because the traffic distribution of a typical switch will change over time. Thus, when it is found that a model fails (e.g., due to traffic changes), the model should be updated to identify new traffic patterns correctly. To update models quickly and minimize traffic disruption, current schemes use traditional methods [18], i.e., retraining ML models on both old and new data simultaneously, and directly updating table entries in MA tables without modifying the switch program. However, in practice, it is difficult to manually annotate enough fresh data for model updates within a short period. Therefore, the amount of new data is usually far less than that of the old data. This means the updated model is usually over-fitted and exhibits low classification accuracy on the new data (i.e., the Few-Shot Learning (FSL) problem [35], [36]). **Thus, designing an incrementally updateable solution that can cope with limited new data remains a key challenge.**

To address these two requirements, we present LINC, a novel framework for **L**ow-resource and **IN**crementally up-

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

979-8-3503-5171-2/24/$31.00 ©2024 IEEE

datable in-network **C**lassification. For the low-resource requirement, LINC customizes a novel neural network and converts it into deployable classification rules by interpreting the learned mappings from input features to target categories. This approach exploits neural networks to directly learn a small number of efficient and complex classification rules from the data, rather than using knowledge distillation to assist decision trees in learning redundant basic rules [23]. This significantly reduces resource consumption. For the incremental updates requirement, where the new data is significantly smaller than the old data, LINC decomposes the update task into binary classification sub-tasks within different subspaces. Each subtask involves classifying the new data against partial old data belonging to an old category within the subspace. This divide-and-conquer strategy reduces task complexity and mitigates the problem of Few-Shot Learning, leading to promised accuracy on both new and old categories.

The main contributions of this paper are as follows:

- **Resource-efficient Explicit Rule Generation:** We customize a novel neural network model to learn distinct mappings for each traffic category individually, with each mapping utilizing only a tiny fraction of the total feature set. For example, our approach allows a mapping to take a mere 4 bits out of a total 96-bit feature set to predict the target category. These mappings are interpreted by directly enumerating all the input-output pairs of mappings, e.g., 2^4 pairs are needed to interpret the mapping that takes 4-bit features as input. Subsequently, these input-output pairs are then converted into simple classification rules, i.e., table entries in an MA table. We also propose a sampling method to further reduce the use of entries.
- **Data-efficient Incremental Rule Updates:** Given limited data for a newly identified category label, LINC first utilizes the old rules to predict an old category label for each new sample and partitions them into subspaces, represented by the corresponding predicted categories. Although the old rules misclassify each sample as a specific category, they can rule out the possibility of belonging to another old category. Within each subspace, LINC then designs new local rules for distinguishing new samples from old samples belonging to the incorrectly predicted old categories. These new local rules are then merged with the old rules to complete the update.

We implement a prototype of LINC[1] on commercial P4 switches [1]. Based on the prototype, we conduct extensive experiments on three public classification tasks: traffic size prediction, traffic type classification, and intrusion detection. We compare the performance of LINC with four state-of-the-art methods: Mousikav2 [23], Planter [21], IIsy [18], and Netbeacon [22]. The results show that: (i) LINC achieves better performance in low-resource scenarios — accuracy is improved by 11.63% over Netbeacon (91.73% vs. 80.1%) with fewer table entries (14 vs. 22); (ii) For incremental updates, LINC greatly improves the prediction accuracy on the newly observed categories of traffic via continuous model updates — LINC achieves 26.1% higher average accuracy than traditional update method on new categories (85.6% vs. 59.5%), while using fewer table entries ($6\times\downarrow$).

[1]The code is in https://github.com/haolinyan/LINC.

II. BACKGROUND AND MOTIVATION

A. Machine Learning in Programmable Switches

Programmable switches based on the Protocol Independent Switch Architecture (PISA), allow direct programming using the P4 programming language [1]. This makes it possible to offload machine learning (ML) models from the control plane to the data plane, enabling line-rate traffic analysis. In general, ML models are deployed by defining the *Match-Action pipeline* for switches via P4. The *Match-Action pipeline* sets a serial of *Match-Action Unit* (MAU) stages, each incorporating multiple parallel *Match-Action* (MA) tables to match their target values in packet header vectors (PHVs), triggering the corresponding actions. Specifically, each MA table sets a predefined lookup key to extract the target values in PHVs and matches the values against its table entries. The standard P4 library defines three standard match kinds, i.e., exact match, ternary match, and longest prefix match. If multiple table entries match the input at the same time, the entry with the highest priority is selected. After matching is done, the corresponding actions are triggered, which support some primitive operations for model inference.

Although switches have greater bandwidth and versatility compared to Smart NICs, implementing ML in them faces more constraints that limit the type and complexity of the ML models. First, the number of MAU stages is limited (e.g., Intel Tofino 1 has only 12 MAU stages), which restricts the number of sequential operations. Second, P4 switches do not support multiplication, division, nor loops or other floating-point operations. Third, P4 switches have limited memory resources, such as SRAM and TCAM. For the Intel Tofino 1, the TCAM of each pipeline is far smaller than the SRAM (6.2MB vs. 120MB), however, current schemes consume more TCAM compared to SRAM (up to $8\times$ consumption, see Section VI-B). Fourth, switches must reserve resources for basic networking functionality or even other in-network applications. These demands further exacerbate the scarcity of switch resources. We therefore argue that it is important to develop more resource-efficient models that can operate on switches.

In parallel to these developments, there has been impressive progress made in Neural Networks (NNs) for traffic classification [28], [37]–[39]. However, the above constraints hinder their practical deployment in switches. Other ML models, such as K-means and SVMs, have been used [18], yet these consume massive resources and often exhibit poor performance [20], [25]. Thus, tree-based models, such as Decision Trees (DT) and Random Forests (RF), are commonly used instead, due to their simple inference logic and low complexity. However, as we analyze in the next subsection, these tree-based models still have to trade model complexity for resource consumption, leading to performance degradation.

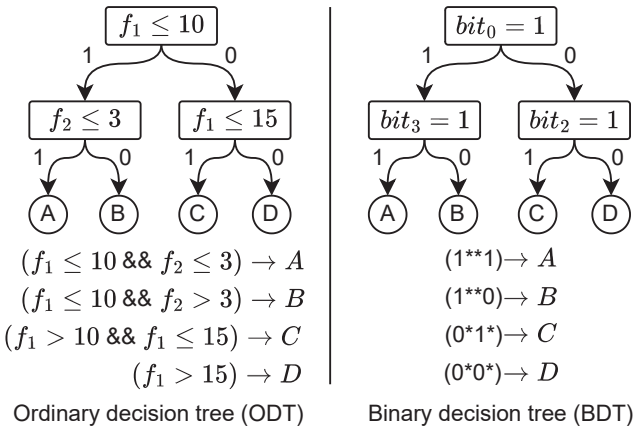

Fig. 1: The generic rule generation process of two types of tree models used in [18], [21], [22] and [23] respectively.

B. Tree-based In-network Classification

Tree-based models are more suitable for switches because their inference logic is intuitively similar to the *Match-Action pipeline*. Previous schemes [30], [31] directly implement the decision tree inference logic in switches, which consumes massive resources and limits the model complexity since the tree depth is limited by the number of MAU stages. To overcome these limitations, current schemes [18], [20], [22], [23], [40] apply encoding-based methods to enable the deployment of more complex tree models such as RF and XGBoost. These methods first convert the tree model into a set of generic rules. Fig. 1 shows two types of decision trees and the generic rules they generate. A generic rule comprises multiple conditions — this includes features and their associated thresholds that must be met concurrently to predict the corresponding category. The encoding-based methods for DT and BDT are reviewed below.

