Resource-Efficient Implementation of Multiple Concurrent Tree-Based Models in P4 Switches using Feature Sharing

Oleg Karandin[1*], Aleix Lahoz Torres[2], Nicola Di Cicco[1], Francesco Musumeci[1], Massimo Tornatore[1]
[1]*Department of Electronics, Information and Bioengineering, Politecnico di Milano, Milan, Italy*
[2]*Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona, Universitat Politècnica de Catalunya, Spain*
[*]Corresponding author: oleg.karandin@polimi.it

***Abstract*—Machine Learning (ML) models have found numerous applications in the automation of complex network management tasks. More recently, thanks to the introduction of new solutions for data-plane programmability (as the P4 programming language), it has become possible for programmable switches to execute ML-models directly in the data plane, with the great advantage that decisions can now be taken at packet-rate, without the involvement of the control plane. Existing works have shown that tree-based ML models, such as Random Forest, can be implemented on P4 switches, despite strict constraints on the available computational and memory resources. However, the (common, and practical) case when multiple models must be concurrently implemented to perform different tasks is still under-investigated. Assigning separate, dedicated resources (i.e., stages in the packet-processing pipeline) to each model can be very inefficient. In this study we propose a new resource-efficient data-plane implementation of multiple concurrent tree-models that share input features. We focus on the problems of DDoS-attack detection and application-traffic identification and demonstrate high accuracy in both problems while saving up to 40% of the required processing stages.**

***Index Terms*—Programmable switch, P4, In-network ML, Random Forest**

I. Introduction

Communication networks are quickly evolving to support new services with demanding bandwidth, latency and security requirements. This evolution largely capitalizes on novel technologies for network automation that leverage monitoring data to effectively re-implement traditional management tasks, such as network resource allocation and network security that have been traditionally performed via heuristic algorithms, based on rigid rules, hand-crafted for a specific problem based on the domain knowledge of network experts. The recent growth in availability of network monitoring data and of in-network computational resources has paved the way for the deployment of Machine Learning (ML)-based solutions for automated network management that can more flexibly adapt to network changes as data is collected and processed.

As of today, ML models are typically trained and executed in the control plane. This means that, as shown in Fig. 1a, the switch collects the telemetry data describing, e.g, a packet/flow behavior, in the data plane and then sends the data to the controller. In turn, the controller performs the inference using a pre-trained ML-model and returns the decision (e.g., to forward or drop the packet) back to the switch. However, round-trip between the switch and the controller (and possibly further delays due to resource contention at the controller) limit the reaction speed, which is a critical parameter in modern switches with multi-Tb/s throughputs.

To achieve a fully data-driven network management, ML-inference must be performed at packet-rate, directly in the data plane (see Fig. 1b). This has been recently enabled by the programmable switches that allow to define custom packet processing logic and that can implement simple ML models [1]–[7]. However, implementation of ML models in the data plane is non-trivial as packet-rate processing imposes low-delay requirement that dramatically limit the type and the number of operations that can be performed on each packet.

P4 (Programming Protocol-independent Packet Processors) [8], [9] is a platform-independent data-plane programming language that allows to run the same program on hardware and virtual switches and on smartNICs. P4 supports integer additions and bit-shifts and match-action (M/A) operations (i.e., *M/A Table [Key]* $->$ *call Action*, where *Action* is a simple function). Packets in the programmable switch pass through a pipeline of processing stages (known as M/A stages), each

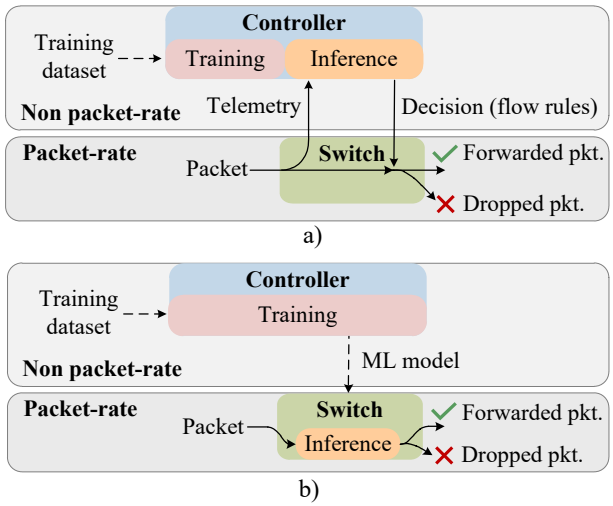

Fig. 1. a) ML-model is trained and executed at the controller. b) ML-model is trained at the controller and executed at the switch.

