Monic: In-Network Mixture-of-Experts Inference on Programmable Data Planes

Xiaoquan Zhang[1], Bowen Liang[1], Fung Po Tso[2], Yuhui Deng[1], Zhen Zhang[1],
Kaimin Wei[1], Weijia Jia[3], and Lin Cui[1,*]
[1]*Department of Computer Science, Jinan University, Guangzhou, China.*
[2]*Department of Computer Science, Loughborough University, UK.*
[3]*Institute of Artificial Intelligence and Future Networks, Beijing Normal University (BNU Zhuhai), China.*

***Abstract*—In-network inference has emerged as a promising paradigm for enabling intelligent packet processing at the line rate within programmable data planes. However, it is fundamentally limited by an inherent conflict between model accuracy and the resource constraints of programmable data planes. This forces a compromise: monolithic deployments can achieve high accuracy but are constrained by the resource limits of a single switch, while distributed approaches leverage the combined resources of multiple switches but are limited in accuracy due to a lack of model coordination. We resolve this trade-off by proposing *Monic*, a framework that enables multiple "expert" submodels to perform collaborative inference. Inspired by the Mixture-of-Experts (MoE) paradigm, *Monic* uses a pipeline-compatible gating mechanism to selectively activate experts across the network. We enable this in practice through a resource-aware mapping and co-optimization strategy that automatically identifies optimal configurations under hardware constraints. We have implemented *Monic* using P4 hardware switches with Intel Tofino ASIC. Our evaluation shows that *Monic* achieves a 17.4% relative improvement over baseline methods and maintains a 29.58% Macro F1 advantage under scalability evaluation, demonstrating that a collaborative approach can simultaneously achieve resource efficiency and superior accuracy.**

***Index Terms*—P4, in-network inference, programmable data plane, decision trees, mixture of experts**

I. Introduction

Recent advances in programmable data planes, such as those enabled by P4 and the Intel Tofino ASIC [1], have given rise to the intelligent data plane (IDP) [2], where machine learning (ML) inference is integrated directly into the packet processing pipeline. IDPs enable network applications that demand high responsiveness and massive throughput at line rate, such as malicious flow detection [3], DDoS mitigation [4], and traffic classification [5]. To this end, researchers have explored hardware-amenable models, such as tree-based systems [5], [6] and quantized neural networks [7], [8], with trees being particularly favored due to their alignment with the switch's match-action paradigm. However, the data plane's stringent constraints on memory, logic, and pipeline stages make deploying any complex model difficult, creating a fundamental challenge of balancing model accuracy against hardware compatibility [6].

Corresponding author: Lin Cui. Email: tcuilin@jnu.edu.cn
Xiaoquan Zhang and Bowen Liang have contributed equally to this paper.

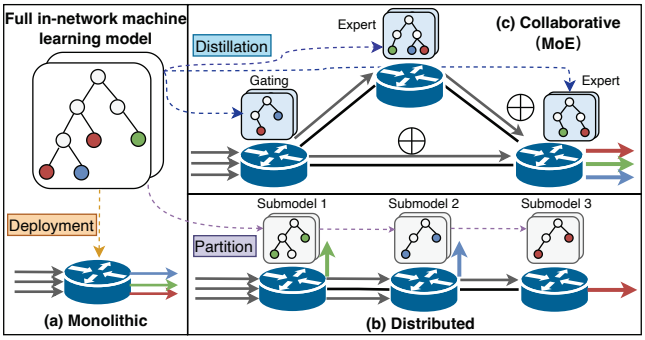
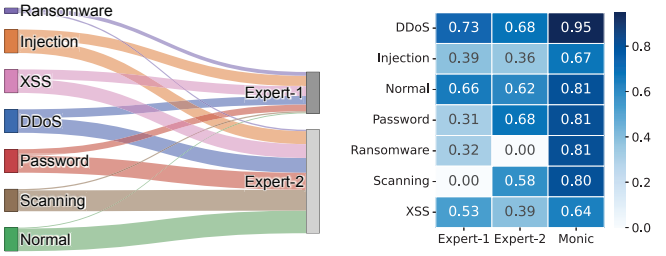


Fig. 1. Conceptual illustration of (a) monolithic, (b) distributed, and (c) the proposed collaborative inference (i.e., MoE) architecture.

Existing implementation of in-network ML models is broadly classified into two categories: *monolithic* and *distributed*, as demonstrated in Fig. 1. The monolithic approach deploys the entire model on a single switch [4], [5], [8], [9]. Although this design is straightforward, it faces significant scalability limitations [6] as model complexity consumes critical hardware resources such as table entries and stage usage. In contrast, distributed approaches partition a full model across multiple switches in sequence to improve resource scalability, for instance, decomposing a random forest into multiple decision trees for deployment [6], [10], or splitting a complete neural network by network layers across switches [11]–[13]. This distributed approach creates an accuracy bottleneck because it executes model partitions sequentially across switches (Fig. 1). With each switch making predictions independently, the submodels operate as uncoordinated fragments, fundamentally capping the system's overall performance at the level of the full model [6].