Ordinary Decision Tree (ODT). As shown in the tree on the left in Fig 1, a generic rule is constructed along a decision path, originating from the root node and terminating at a leaf node. This can be deployed in an MA table using range match. However, this method encounters several challenges: (i) The length of these rules is not always the same, i.e., some rules consist of more conditions. Consequently, the size of the lookup key must be determined by the longest rule, leading to memory inefficiency; (ii) As tree models become increasingly complex, a large number of decision path combinations can lead to an exponential increase in the number of rules.Therefore, despite this method relying on a single MA table, substantial memory resources are consumed.

To address these limitations, IIsy [18] uses multiple MA tables (refered to as feature tables) to independently encode each input feature. The output codes are then concatenated as a query to the subsequent MA table, i.e., the model table, to determine the corresponding category. Planter [21] and Netbeacon [22] further propose similar feature table encoding and model table voting techniques to deploy more complex models (e.g., Random Forest and XGBoost [41]). However, these schemes demand more switch resources as the model complexity increases [25]. For example, more MA tables are needed to encode more features or to act as model tables for more estimators in ensemble models. If the number of nodes in trees is large, this will result in significant number of table entries and the use of longer lookup keys.

Binary Decision Tree (BDT). As shown in the tree on the right in Fig. 1, the threshold in each node is represented by 1 bit. Thus, a BDT can be converted into a set of ternary rules by combining the thresholds along each decision path and masking the ignored features as "*". These generic rules can be directly deployed in an MA table using ternary match that is more efficient and widely supported across different P4 standards compared to range match. To mitigate the combinatorial explosion problem mentioned, Mousikav2 [23] uses complex models such as RF and NNs to guide a BDT to generate a simple but efficient tree structure through knowledge distillation. This manner indirectly leverages the performance of NNs while bypassing the strict limitations imposed by switches. However, knowledge distillation is not lossless [42], and it has been found that Mousikav2 may fail to transfer knowledge from NNs to BDT on some tasks [22], [24], leading to a complicated BDT that subsequently encounters the combinatorial explosion problem again.

Summary. Although encoding-based methods can deploy more complex tree models, these models can only learn simple classification patterns at each tree node and then combine them into a large number of redundant rules, resulting in a waste of resources. In a low-resource scenario, these schemes therefore have to trade-off model complexity for resources, e.g., reduce the number of features and estimators, control the model depth and maximum number of leaves. However, this results in worse classification performance. Although Mousikav2 can convert NNs into a set of rules by knowledge distillation, this indirect approach does not fully exploit the excellent pattern recognition capabilities of NNs (the generic rules are actually converted from a tree model), which inspires us to design a new method to convert the NNs directly into a set of equivalent generic rules.

C. Incrementally updatable classification

In addition to improving the model performance in a low-resource scenario, it is also crucial to consider incremental model updates [43]. In practice, once a ML model has been deployed on the switch, to evaluate its performance, it is common to periodically collect data from the data plane and submit it for manual inspection. When the model becomes ineffective due to changes in the traffic distribution or the emergence of a new traffic attacks, it is necessary to update the model, e.g., add a new category label for classification. The traditional method [18] adds the newly annotated data to the existing dataset. The model is then retrained and deployed on the switch again. However, annotating sufficient new data within a short period is difficult. Consequently, models are often retrained on imbalanced datasets, where the proportion of new data is minimal, resulting in low accuracy for the newly introduced category [35], [36]. To the best of our knowledge,

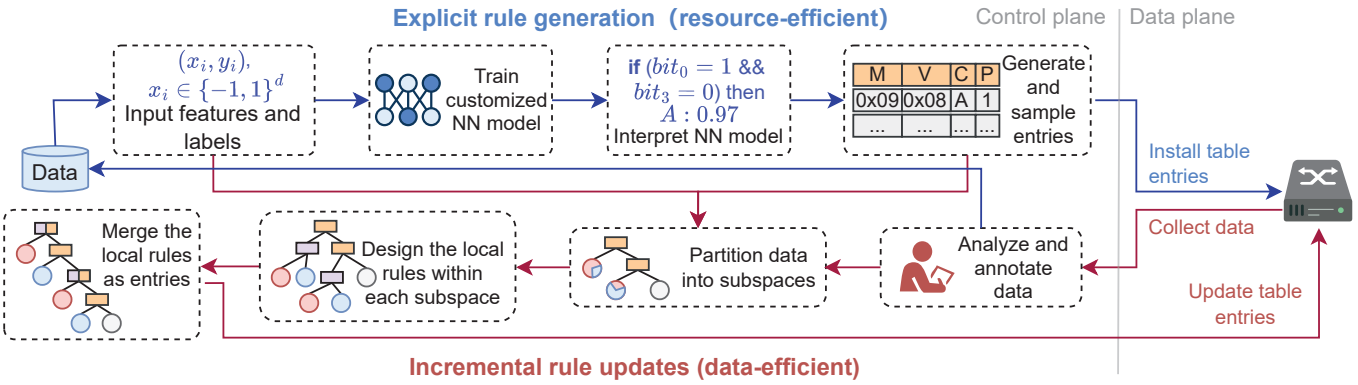

Fig. 2: Overview of LINC framework

only IIsy [18] has considered the necessity of incremental model updates. However, it does not propose an effective strategy to address the above challenge. Therefore, we propose a divide-and-conquer strategy that decomposes the update task into multiple binary classification sub-tasks. Each sub-task focuses only on distinguishing new data from partial old data, thereby balancing the dataset and reducing the task complexity.

III. OVERVIEW OF LINC

Fig. 2 illustrates the LINC framework, which comprises of two components: (i) explicit rule generation, and (ii) incremental rule updates. Initially, LINC uses explicit rule generation to achieve resource-efficient in-network classification. Here, LINC first designs a novel neural network architecture dedicated to the generation of generic rules and trains this NN model to identify traffic categories.

Based on the trained NN model, LINC interprets the NN model by deducing the key features and their binary thresholds to form generic rules for each category, e.g., the blue generic rule in Fig. 2 (detailed in Section IV-A). These generic rules are then converted into the ternary entries, followed by a sampling process to further reduce the resource usage (detailed in Section IV-B). Last, these sampled entries are installed in an MA table in the data plane.

LINC then periodically collects traffic data for expert sampling and inspection. If the traffic changes over time or a new class emerges, LINC utilizes a divide-and-conquer strategy for incremental rule updates in a data-efficient manner. First, the existing prioritized ternary entries serve as a decision tree to classify the new and old data into subspaces (detailed in Section V-A). In this tree, each prioritized entry is represented as a node, and the highest-prioritized entry is the root node. Subspaces are then represented by the tree leaves. In Fig. 2, these are visually categorized as the blue slice for old data, the red slice for new data, and the grey slice for the default category (i.e., the default action in an MA table).

LINC then utilizes the BDT to design the new local rules within each subspace, e.g., inserting the new purple rules to identify the new data. These rules are converted to ternary entries, merged with their parent entries, and given a priority one level higher than their parents, so if these new entries

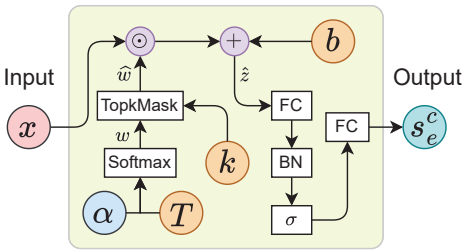

Fig. 3: The computational graph of the Neural Network Unit (NNU).

are not matched, the tree will attempt to match the next level of entries (the parent nodes). After rule updates, these entries can be used directly to update the MA table in the data plane. Section V-B presents detailed descriptions and the comparison with explicit rule generation.