978-3-903176-66-9 ©2024 IFIP

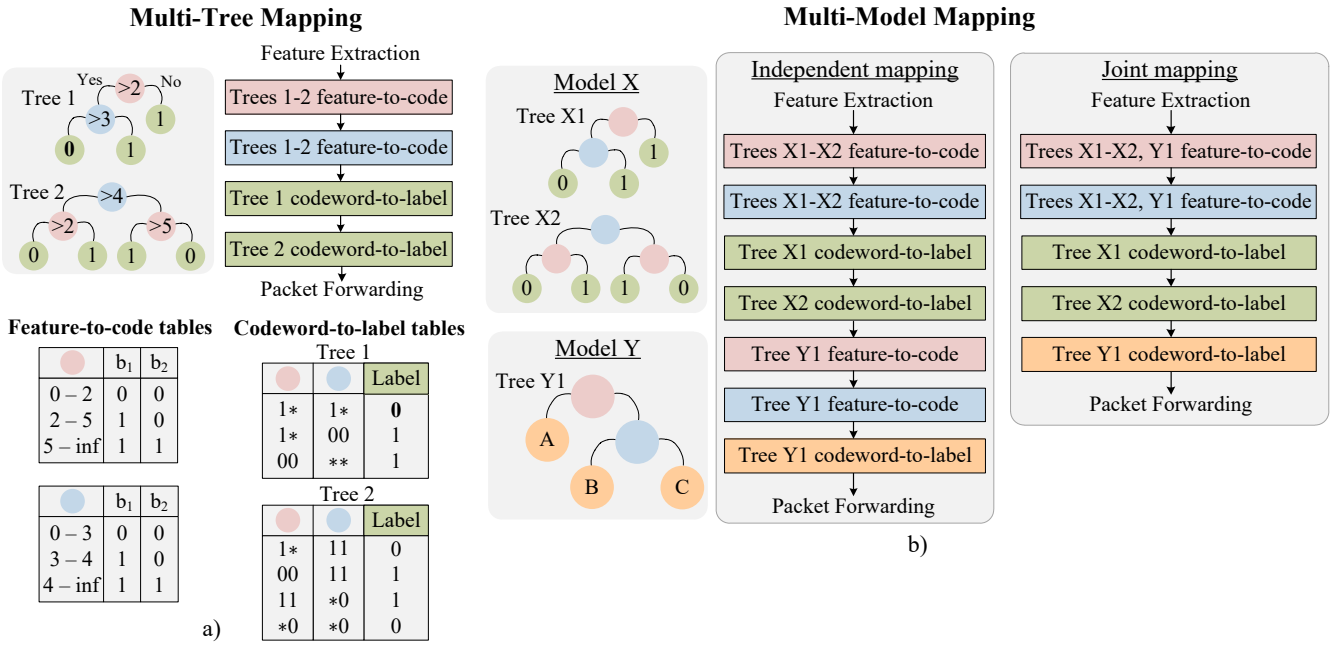

Fig. 2. Feature-based mapping of a) Single tree-based model and b) Two tree-based models with independent or joint feature-to-code mapping

equipped with memory to store M/A tables and Arithmetic Logical Units (ALUs) used to execute the corresponding Actions. The previously-mentioned delay constraint limits the number of M/A stages, while each stage also has a limited amount of memory and ALUs.