To overcome these limitations, we introduce an in-network *collaborative* inference framework inspired by the mixture-of-experts (MoE) paradigm [14]. Unlike the uncoordinated distributed method, our approach models each deployed submodel as an *expert* and uses a gating mechanism to coordinate their participation. This allows multiple experts to participate in each inference via soft voting, thereby capturing complementary knowledge across network devices. To preliminarily validate this idea, we conducted a motivation experiment using



(a) Gating behavior. (b) Prediction accuracy.

Fig. 2. Motivation experiment based on ToN-IoT dataset with one gating network and two experts. (a) The thickness of each link indicates the strength of the association between a traffic class and an expert. (b) Experts with small-sized trees through collaborative inference can achieve highly accurate predictions.

an MoE model comprising a *gating network* and two decision tree experts. The results in Fig. 2 demonstrate that different experts exhibit distinct inference capabilities. Even when individual experts with small-scale models achieve relatively low accuracy, the aggregated predictions guided by the gating mechanism achieve high overall accuracy. This property is particularly well-suited for resource-constrained environments, such as in-network inference.

However, the design of this MoE framework presents three key challenges. (i) Since each expert operates on a partial view of the input space, enabling collaboration among experts becomes a key challenge for improving inference accuracy beyond the limits of non-collaborative distributed models. (ii) Jointly selecting and training the optimal expert configuration under hardware resources becomes a non-trivial challenge, as the design space is large and conventional search strategies are impractical. (iii) Due to unsupported complex operations such as multiplication and strict per-stage limitations, efficiently mapping the MoE mechanism onto the hardware pipeline presents a practical deployment challenge.

To tackle these challenges, we propose *Monic* (Mixture of experts in switches), a framework that leverages the MoE paradigm for joint inference on programmable switches. We introduce a hardware-aware mapping methodology that uses Bayesian optimization to find the optimal expert configuration under strict hardware constraints. Then, we design an expectation-maximization-inspired refinement process where a high-fidelity teacher model guides the experts and the gating network towards collaborative accuracy. We also develop a modular P4 data plane architecture with composable processing units to ensure scalable MoE deployment.

The main contributions of this paper are:

- A novel collaborative inference framework, *Monic*, that successfully deploys a Mixture-of-Experts (MoE) architecture on programmable data planes to overcome the accuracy and scalability limitations of prior monolithic and distributed approaches.
- A hardware-aware optimization methodology that automatically decomposes complex models into resource-fit experts and allocates features under hardware constraints, leveraging a prediction-aligned refinement strategy to maximize systemic accuracy.
- A testbed prototype of *Monic* is implemented using Intel Tofino ASIC switches. Experiment results demonstrate a superior performance over state-of-the-art baselines, achieving 98.09% accuracy (17.4% relative improvement) and showing robust scalability with a 29.58% accuracy gain.

II. Background and Related Works

A. Background

Hardware constraints: ML deployment in programmable switches like the Intel Tofino [1] is governed by the stringent hardware constraints of the protocol-independent switch architecture (PISA). Each match-action pipeline stage provides limited SRAM for exact matching and registers, and TCAM for ternary or prefix matches. The total available memory is modest (e.g., 15MB SRAM), and TCAM width is capped at fixed lengths (e.g., 36, 72, 144, or 288 bits), limiting support for high-dimensional feature comparisons. Crucially, stateful memory components such as registers can only be accessed once per packet per pipeline, prohibiting read-modify-write behavior. Thus, actions must be staged carefully due to inter-stage dependencies and limited per-stage resources.

In-network model deployment: In a decision tree, a path from the root to a leaf can be viewed as a conjunction of feature range checks. These paths can be directly encoded into match-action entries via *range* matching [4], [5], [9], [15], [16]. Practical implementation requires transforming range conditions into ternary bit patterns using prefix expansion, which can lead to a combinatorial explosion in table size under moderately complex trees, limiting scalability.

Neural network inference requires complex floating-point matrix operations not natively supported by data plane hardware. This is overcome by converting models into fixed-point representations through quantization [7], [8], resulting in a compromise between accuracy and hardware feasibility.