IV. EXPLICIT RULE GENERATION

In this section, we present the architecture of the customized NN model and its training. We then explain how we convert the NN model into the generic rules and elaborate on the generation and sampling of table entries.

A. Model Construction and Conversion

Data Preprocessing. LINC converts packet-level features into several binary features (e.g., ip.ihl into 4 different binary features) and concatenates them into a d-bit binary vector $x_b \in \{0,1\}^d$, which is then linearly mapped to $\{-1,1\}^d$ by $x = 2x_b - 1$ for efficient training convergence [44]–[47]. The dataset is denoted as $\mathcal{D} = \{(x_i, y_i)\}_{i=0}^{N-1}$, where y_i is the label in the one-hot format and x_i is the d-bit preprocessed vector.

Customized Neural Network. Given a classification task containing C categories, we design a neural network that integrates E *Neural Network Units* (NNUs) for each category, i.e., $C \times E$ NNUs in total. Fig. 3 shows the computational graph of an NNU that aims to select k-bit features from the preprocessed d-bit features and predicts the score that indicates the category probability. Concretely, each NNU sets its specific learnable position parameter $\alpha \in \mathbb{R}^d$ to represent the importance of the corresponding features. The normalized importance weights w is obtained by

$$\mathrm{w} = \mathrm{Softmax}(\frac{\alpha}{\mathrm{T}}), \tag{1}$$

where T is a temperature coefficient used to control the distribution difference of the weights [48]. In our experiments, we find the optimal T to be between 0.5 and 0.6, using hyperparameter search. Then we define an operation named $\mathrm{TopkMask}$ that sorts the values in the vector w, keeps the top highest k bit values, and sets the remaining values to 0:

$$\hat{w} = \mathrm{TopkMask}(\mathrm{w}, k), \tag{2}$$

where $\hat{w}$ is the feature selection weight. To obtain the selected features $\hat{z}_i$, NNU first multiplies x_i by $\hat{w}$, which contains 0s to drop the features that are not selected, and then adds a learnable bias vector c to enhance the features expressiveness [46], [47]:

$$\hat{z}_i = x_i \odot \hat{w} + c. \tag{3}$$

Then, as shown in Fig. 3, each NNU has a fully connected layer to halve the feature dimension from d to $\frac{d}{2}$, as follows:

$$h_i = W_0 \hat{z}_i + b_0, \tag{4}$$

where $W_0 \in \mathbb{R}^{\frac{d}{2} \times d}$ represents the weight matrix, and b_0 is a scalar serving as the bias term. Subsequently, we incorporate BatchNorm (BN) [49] and PReLU (σ) [50] activation functions to facilitate non-linear transformations:

$$\hat{h}_i = \mathrm{BN}(h_i), \tag{5}$$

$$a_i = \sigma(\hat{h}_i). \tag{6}$$

To predict the score that indicates the category probability, an additional fully connected layer is introduced to map the vector a_i to a scalar value:

$$s_i = W_1 \hat{a_i} + b_1. \tag{7}$$

Here, $W_1 \in \mathbb{R}^{\frac{d}{2}}$ is the weight vector, and b_1 is a scalar bias. Note that each NNU of $C \times E$ NNUs has independent learnable parameters such as α, W_0, W_1, etc., and outputs a score s_i which is denoted as s_e^c in the later text.

Model Training. To optimize an NN model, it needs to predict the distribution of category probabilities $\hat{P} \in \mathbb{R}^C$ for the loss calculation. However, for a sample x_i, $C \times E$ NNUs will predict $C \times E$ category scores (E scores per category), which cannot be used directly as the category distribution $\hat{P}$. To train these NNUs simultaneously, for each category, a category score is randomly sampled from the E candidate scores based on probability as the final score for that category. Concretely, for category c, the normalized probability distribution, P_c, is calculated, and the final category score $\hat{s}_c$ is sampled according to the distribution P_c, i.e.,

$$P_c = \mathrm{Softmax}([s_c^0, s_c^1, ..., s_c^{E-1}]), \tag{8}$$

$$\hat{s}_c = \mathrm{RandomSample}(P_c, [s_c^0, s_c^1, ..., s_c^{E-1}]). \tag{9}$$

Then, the sampled final category scores of all categories are concatenated into a C dimensional logits $\hat{S} = [\hat{s}_0, \hat{s}_1, ..., \hat{s}_{C-1}]$. Last, the predicted distribution of category probabilities, $\hat{P}$, is calculated by applying the Softmax function on $\hat{S}$, i.e.,

$$\hat{P} = \mathrm{Softmax}(\hat{S}). \tag{10}$$

Standard DNN optimization techniques are then applied to optimize the model parameters, namely cross-entropy loss calculation, backpropagation, and parameter update.

Rule Derivation. The next task is to generate equivalent lightweight rules from the NN model. Concretely, LINC deduces the generic rules by analyzing the feature selection weight $\hat{w}$ of each NNU. Each weight $\hat{w}_e^c$ proposes k-bit features for identifying the category c. To obtain the corresponding thresholds, thanks to the binary features, a straightforward approach is to enumerate the combination of selected feature values, i.e., a total of 2^k threshold combinations from an NNU. These feature value combinations are then fed into the corresponding NNUs to generate the category scores.

Up to this point, we have formulated the generic rules. Each rule comprises k sets of conditions that need to be satisfied simultaneously, along with the predicted category label and its category score. In this way, we can deduce a total over $C \times E \times 2^k$ generic rules from the customized model. Compared to the ordinary neural network, the customized NN enables us to control the quantity of the generic rules by setting the hyper-parameters E and k, and ensure uniform rule lengths.

B. Entries Generation and Sampling

Once we have compiled the generic rules, e.g., if ($bit_0 = 1 \,\&\&\, bit_3 = 0$) then category A, which are in the same format as that used by the BDT, we convert them to ternary table entries (e.g., if 1**0 then category A). Additionally, the output category scores are transformed into the entry priorities. In an MA table, if several table entries match the input at the same time, the entry with the highest priority is selected.

Recall that each NNU selects only k-bit features ($k = 4$ or 8 in our experiments) that are highly relevant to the target category. For most data belonging to the same category, their values on these key features are typically the same. For example, HTTP commonly uses port 80, while HTTPS uses port 443. Similarly, the standard ports for email protocols are 25 for SMTP and 143 for IMAP. Therefore, directly enumerating and retaining all possible combinations of feature values would result in redundant table entries. Furthermore, under the assumption that the distribution of the training set is analogous to the actual data distribution (i.e., the i.i.d. assumption), machine learning theory [51], [52] ensures that the distributions of values on the key feature combinations learned by ML models should be consistent, whether in the training data or the actual data, which is particularly pronounced for binary features.