Tree-based ML-models, such as Random Forest (RF), are considered as the most suitable models for predictions on tabular data [10], while they also have a simple structure and can satisfy P4-constraints without any modifications to training and inference processes. However, constraints on the computational and memory resources pose a limit on the number of trees in the forests and tree depth, as well as the number of input features, therefore affecting the potential of RFs in terms of prediction accuracy and generalization capability. For these reasons, research efforts on more efficient tree-model implementations are needed. For example, instead of traversing the decision tree layer by layer, the recently proposed *feature-based* approach [3] (described in details in Section II) first maps the values of input features to the corresponding region of the feature space and then determines which label was assigned to that region during model-training. With this approach RFs can be implemented with a number of M/A stages that is dependent on the number of features and the number of trees, regardless the tree depth.

Moreover, to automate multiple network management tasks, in general, multiple ML models must be concurrently executed at the data plane. For example, network-operators could largely benefit from the concurrent implementation of attack detection and application-traffic identification. In this context, given the aforementioned memory and computational constraints, it would be desirable to share some operations, such as computing features and/or performing inference, across model performing the different tasks, instead of implementing each model independently, i.e., using dedicated computational resources. In this work, for the first time in literature to the best of our knowledge, we investigate the implementation of multiple concurrent ML models at the data plane. We extend the feature-based implementation to multiple tree-based models that share input features and map the values of input features to the corresponding region in the feature space simultaneously for multiple models and than determine the label that was assigned to that region by each individual model. It is realistic to assume that models can use the same features, as only a limited number of features is available on the switch, e.g., the statistics of the number of packets and inter-arrival time. We demonstrate that by using the proposed approach concurrent on-switch DDoS-detection and application-traffic identification can be performed with sufficient accuracy while using up to 40% fewer M/A stages (hence, with lower delay).

The rest of the paper is organized as follows. Section II describes state-of-the-art of how tree-based models are implemented using P4. Section III describes the proposed approach for a resource-efficient implementation of multiple concurrent tree-based models in P4. In Section IV we numerically evaluate the number of M/A stages used to implement concurrent on-switch DDoS-detection and application-traffic identification and the accuracy achieved by both tasks.

II. Background and Related Works

A. Related Works

Many recent works describe the implementation of supervised [5] and unsupervised [6] ML models in programmable switches [1]. The structure of the tree-based models makes them the easiest for implementing in P4, and authors in [7] even propose to use knowledge distillation and implement more complex models as simple binary decision trees.

Tree-based models (e.g., RFs) are supervised ML algorithms with a tree-like structure that can be used for both classification and regression [11]. Leaf nodes correspond to the decisions (e.g., class labels), while internal nodes correspond to the decision rules, where each rule compares the value of one feature with a threshold that was optimized during training. Inference for a data point starts at the root and proceeds down the tree, branching left or right at each layer depending on the feature value, till it arrives at one leaf, where the data point is associated with the label.

Two main approaches to implement tree-based models in P4 switches have been proposed in literature: depth-based [2] and feature-based mapping [3], while authors in [4] propose a hybrid approach. *Depth-based mapping* hierarchically encodes each layer of each tree into the corresponding M/A table. The P4 program then sequentially traverses the tables, till it arrives to the layer of each tree that holds a class label. A constraint on the number of M/A stages limits the total number of layers (depth) of the trees. *Feature-based mapping* assigns a binary code to every feature interval created by the thresholds used across all the trees. One bit of the encoded feature defines if it is smaller or greater than the corresponding threshold-value. Concatenated feature encodings generate a *codeword* that describes a region of the feature-space and hence corresponds to an output class.

B. Feature-Based Encoding

In Fig. 2a we show an example of the feature-based implementation of a model with 2 trees and 2 features (red and blue). The P4 program first uses feature-to-code M/A tables to assign a code to the value of each input feature. Each feature is processed exactly once even if used across multiple layers of different trees. Finally, the program uses a codeword-to-label M/A table to associate a codeword with the label in each tree.