B. Related Work

Monolithic models: Prior works have proposed in-network monolithic inference using both tree-based and neural network models on programmable switches. For tree-based models, pForest [15] and pHeavy [5] validated early deployment, while Jewel [17] enhanced feature expressiveness to improve accuracy. Later efforts, such as Planter [9] and Flowrest [16], introduced encoding strategies to reduce TCAM usage, and NetBeacon [4] employed sparse vector encoding and cross-path compression to scale random forest deployment. SentinelX [3] further partitioned large trees into multi-stage flow tables and proposed DualTree to address redundant traffic patterns. In parallel, Quark [8] explored in-network neural inference by implementing quantized matrix operations with match-action tables and using recirculation to support multi-layer CNNs. Despite their innovations, they are fundamentally limited by the resource capacity of a single switch,

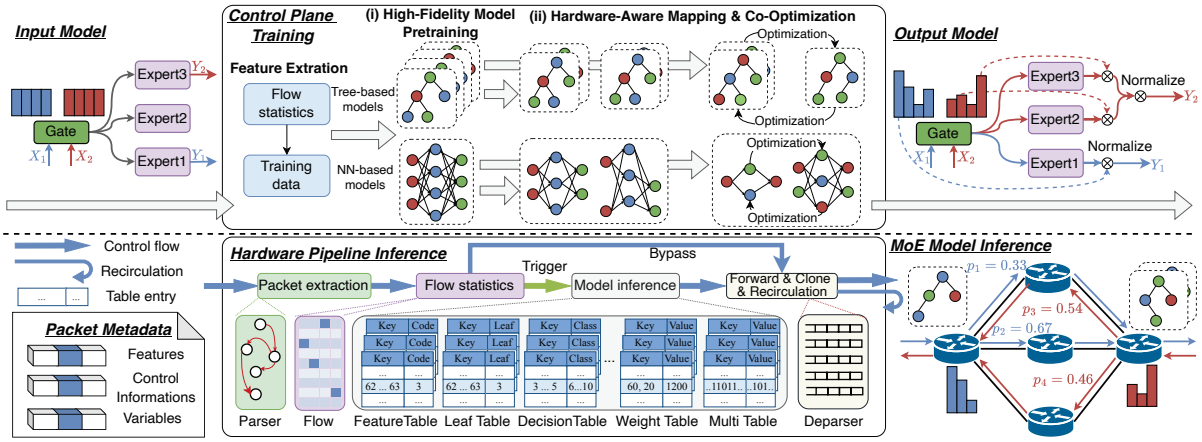


Fig. 3. An overview of the *Monic* architecture.

thus restricting their use in more sophisticated or large-scale inference tasks in practice.

Distributed models: To enable the deployment of models beyond the capacity of a single switch, several works have proposed distributing model components across multiple switches. DUNE [6] decomposed a random forest into smaller sub-models, each responsible for a subset of classes and features, and deployed them sequentially along the packet path. In parallel, recent studies [11]–[13], [18] have extended this idea to neural networks by assigning different layers to multiple switches and transmitting intermediate results through packet headers. However, in both tree- and neural network-based approaches, each sub-model only represents a fragment of the full model and operates independently, without collaborative reasoning, which fundamentally limits the overall inference accuracy.

Progress beyond the state of the art: *Monic* combines the strengths of both monolithic and distributed approaches by deploying small, individual expert models across a distributed set of switches. These experts then collaborate to deliver the high accuracy of a complex, single model while retaining the resource efficiency and scalability of a distributed system.

III. Overview

The *Monic* architecture, depicted in Fig. 3, comprises two synergistic components that handle model preparation and execution separately: the control plane for training and optimization, and the data plane for high-speed packet inference.

A. Control Plane

The control plane is responsible for building a resource-aware MoE model that maximizes inference accuracy within the constraints of programmable switch resources. It consists of two key modules. *(i)* The *High-Fidelity Model PreTraining Module* trains an unconstrained, full-precision model that serves as a teacher to guide the subsequent optimization process. *(ii)* The *Hardware-Aware Mapping and Co-Optimization Module* jointly explores the MoE architecture space and refines model parameters, distilling the teacher's knowledge into a hardware-feasible gating network and a set of expert models. The output model supports distributed deployment across multiple switches, enabling collaborative and accurate in-network inference under stringent resource constraints.

B. Data Plane

1) Control Flow. The hardware pipeline supports online inference by integrating flow feature statistics with pre-defined gating and expert decision logic. Upon packet arrival, the parser extracts the header fields. Then, flow statistics (e.g., packet length, TCP flags) are accumulated and maintained in registers. Once the triggering condition is satisfied (e.g., after observing the initial n packets of a flow), in-network ML model inference will execute. The resulting prediction is stored in the flow's register state, and all subsequent packets of the same flow bypass both feature accumulation and model inference, directly reusing the cached decision.

2) Model Inference. *Monic* utilizes MATs to apply ML models' inference, including tree-based models [4], [9], [17] and neural network models [8]. The MoE model generalizes this pipeline to support collaborative inference across distributed experts. Packets, arriving from arbitrary directions, are first processed by the gating logic to assign soft weights to each expert, e.g., p_i to the i-th expert. If a single expert dominates (e.g., p_i exceeds a confidence threshold), Top-1 routing forwards the packet exclusively to that expert; otherwise, Top-K routing dispatches the packet to multiple experts in parallel. Each selected expert executes its local inference pipeline and generates partial class probability distributions. These results are subsequently forwarded to an aggregation point, where a weighted combination of expert outputs is computed to produce the final prediction.