These insights inspire us to design an efficient method to sample a subset of entries. We traverse each sample in the training set, and for each sample, $C \times E$ table entries will match with it simultaneously. The entry that corresponds to the true category and has the highest priority will be sampled in the subset. We then remove duplicate entries from the subset and then install it in the data plane. From the system perspective, packets that have been classified at the data plane can be copied to the control plane and checked by matching them with the original entries. When the packet triggers an

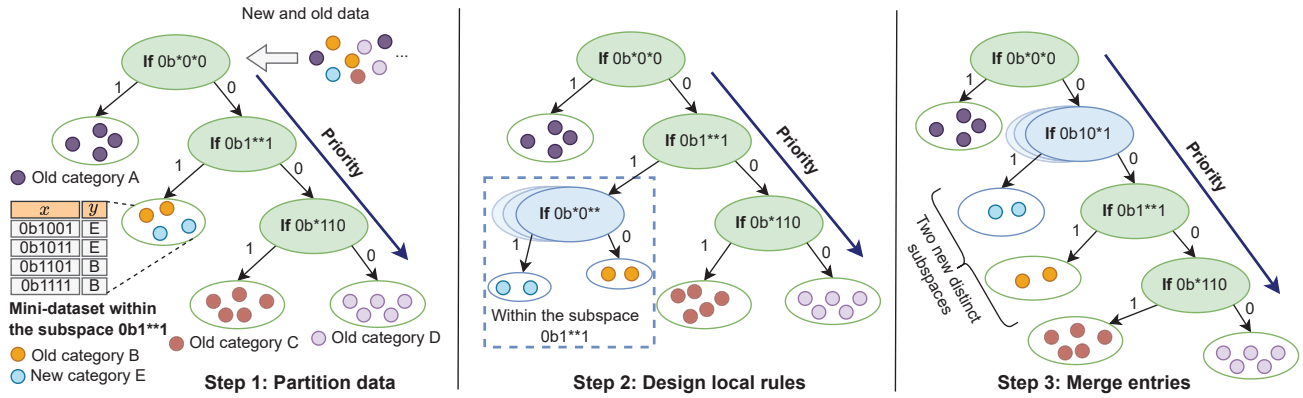

Fig. 4: The flowchart of incremental rule updates. The blue ellipses in the second tree represent multiple newly generated entries, and this figure only displays two decision branches of a blue entry for convenience.

unsampled entry, the entry can be directly installed into the data plane, ensuring the correct classification of subsequent packets in the same category. We evaluate the effectiveness of this sampling method in Section VI-F.

V. Incremental Rule Updates

As stated previously, sometimes experts will identify new changes in the traffic distribution or the emergence of novel traffic attacks from a small sample of collected data. In such cases, it is necessary to update the model. Existing approaches involve retraining the model on all data. However, this approach often fails to achieve ideal performance on the new category because of the scarcity of newly labeled data. For example, a newly identified attack pattern may have only a handful of labeled samples to train on. This section introduces a divide-and-conquer strategy that requires little new data to achieve efficient model updates.

A. Data Partitioning

The proposed strategy first uses the old model (i.e., the old entries before the update) as a decision tree to partition both the old and new data into subspaces and decomposes the entire task into several simpler sub-tasks. Concretely, for a newly identified category (e.g., a new type of attack), each update involves the introduction of a new category label to the model task. For the new samples that belong to an old category, but are misclassified due to distribution changes, LINC assigns them a new category during the update process.

Step 1 of Fig. 4 illustrates how the old entries act as a decision tree. The root node represents the entry with the highest priority. As the tree gets deeper, the priority gradually decreases until reaching the default entry. Therefore, the old entries act as a tree to classify both new and old samples into leaf nodes, where each leaf represents a subspace of an old category. Specifically, for an old sample that matches an entry, if the entry correctly classifies its category, the sample is appended to the subspace as a positive sample. For a new sample, it is added directly to the subspace it matched, as a negative sample. Up to this point, each subspace has collected a mini-dataset containing both positive and negative samples (e.g., the orange table in subspace 0b1**1 in Step 1 of Fig. 4). This step leverages the old entries to initially distinguish the new data from a fraction of the old data (e.g., distinguish the two new blue samples of category E from the categories A, C and D in Fig. 4), thereby reducing the task complexity. For both the old and new data in the same subspace, LINC automatically designs new local rules to identify the new category.

B. Automatic Local Updates

After data partitioning, LINC allows users to leverage various methods to design new local rules. For example, besides the proposed explicit rule generation, other methods, such as Mousikav2 [23], can be used to design new rules for regional binary classification tasks. Even heuristic design can be used for simple classification.

For automatic local updates, LINC utilizes the BDT for fast local updates since a BDT can also be converted as a ternary rule set in an MA table and has better performance and time efficiency than NNs in data-scarce scenarios (detailed in Section VI-D). As shown in Step 2 of Fig. 4, from low-priority to high-priority subspaces, LINC focuses on one subspace at a time (e.g., the subspace 0b1**1) and uses the BDT to fit the mini-dataset. The new generic rules generated by the BDT can distinguish the new and old data in this subspace (the new rules are displayed in the form of blue entries for convenience). In addition, the learned BDT ensures that these new generic rules will not be matched simultaneously so that they have the same priority.

The hierarchical structure of the decision tree guarantees that re-partitioning these subspaces (i.e., splitting tree nodes) will not affect each other. However, the new tree (the second tree in Fig. 4) cannot be directly converted back to the prioritized ternary entries. Therefore, LINC converts these new generic rules into ternary entries and merges them with their parent entries in Step 3 of Fig. 4. These new merged entries have one level higher than their parents, so if they are not matched, the tree will attempt to match the next level of entries (the parent nodes). For example, the blue newly generated entry 0b*0** is merged with its parent entry 0b1**1 as a new

entry 0b10*1 and given one level higher priority than 0b1**1 in the right tree in Fig. 4. After this update, the classification of old data that does not match with entry 0b1**1 will remain unaffected, ensuring that local updates do not impact the classification of other subspaces. The new and old data that match with 0b1**1 will be correctly classified by the new blue entries and the parent entry 0b1**1, respectively, ensuring the classification accuracy for the new category.

Compared to explicit rule generation, this divide-and-conquer strategy for rule updates is suitable for data-scarce scenarios due to its data-efficient nature. Instead, explicit rule generation can achieve better performance if there is enough labelled data, while consuming fewer table entries.

VI. Evaluation

This section compares the performance and resource usage of LINC against four state-of-the-art methods in low-resource scenarios. We evaluate our incremental rule update strategy and compare it to explicit rule generation to clarify the applicable scenarios for both methods in LINC. In addition, we evaluate the time efficiency of LINC and verify the performance of the entry sampling method.

A. Experimental Settings

We utilize three publicly accessible datasets: (i) Traffic Size Prediction: The UNIV1 dataset [53] is used to predict flow size, specifically distinguishing between large flows (referred to as "elephants") and small flows (referred to as "mice"); (ii) Traffic Type Classification: The ISCX Dataset [54] is used to identify different application types of traffic, such as Email, Chat, Streaming, File Transfer, VoIP, and P2P; (iii) Intrusion Detection: The UNSW-NB15 Dataset [55] is used to identify intrusion attacks, differentiating between normal and malicious traffic. Table I lists the features used in LINC. These features are pre-processed in Section IV-A before being fed into our NN model.

TABLE I: Packet-level features used in LINC.

Header	Features
IPv4	ihl, tos, length, flags, ttl, proto
TCP	dataoffset, flags, window
UDP	length

We compare LINC with four state-of-the-art methods, including IIsy [18] (DT), Planter [20] (RF), Netbeacon [22] (RF for packet-level classification), and Mousikav2 [23] (distilled BDT). All models are trained on a server with an Intel(R) Xeon(R) E5-2643 v4 @3.40GHz CPU, a Tesla M60 GPU, Python 3.9 and Pytorch 1.12.0.

In addition, we deploy the LINC framework in three commodity P4 switches (EdgeCore Widge 100BF-65X3,[2] H3C S9850-32H4,[3] and OpenMesh BF-48X6Z5.[4]) The KEYSIGHT XGS12-SDL traffic generator[5] generates test traffic at a speed of 100 Gbps. This traffic is generated in Internet Hybrid (IMIX) mode to closely resemble real-world traffic patterns.