In the example of Fig. 2a, each feature is used with two thresholds across the two trees: red feature with thresholds 2 and 5, and blue feature with thresholds 3 and 4, thus both features are encoded with 2 bits. For example, for the red feature the first bit is 0 (respectively, 1) for values smaller (larger) than 2, while the second bit is 0 (respectively, 1) for values smaller (larger) than 5. The codeword-to-label M/A table uses ternary encoding (i.e., with an $*$ used to match both 0 and 1) and contains one entry for each tree leaf. For example, the leftmost leaf with label 0 in Tree 1 is reached if the red feature is larger than 2, and the blue feature is larger than 3, thus the first bits in the encoding of each feature must be equal to 1, while the second bits that correspond to the other two thresholds are irrelevant (i.e., can be 0 or 1), and are thus represented with an $*$, and the final codeword obtained through the encoding process would be $1*1*$.

Assuming that one M/A stage is used for one feature-to-code or codeword-to-label mapping, the number of required M/A stages is the sum of the number of features and trees, independent from the number of layers in each tree. However, the size of the tables increases with tree depth, so it is limited by the constraint on the memory available per M/A stage.

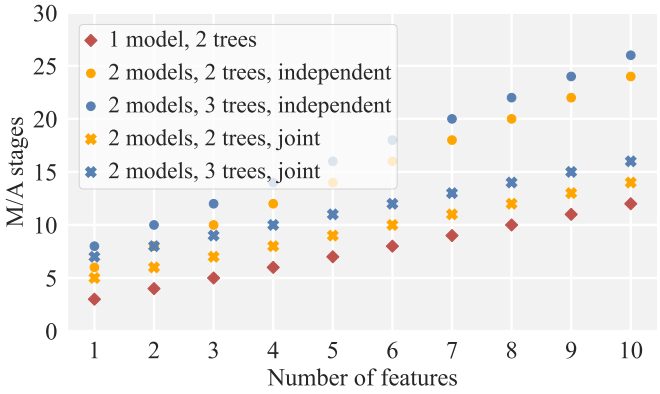

Fig. 3. M/A stages used for feature-to-code and codeword-to-label mapping for different numbers of trees, models and features, with joint and independent feature-mapping

III. Feature-Sharing for Resource-Efficient Implementation of Multiple Tree-Based Models

A. Feature-based mapping of multiple tree-based models

Feature-based mapping allows to process each feature once not only among different trees of the same model (as shown in Fig. 2a), but also among different trees of different models, as long as they use the same features. In Fig. 2b we show two classifiers X (with 2 trees) and Y (with 1 tree), that use the same 2 features and solve a binary (0/1) and a ternary (A/B/C) classification problem, respectively. If mapped independently, the two models will be executed in 7 M/A stages, while if feature-to-code mapping is performed jointly, only 5 M/A stages are needed. As a result, we can either reduce the inference delay, or we can leverage the saved M/A stages to improve prediction accuracy (e.g., by using more features or trees) or perform other operations in the switch. Using the same features in different models is a realistic assumption, as only a limited number of features is available on the switch, mainly the statistics of the number of packets and inter-arrival time. Note that the different models can share only a subset of features, e.g., if certain features are more important for one specific model than for the others.

Assuming that models share all features, in Fig. 3 we show how many M/A stages are used by models in different configurations using the following formulas:

$$N_{stages}^{Independent} = N_{models} \times (N_{trees} + N_{features}) \quad (1)$$

$$N_{stages}^{Joint} = N_{models} \times N_{trees} + N_{features} \quad (2)$$

The number of M/A stages required by a single model (red diamonds) grows linearly with the number of features. Implementing the second model independently requires a large number of extra M/A stages (orange and blue circles), while significantly fewer M/A stages are needed with a joint implementation (orange and blue squares).