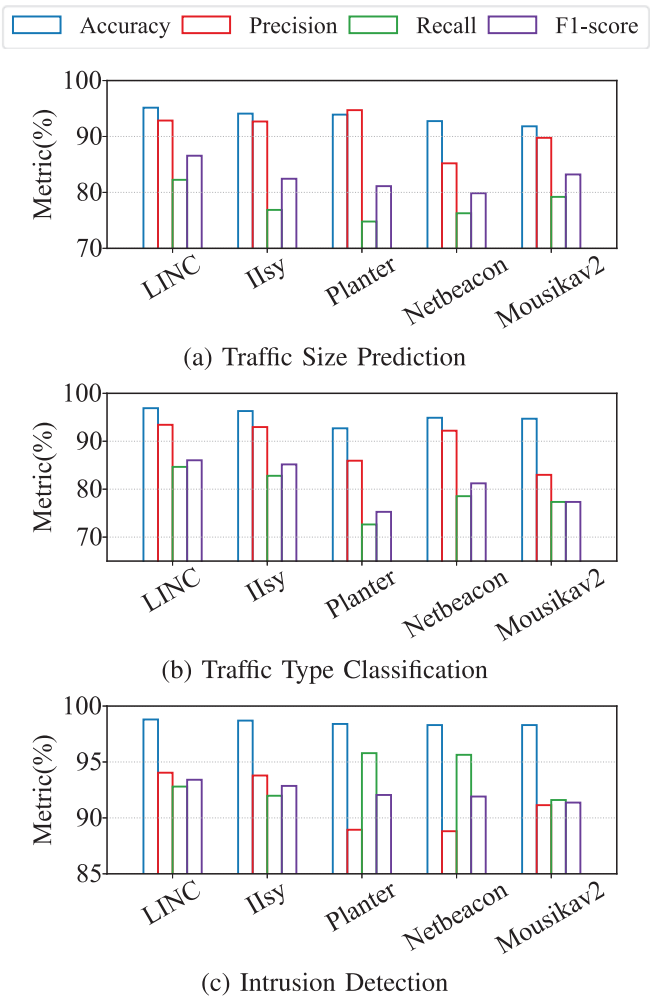

Fig. 5: The classification performance on three tasks.

B. Classification Performance and Resource Usage

We first compare the classification performance of the methods across the three tasks. As illustrated in Fig. 5, LINC outperforms other methods across the majority of considered metrics including accuracy, precision, recall, and f1-score.

As shown in Fig. 5a, on the traffic size prediction task dataset, LINC improves the recall by 7.45% compared with Planter (82.25% vs. 74.79%) and the f1-score by 6.71% compared with Netbeacon (86.56% vs. 79.85%). As shown in Fig. 5b, LINC achieves a 10.76% improvement in f1-score over Planter in the traffic type classification task (86.03% vs. 75.27%). For the intrusion detection task in Fig. 5c, LINC improves the precision by 5.1% compared with Planter (94.04% vs. 88.94%). That said, Planter and Netbeacon outperform LINC by precision and recall on a small subset of experiments. For example, compared to LINC, Planter and

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

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

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

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

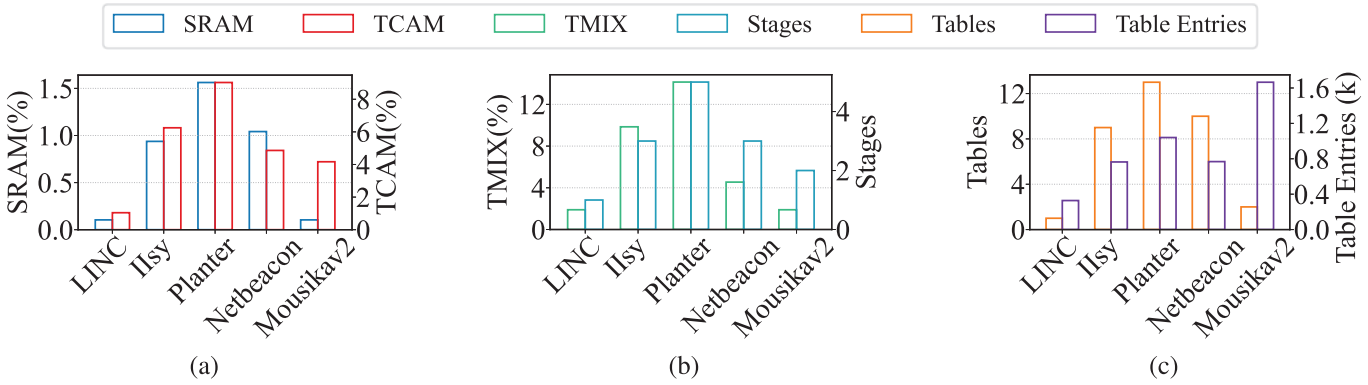

Fig. 6: The comparison of different resource usage of switches.

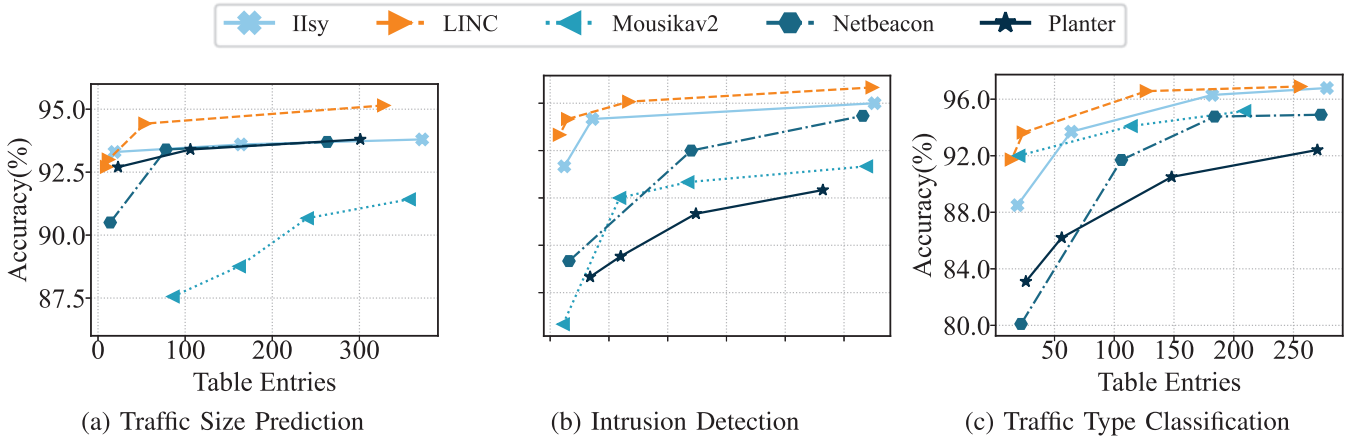

Fig. 7: The comparison of model performance under different budgets of table entries on three networking tasks.

Netbeacon improve recall in the intrusion detection task by 3.0% and 2.8% respectively. As we analyze below, they consume massive switch resources due to their complicated P4 programs.

To compare the resource consumption, we deploy P4 programs generated by these methods for the task of traffic size prediction in switches. We assess these methods in terms of memory utilization (SRAM and TCAM), Ternary Match Input Xbar (TMIX), the number of MAU stages, the count of MA tables, and the total number of table entries. As shown in Fig. 6, LINC and Mousikav2 significantly reduce the use of SRAM, TMIX, MAU stages, and MA tables. This is due to their simple P4 programs, which in fact only depend on one MA table, while others require several MA tables and MAU stages to deploy models. For example, compared to Planter, LINC reduces 15.6× of SRAM (0.10% vs. 1.56%) and 9× of TRAM (1% vs. 9%) respectively, while achieving better performance (see the Fig. 5a). In Fig. 6b, LINC reduces 5.21× of TMIX compared with IIsy (1.89% vs. 9.85%). In Fig. 6c, LINC reduces 2.34× of entries compared with Netbeacon (328 vs. 768). The only difference between LINC and Mousikav2's P4 program is that LINC removes the redundant MA table used for feature concatenation in Mousikav2. However, Mousikav2 tends to generate a complex binary decision tree when the distillation technique fails to transfer the knowledge from NNs to the BDT, leading to the combinatorial explosion problem [24], [56]. As shown in Fig. 6a and Fig. 6c, Mousikav2 generates a large number of table entries, resulting in massive TCAM consumption. LINC leverages the powerful neural network to directly learn a small number of represent-efficient classification rules, thereby reducing the number of table entries (5× less than Mousikav2) and TCAM consumption (4× less than Mousikav2).

C. Performance Under Resource Constraints

To facilitate comparison, we simulate a low-resource scenario by limiting the number of table entries. Specifically, we limit the number of entries to a maximum of 400. As the budget for the number of available entries decreases, we train models of varying sizes by controlling the hyper-parameters, including tree depth, the number of estimators for IIsy, Planter, and Netbeacon, the quantity of training data for Mousikav2, and the values of E and k for our customized model.

Fig. 7 compares the classification accuracy of these different models under given resource budgets. LINC achieves better accuracy than other methods under similar budgets, i.e., it can provide similar accuracy with much fewer table entries than other methods. For example, in the traffic type classification task in Fig. 7c, LINC achieves an 11.63% improvement in accuracy over Netbeacon (91.73% vs. 80.1%), while using a fewer number of table entries (14 vs. 22). For the traffic

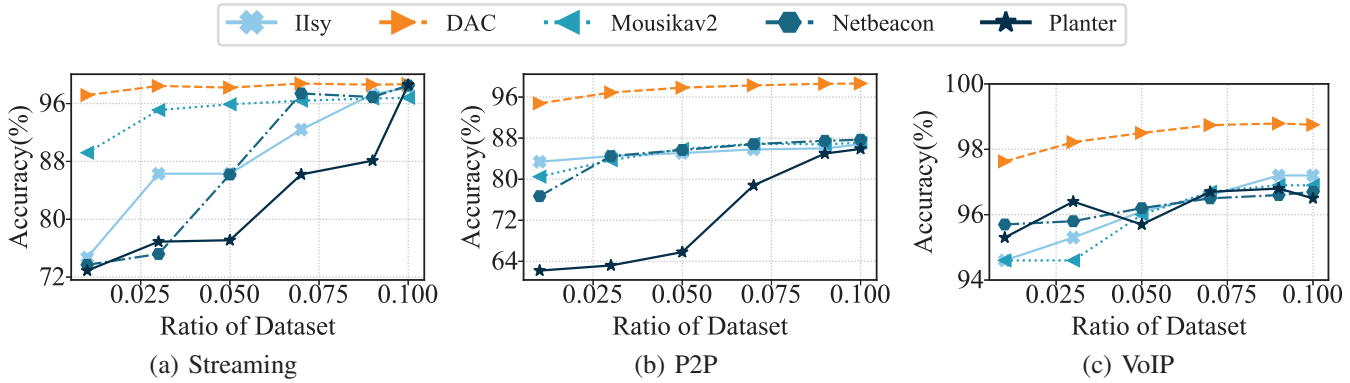

Fig. 8: Performance of updated models given different amounts of the data. We compare the accuracy of different methods during the three updates (P2P, Streaming, and VoIP respectively) by controlling the data volume.

size prediction task in Fig. 7a, LINC overtakes Mousikav2 by 5.14% (92.71% vs. 87.56%) using fewer table entries (10 vs. 86). In the intrusion detection of Fig. 7b, LINC improves the accuracy by 0.9% than Planter using fewer table entries (8 vs. 34). The results show that IIsy's performance is close to that of LINC, yet the entries produced by IIsy and Planter are employed for range match in feature tables, whereas LINC, Mousikav2, and Netbeacon are utilized for ternary match. Therefore, using the same number of entries, IIsy actually consumes more resources than LINC.

D. Incremental Model Updates

Next, we compare the performance of the proposed divide-and-conquer strategy vs. the traditional update method on newly observed categories. Here, we test them under differing volumes of new training data. Specifically, we initially train all models using the three categories of traffic type (Chat, File Transfer, and Email). To emulate the arrival of new traffic categories, we then introduce three new types, separately (Streaming, P2P, and VoIP). The baseline models are then retrained on all available datasets, including the old and new ones. LINC is updated via the **D**ivide-**A**nd-**C**onquer strategy, abbreviated as *DAC* for the sake of brevity.

Fig. 8 reveals that the traditional method exhibits limited accuracy when the dataset is small, i.e., when it faces the Few-Shot Learning problem. Compared to the traditional method, *DAC* is able to update models to acceptable accuracy (over 95%) in three updates, even with very little data. For example, in the task of identifying Streaming traffic, *DAC* improves the accuracy by 1.89% compared with Mousikav2 (98.69% vs. 96.8%) when using 10% of the dataset, and improves the accuracy by 24.27% compared with Planter (97.17% vs. 72.9%) when using 1% of the dataset. This constitutes a significant improvement.

To assess the efficiency of incremental rule updates in LINC, we update the models three times consecutively, incorporating each of the three categories (Streaming, P2P, and VoIP) into training one at a time. Here, we use a restricted quantity of fresh data (comprising 0.05% of new data, which included 63 samples for Streaming, 44 for P2P, and 28 for VoIP). For comparison between **E**xplicit **R**ule **G**eneration (*ERG*) and *DAC*, *ERG* retrains the NN model using both new and old data and converts it into updated table entries.

Table II presents the accuracy of the models during the three consecutive updates of new categories (Acc (n)), old categories (Acc (o)), and all categories collectively. The consumption of table entries is also reported during each update. The results show that *DAC* has superior classification accuracy across all categories when the model is updated sequentially. Further, it consumes fewer table entries. Although Mousikav2 and Planter exhibit superior accuracy for the old categories during the first update (just a 0.5% improvement), as the number of updates increases, these methods lag behind LINC on all metrics. For example, compared to Mousikav2, *DAC* improves the average accuracy for all categories by 6.7% (94.6% vs. 87.9%), by 26.1% for new categories (85.6% vs. 59.5%), and by 4.1% for old categories (95.6% vs. 91.5%), while reducing 6× the number of table entries, on average (104 vs. 643). *ERG* exhibits inferior accuracy when the dataset is small, as neural networks, compared to tree-based models, are more dependent on a large amount of training data.

To compare *ERG* and *DAC*, we use all the new data for the three consecutive model updates. Recall that *DAC* utilizes the BDT for local rule updates, which can lead to significant consumption of entries when dealing with large volumes of data. As shown in Table III, *ERG* outperforms *DAC* and baseline methods, achieving better average performance (96.9% vs. 81.7%) and fewer entries (162 vs. 462). Therefore, in LINC, *ERG* is suitable when a sufficient amount of data is available. However, to quickly adapt to changes in traffic distribution with a small amount of training data, it is superior to use *DAC* for efficient incremental rule updates.

Note, the baseline models also consume more entries in Table II than in Table III. This is because these tree-based models require more complex trees to overfit the small set of new data items for acceptable accuracy, thereby consuming a large number of table entries.

E. Time Efficiency of LINC

First, we evaluate the time efficiency of LINC's training and update pipeline. The initial NN model is trained on a Tesla M60 GPU using 908,469 samples for 32 epochs in

TABLE II: The comparison of models' performance after three consecutive incremental category updates using partially new data.