In this work we assume that one M/A stage is used to store one M/A table, independently of the table size. In practice, based on the amount of memory per M/A stage, multiple small tables can be stored in a single M/A stage, while large tables are stored across multiple M/A stages. As sharing of feature-mapping M/A stages between multiple models will increase

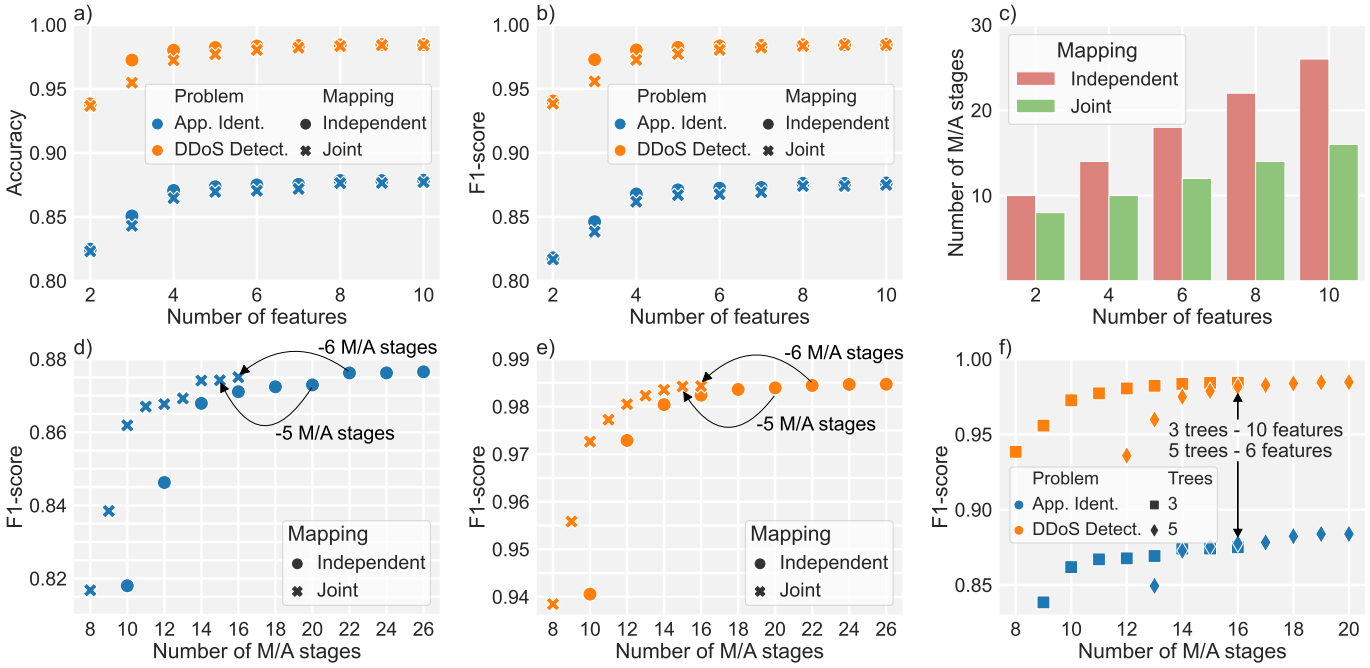

Fig. 4. a) Accuracy, b) F1-score and c) number of M/A stages vs. number of features for independent and joint feature-mapping and for Application Identification and DDoS Detection (3 trees per model); d-e) F1-score for Application Identification (d), and DDoS Detection (e) vs. number of M/A stages for independent and joint feature-mapping (3 trees per model); f) F1-score vs. the number of M/A stages with 3 and 5 trees per model and joint feature-mapping.

the size of both feature-to-code and codeword-to-label M/A tables, we must ensure that M/A stages that we save with joint feature-mapping do not get consumed by larger tables.