Methods	Streaming				P2P				VoIP				Average			
	Acc	Acc(o)	Acc(n)	Entries	Acc	Acc(o)	Acc(n)	Entries	Acc	Acc(o)	Acc(n)	Entries	Acc	Acc(o)	Acc(n)	Entries
ERG	85.6	97.4	0.0	116	78.9	85.5	0.0	**81**	74.9	78.6	0.0	82	79.8	87.2	0.0	**93**
DAC	**95.2**	97.5	**78.2**	**96**	**94.3**	**95.1**	**84.5**	106	**94.2**	**94.2**	**94.1**	**109**	**94.6**	**95.6**	**85.6**	104
IIsy	90.3	97.3	39.1	171	88.7	90.9	62.9	509	89.1	88.8	94.1	643	89.4	92.3	65.4	441
Planter	90.8	**98.0**	39.0	1417	88.6	90.8	62.4	1402	88.8	88.5	94.1	1703	89.4	92.4	65.2	1507
Netbeacon	90.5	97.7	39.0	808	88.3	90.5	62.2	781	88.4	88.4	89.5	432	89.1	92.2	63.6	674
Mousikav2	89.5	**98.0**	28.6	808	87.3	89.5	60.5	643	87.0	86.9	89.5	479	87.9	91.5	59.5	643

TABLE III: The comparison of models' performance after three consecutive incremental category updates using all new data.

Methods	Streaming				P2P				VoIP				Average			
	Acc	Acc(o)	Acc(n)	Entries	Acc	Acc(o)	Acc(n)	Entries	Acc	Acc(o)	Acc(n)	Entries	Acc	Acc(o)	Acc(n)	Entries
ERG	**97.5**	97.2	**99.5**	**120**	**96.7**	97.0	93.3	**179**	**96.5**	**96.6**	94.2	**187**	**96.9**	**96.9**	95.7	**162**
DAC	**97.5**	**97.3**	99.0	200	96.4	96.2	**99.6**	471	51.3	49.0	**99.1**	715	81.7	80.8	**99.2**	462
IIsy	97.2	96.9	99.4	191	96.3	**97.1**	87.1	197	96.3	96.4	94.8	183	96.6	96.8	93.8	190
Planter	95.4	94.8	99.4	226	93.0	95.3	65.5	289	92.4	91.1	94.7	284	93.6	93.7	86.5	266
Netbeacon	96.6	96.3	99.3	269	88.4	88.4	89.5	432	93.5	93.6	91.0	242	92.8	92.8	93.3	314
Mousikav2	96.6	96.3	99.3	185	95.2	95.0	96.8	206	95.2	95.1	96.9	209	95.7	95.5	97.7	200

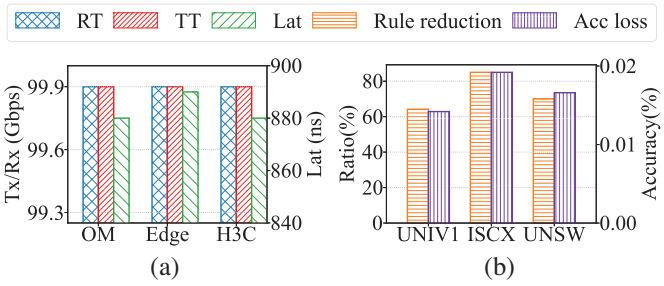

Fig. 9: (a) Throughput and latency for LINC across three switches. (b) Impact of rule sampling method on rule reduction and accuracy loss.

137 seconds. It then takes 4 seconds to convert the model to rules. In three consecutive rule updates, the average time cost is 14.31 seconds when using 0.05% of the new data, and 36.46 seconds when using all the new data. LINC's time cost is minimal versus manual data labelling. It only needs little fresh data for quick, efficient model updates.

To measure the runtime throughput and latency, we deploy LINC on three commodity switches and measure the performance under a traffic rate of 100 Gbps. As depicted in Fig 9a, both the received throughput (RT) and transmitted throughput (TT) can reach 99.9 Gbps, and the average latency of packet processing is below 890 nanoseconds, indicating that LINC achieves line-speed inference on real network devices.

F. Efficiency of Entry Sampling Method

Finally, we investigate the average proportion of redundant entries reduced by the entry sampling method proposed in Section IV-B and measure the resulting loss of accuracy. Fig. 9b shows the results on three tasks, represented by the names of the datasets (UNIV1, ISCX, and UNSW) for convenience. Our method significantly reduces the redundant entries by up to 85%, while maintaining minimal impact on the model's classification accuracy (less than a 0.019% decrease).

VII. CONCLUSION

In this paper, we have proposed LINC, a novel framework for low-resource in-network classification and efficient incremental model updates. Its goal is to improve in-network model performance under the low-resource scenarios. For this, we tailor a neural network for classification and for learning the representative mapping between input features and output categories. We indirectly deploy this NN model in the data plane by deducing generic rules from it, and utilize an efficient entry sampling method to reduce the redundant entries. For incremental model updates, we propose a novel divide-and-conquer strategy that decomposes the entire task into multiple regional binary classification tasks and updates these tasks respectively, thus improving the update performance.

Our experimental results show that LINC reduces resource consumption while improving the model performance. LINC reduces the use of various switch resources, including the table entries (5× ↓), SRAM (15.6× ↓), TCAM (8.7× ↓) and TMIX (7.5× ↓), while improving the accuracy (11.63% ↑). For incremental model updates, LINC enhances the average accuracy of new categories by 26.1% ↑, despite using fewer table entries (6× ↓).

VIII. ACKNOWLEDGMENT

We thank our shepherd Prof. Xiaoqi Chen and anonymous reviewers. This work is supported by the Major Key Project of PCL under grant No. PCL2023A06, the National Key Research and Development Program of China under grant No. 2022YFB3105000, and the Shenzhen Key Lab of Software Defined Networking under grant No. ZDSYS20140509172959989.

REFERENCES

[1] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, p. 87–95, jul 2014. [Online]. Available: https://doi.org/10.1145/2656877.2656890

[2] R. MacDavid, C. Cascone, P. Lin, B. Padmanabhan, A. ThakuR, L. Peterson, J. Rexford, and O. Sunay, "A p4-based 5g user plane function," in *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*, ser. SOSR '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 162–168. [Online]. Available: https://doi.org/10.1145/3482898.3483358

[3] J. Haxhibeqiri, P. H. Isolani, J. M. Marquez-Barja, I. Moerman, and J. Hoebeke, "In-band network monitoring technique to support sdn-based wireless networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 627–641, 2021.

[4] R. Ben Basat, X. Chen, G. Einziger, and O. Rottenstreich, "Designing heavy-hitter detection algorithms for programmable switches," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1172–1185, 2020.

[5] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 121–136. [Online]. Available: https://doi.org/10.1145/3132747.3132764

[6] M. Liu, L. Luo, J. Nelson, L. Ceze, A. Krishnamurthy, and K. Atreya, "Incbricks: Toward in-network computation with an in-network cache," *SIGARCH Comput. Archit. News*, vol. 45, no. 1, p. 795–809, apr 2017. [Online]. Available: https://doi.org/10.1145/3093337.3037731

[7] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and S. Pack, "Wave: Popularity-based and collaborative in-network caching for content-oriented networks," in *2012 Proceedings IEEE INFOCOM Workshops*, 2012, pp. 316–321.

[8] X. Zuo, Q. Li, J. Xiao, D. Zhao, and J. Yong, "Drift-bottle: a lightweight and distributed approach to failure localization in general networks," in *ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT), Roma, Italy, December 6-9, 2022*. ACM, 2022, pp. 337–348.

[9] Q. Li, J. Xiao, D. Zhao, X. Zuo, W. Tang, and Y. Jiang, "Themis: A passive-active hybrid framework with in-network intelligence for lightweight failure localization," *Available at SSRN 4604412*, 2023.

[10] L. Ye, Q. Li, X. Zuo, J. Xiao, Y. Jiang, Z. Qi, and C. Zhu, "PUFF: A passive and universal learning-based framework for intra-domain failure detection," in *IEEE International Performance, Computing, and Communications Conference (IPCCC), Austin, TX, USA, October 29-31, 2021*. IEEE, 2021, pp. 1–8.

[11] C. Miao, M. Chen, A. Gupta, Z. Meng, L. Ye, J. Xiao, J. Chen, Z. He, X. Luo, J. Wang, and H. Yu, "Detecting ephemeral optical events with optel," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI), Renton, WA, USA, April 4-6, 2022*. USENIX Association, 2022, pp. 339–353.

[12] H. T. Dang, P. Bressana, H. Wang, K. S. Lee, N. Zilberman, H. Weatherspoon, M. Canini, F. Pedone, and R. Soulé, "P4xos: Consensus as a network service," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1726–1738, 2020.

[13] J. Li, E. Michael, and D. R. K. Ports, "Eris: Coordination-free consistent transactions using in-network concurrency control," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 104–120. [Online]. Available: https://doi.org/10.1145/3132747.3132751

[14] J. Li, J. Nelson, E. Michael, X. Jin, and D. R. K. Ports, "Pegasus: Tolerating skewed workloads in distributed storage with in-network coherence directories," in *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'20. USA: USENIX Association, 2020.

[15] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2890955.2890968

[16] J. Xiao, X. Zuo, Q. Li, D. Zhao, H. Zhao, Y. Jiang, J. Sun, B. Chen, Y. Liang, and J. Li, "Flexnf: Flexible network function orchestration for scalable on-path service chain serving," *IEEE/ACM Transactions on Networking (ToN)*, 2023.

[17] C. H. Benet, A. J. Kassler, T. Benson, and G. Pongracz, "Mp-hula: Multipath transport aware load balancing using programmable data planes," in *Proceedings of the 2018 Morning Workshop on In-Network Computing*, ser. NetCompute '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 7–13. [Online]. Available: https://doi.org/10.1145/3229591.3229596

[18] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, ser. HotNets '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 25–33. [Online]. Available: https://doi.org/10.1145/3365609.3365864

[19] Z. Liu, H. Jin, Y.-C. Hu, and M. Bailey, "Practical proactive ddos-attack mitigation via endpoint-driven in-network traffic control," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1948–1961, 2018.

[20] C. Zheng, M. Zang, X. Hong, R. Bensoussane, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "Automating in-network machine learning," 2022.

[21] C. Zheng and N. Zilberman, "Planter: seeding trees within switches," in *Proceedings of the SIGCOMM '21 Poster and Demo Sessions*, ser. SIGCOMM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 12–14. [Online]. Available: https://doi.org/10.1145/3472716.3472846

[22] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 6203–6220. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/zhou-guangmeng

[23] G. Xie, Q. Li, G. Duan, J. Lin, Y. Dong, Y. Jiang, D. Zhao, and Y. Yang, "Empowering in-network classification in programmable switches by binary decision tree and knowledge distillation," *IEEE/ACM Transactions on Networking*, vol. 32, no. 1, pp. 382–395, 2024.

[24] A. T.-J. Akem, B. Bütün, M. Gucciardo, and M. Fiore, "Henna: hierarchical machine learning inference in programmable switches," in *Proceedings of the 1st International Workshop on Native Network Intelligence*, ser. NativeNi '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1–7. [Online]. Available: https://doi.org/10.1145/3565009.3569520

[25] A. T.-J. Akem, M. Gucciardo, and M. Fiore, "Flowrest: Practical flow-level inference in programmable switches with random forests," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, 2023, pp. 1–10.

[26] C. Fu, Q. Li, M. Shen, and K. Xu, "Realtime robust malicious traffic detection via frequency domain analysis," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 3431–3446. [Online]. Available: https://doi.org/10.1145/3460120.3484585

[27] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," 2018.

[28] 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 *2023 IEEE 31st International Conference on Network Protocols (ICNP)*, 2023, pp. 1–12.

[29] Barefoot Networks, "Tofino switch," 2022. [Online]. Available: https://www.barefootnetworks.com/products/brief-tofino/

[30] J.-H. Lee and K. Singh, "Switchtree: in-network computing and traffic analyses with random forests," *Neural Computing and Applications*, 2020. [Online]. Available: https://doi.org/10.1007/s00521-020-05440-2

[31] C. Busse-Grawitz, R. Meier, A. Dietmüller, T. Bühler, and L. Vanbever, "pforest: In-network inference with random forests," 2022.

[32] I. Lee, H. Roh, and W. Lee, "Poster abstract: Encrypted malware traffic detection using incremental learning," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2020, pp. 1348–1349.

[33] Y. Chen, T. Zang, Y. Zhang, Y. Zhou, L. Ouyang, and P. Yang, "Incremental learning for mobile encrypted traffic classification," in *ICC 2021 - IEEE International Conference on Communications*, 2021, pp. 1–6.

[34] G. Bovenzi, L. Yang, A. Finamore, G. Aceto, D. Ciuonzo, A. Pescapè, and D. Rossi, "A first look at class incremental learning in deep learning mobile traffic classification," 2021.

[35] S. Blaes and T. Burwick, "Few-shot learning in deep networks through global prototyping," *Neural Networks*, vol. 94, pp. 159–172, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608017301533

[36] Y. Huang, F. Chang, Y. Tao, Y. Zhao, L. Ma, and H. Su, "Few-shot learning based on attn-cutmix and task-adaptive transformer for the recognition of cotton growth state," *Comput. Electron.*

Agric., vol. 202, no. C, nov 2022. [Online]. Available: https://doi.org/10.1016/j.compag.2022.107406

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

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

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

[40] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, ser. HotNets '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 25–33. [Online]. Available: https://doi.org/10.1145/3365609.3365864

[41] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 785–794. [Online]. Available: https://doi.org/10.1145/2939672.2939785

[42] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

[43] D. Han, Z. Wang, W. Chen, K. Wang, R. Yu, S. Wang, H. Zhang, Z. Wang, M. Jin, J. Yang *et al.*, "Anomaly detection in the open world: Normality shift detection, explanation, and adaptation," in *30th Annual Network and Distributed System Security Symposium (NDSS)*, 2023.

[44] G. Siracusano and R. Bifulco, "In-network neural networks," 2018.

[45] G. Siracusano, S. Galea, D. Sanvito, M. Malekzadeh, H. Haddadi, G. Antichi, and R. Bifulco, "Running neural networks on the nic," 2020.

[46] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 525–542.

[47] N. Guo, J. Bethge, C. Meinel, and H. Yang, "Join the high accuracy club on imagenet with a binary neural network ticket," 2022.

[48] F. Wang and H. Liu, "Understanding the behaviour of contrastive loss," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 2495–2504.

[49] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 448–456.

[50] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ser. ICCV '15. USA: IEEE Computer Society, 2015, p. 1026–1034. [Online]. Available: https://doi.org/10.1109/ICCV.2015.123

[51] N. Courty, R. Flamary, A. Habrard, and A. Rakotomamonjy, "Joint distribution optimal transportation for domain adaptation," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 3733–3742.

[52] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 1180–1189.

[53] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 267–280. [Online]. Available: https://doi.org/10.1145/1879141.1879175

[54] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related," in *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, 2016, pp. 407–414.

[55] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, 2015, pp. 1–6.

[56] C. Kim, A. Sivaraman, N. P. K. Katta, A. Bas, A. A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:15782087