Algorithm 1 Joint feature-selection algorithm

Require: F: full set of features, N: desired № of features, $Data_i$: dataset for problem i, i = 1..M

while $|F| > N$ **do**
  for i = 1 ... M **do**
    $Model_i = train(Data_i^{train})$
    $I_i = PFI(Model_i, Data_i^{validation})$
  end for
  $I = CombineImportance(I_1, ..., I_M)$
  $f = MinElement(I)$
  $F = F \setminus f$
end while

B. Joint feature-selection algorithm

To select the shared set of features used by multiple models we propose an algorithm based on recursive feature elimination (RFE) [12] that is summarized in Algorithm 1. We start with a full set of features and iteratively eliminate the feature that has the smallest contribution to the classification performance (e.g., accuracy or F1-score) across the models. To estimate the contribution of the feature to the model classification performance we use permutation feature importance (PFI) [11] which is a decrease in a given model quality metric (e.g., accuracy or F1-score) when the values of a single feature in the validation dataset are randomly shuffled. Low value of PFI means that randomization of the feature value does not decrease model quality metric, hence the feature is not important. To assess the feature importance for all the different models globally, we sum per-model PFI values to find the PFI across the models and eliminate the feature with the smallest total PFI at each iteration of the algorithm.

To estimate time complexity of the algorithm we note that to select N features out of F by eliminating one feature at a time, we must perform training and compute feature importance for each of the M models $F - N$ times. Model training time cannot be estimated analytically, but is expected to decrease with the decrease of the number of features. We define it as $T_{Train}(i, f)$ for the i-th model that uses f features. Computing the PFI metric, used in our work, requires, for each feature, to perform inference on the validation dataset after shuffling the values of that feature. We define inference time as $T_{Inf}(i, f)$ for the i-th model that uses f features. Total time can be estimated as:

$$\sum_{i=1}^{M} \sum_{f=F}^{N} T_{Train}(i, f) + f \times |Data_i^{validation}| \times T_{Inf}(i, f) \quad (3)$$

IV. Illustrative Numerical Results

To validate the proposed approach we evaluate classification performance and number of used M/A stages for tree-based classifiers with joint and independent feature-mapping focusing on the concurrent implementation of DDoS-attack detection and application-traffic identification.

For DDoS-detection we use the dataset in [13] that contains flow-level features of legitimate and malicious traffic. We focus on application-layer attacks and assign all flows of Hulk-, GoldenEye-, slowloris-, Slowhttptest-attacks to one class, whereas legitimate traffic to another class, resulting in a binary classification problem with 10000 samples per class.

For application-traffic identification we use the dataset in [14], [15] that contains flow-level features of traffic generated

by different applications. In this case, we solve a multi-class problem, where we consider the following classes: 1) Real-time (labelled as "Skype" in the original dataset), 2) Non-real-time ("Dropbox"), and 3) Website ("Google"), with 10000 samples per class.

Both datasets use the same set of 77 flow-level features. Starting from the original feature-set, we discard metadata (e.g., IP addresses, application ports, protocol number, timestamp) [16], cumulative features that become known at the end of the flow-life (e.g., total flow duration, total number of packets, etc.) and standard-deviation statistics that cannot be easily calculated in P4. We end up with 18 features, namely, maximum, minimum and mean values of packet-length and inter-arrival time for total, forward and backward flows.

For each problem we train a RF classifier with at most 20 layers per tree. We use 60% of the data for training, 20% for feature selection and 20% for testing. Reported results are averaged across 100 random dataset-splits, where classes proportions are always maintained as in the entire dataset.

In Fig. 4a-c we report classification accuracy, F1-score and number of used M/A stages for the 2 models using 3 trees each and comparing independent and joint feature-mapping for increasing number of features selected by the algorithm described in Section III-B. From Fig. 4a we see that DDoS-detection classifier reaches accuracy of 98%, whereas application identification provides accuracy up to 88%. F1-score (see Fig. 4b) shows similar trends as accuracy for all cases. As we increase the number of used features, accuracy and F1-score start to improve and then saturate when more than 5 features are used. We can also see that classification performance is insignificantly affected by joint feature-mapping (cross-markers are close to circle-markers), even though both models are constrained to use the same features. Reduction in accuracy and F1-score is noticeable for a small number of features and almost disappears when more than 7 features are used. Fig. 4c demonstrates that joint feature-mapping saves up to (20-40)% of M/A stages. Savings in M/A stages increase with the number of used features, as number of stages used for feature-to-code mapping grows, while 6 stages (3 trees per model) are always used for codeword-to-label mapping (see Fig. 2).

In Fig. 4d-e we report F1-score for the different number of M/A stages used to implement the 2 problems (see Eq. 1, 2). We can see that the highest F1-score reached with joint feature-mapping is comparable to the one reached with independent mapping, while 5-6 fewer M/A stages are used, confirming that around 25% M/A stages can be saved at the same classification performance.

In Fig. 4f we focus on the possible trade-offs from using higher number of trees in the scenario with joint feature-mapping. We report F1-score for the different number of M/A stages used to implement the 2 problems when 3 or 5 trees are used in each model. We can see that at 16 M/A stages we can achieve the same F1-score with the higher number of trees, but use fewer features (i.e., 6 features with 5 trees vs. 10 features with 3 trees), thus saving some additional M/A stages used to calculate features in the switch.

V. Conclusion

We propose joint feature-mapping for resource-efficient implementation of multiple concurrent tree-models in P4 switches and demonstrate that different network-management tasks (i.e., DDOS detection and application identification considered in our work) can be performed using a small number of shared features, and up to 40% of M/A stages in the packet processing pipeline can be saved, while maintaining high classification accuracy.

As future work, we plan to evaluate the proposed approach on real devices to consider M/A table size when quantifying the number of used M/A stages, and account for increased memory consumption associated with the sharing of feature-mapping stages between multiple models. We will also investigate different feature-sharing scenarios (e.g., partial sharing of features between models) and more intelligent joint feature-selection across different models.

VI. Acknowledgement.

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

References

[1] C. Zheng, et al., "In-Network Machine Learning Using Programmable Network Devices: A Survey," in IEEE Communications Surveys & Tutorials, vol. 26, no. 2, pp. 1171-1200, 2024
[2] C. Busse-Grawitz et al., "pForest: In-Network Inference with Random Forests", ArXiv, abs/1909.05680, 2019
[3] C. Zheng, et al., "Planter: seeding trees within switches," in the Proceedings of SIGCOMM '21, pp. 12–14, 2021.
[4] S.U. Jafri et al., "Leo: Online ML-based Traffic Classification at Multi-Terabit Line Rate, in the Proceedings of NSDI 24, pp. 1573-1591, 2024
[5] G. Siracusano and R. Bifulco, "In-network Neural Networks," arXiv preprint arXiv:1801.05731, 2018.
[6] L. Cannarozzo et al., "Spinner: Enabling In-network Flow Clustering Entirely in a Programmable Data Plane," in the Proceedings of NOMS 2024, 2024, pp. 1-9
[7] G. Xie et al., "Mousika: Enable General In-Network Intelligence in Programmable Switches by Knowledge Distillation," in the Proceedings of IEEE INFOCOM 2022, pp. 1938-1947, 2022
[8] P. Bosshart, et al., "P4: programming protocol-independent packet processors" in ACM SIGCOMM Computer Communication Review 44 (3), pp. 87–95, 2014.
[9] https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html
[10] L. Grinsztajn et al., "Why do tree-based models still outperform deep learning on typical tabular data?" in the Proceedings NeurIPS 2022, vol. 35 pp. 507-520, 2022
[11] L. Breiman, "Random Forests", Machine Learning, 45. pp. 5–32, Jan. 2001.
[12] I. Guyon, et al., "Gene Selection for Cancer Classification Using Support Vector Machines", Machine Learning, 46. pp. 389-422, Jan. 2002.
[13] https://www.unb.ca/cic/datasets/ids-2017.html
[14] https://www.kaggle.com/datasets/jsrojas/ip-network-traffic-flows-labeled-with-87-apps
[15] J. S. Rojas, et al., "Consumption Behavior Analysis of Over the Top Services: Incremental Learning or Traditional Methods?," in IEEE Access, vol. 7, pp. 136581-136591, 2019
[16] L. D'hooge, et al., "Establishing the Contaminating Effect of Metadata Feature Inclusion in Machine-Learned Network Intrusion Detection Models," in the Proceedings of DIMVA 2022, June 2022.