IV. Model Training for *Monic*

In this section, we focus on the training of MoE architectures based on decision tree models for two principled reasons: high accuracy for in-network inference and inherent training difficulty stemming from the non-differentiable structure, and

we also provide a brief discussion on neural network-based MoE training in Section IV-D.

A. High-Fidelity Model PreTraining Module

To guide a collaborative inference model, the training process begins with a resource-unconstrained teacher model, M_T, which leverages the complete feature set without being limited by P4 switch constraints to maximize accuracy. This high-fidelity model serves two critical roles. First, its peak accuracy establishes the quantitative benchmark against which the final *Monic* model is evaluated. Second, it generates the dense supervisory signals required to guide the hardware-aware mapping module in Section IV-B and the co-optimization module in Section IV-C.

Monic employs a random forest (RF) as the tree-based teacher model. This choice offers two strategic advantages: it provides interpretable feature importance metrics essential for hardware-aware mapping (Section IV-B), and its strong empirical performance on high-dimensional network data makes it an ideal baseline for establishing a robust performance ceiling.

Using the full set of network features, the hyperparameters of M_T are optimized via a grid search to identify the configuration that yields the highest accuracy. The crucial output of this module is the set of soft-label probability distributions, $P_T = M_T(\mathbf{x})$, for all training instances. In contrast to sparse hard-labels, these probabilistic vectors convey the teacher's nuanced reasoning, including their confidence and assessment of inter-class similarities. This rich information is fundamental to guiding the subsequent training module.

B. Hardware-Aware Mapping Module

The problem of concurrently partitioning the feature set F and distributing the finite computational budget (i.e., decision trees) among m experts and a gating network is a complex joint optimization problem. This task can be formulated as a generalized assignment problem [19], which is a known NP-hard problem. The problem is made particularly challenging by the inter-dependencies inherent in the MoE model, since allocating features to one expert inevitably influences the optimal resource allocation of all others.

To solve this problem, we decompose the problem into two sequential, more manageable sub-problems: *(1)* establishing a robust feature allocation for expert specialization, and *(2)* optimizing resource allocation across this fixed structure.

1) Hierarchical Feature Allocation. The effectiveness of MoE model hinges on a critical design trade-off: experts must share enough knowledge to understand the problem space but possess unique feature sets to develop specialized perspectives. To address this, we employ a hierarchical feature allocation strategy centered on Recursive Feature Elimination (RFE). First, a shared core of the most globally important features, F_{core}, is allocated to all experts to ensure a baseline of common knowledge. Subsequently, RFE is used to select specialized features for each expert. As an embedded method, RFE iteratively builds models and discards the least important features, enabling it to account for feature dependencies and interactions. By balancing tractability and robustness, RFE enables the discovery of complementary and highly predictive feature subsets that support effective expert collaboration.

2) Optimal Resource Allocation under Hardware Constraints. *Monic* targets the efficient deployment of a modular MoE architecture with resource constraints. Given m experts and a gating network, the model must determine an allocation vector $\mathbf{n} = [n_1, \ldots, n_m, n_g]$, where n_i and n_g denote the resources assigned to expert E_i and the gating network G, respectively. The objective is to maximize prediction accuracy while ensuring that the overall SRAM and TCAM footprint fits within hardware limits:

$$\mathbf{n}^* = \arg\max_{\mathbf{n}} \ \text{Accuracy}(M_{\text{MoE}}(\mathbf{n}, D_{\text{train}}), D_{\text{val}}) \quad (1)$$

$$\text{s.t. } \text{S}_{\text{total}}(\mathbf{n}) \le B_{\text{SRAM}},\ \text{T}_{\text{total}}(\mathbf{n}) \le B_{\text{TCAM}},\ n_i, n_g \ge 1, \quad (2)$$

where $\text{S}_{\text{total}}(\mathbf{n})$ and $\text{T}_{\text{total}}(\mathbf{n})$ represent the estimated memory footprint, and B_{SRAM}, B_{TCAM} are the hardware budgets. Since both each expert and the gating network can be transformed into either tree-based models [9] or neural network-based models [8] represented by match-action table entries, S_{total} and T_{total} are computed as the total size of the resulting match-action tables. Thus, we can leverage existing placement approaches [20], [21] to ensure that per-stage hardware resource constraints are satisfied.

In the context of Tree-based MoE model, *Monic* employs Bayesian Optimization (BO) with a Tree-structured Parzen Estimator (TPE) as the core strategy for optimal resource allocation. BO builds a probabilistic surrogate model of the objective function to guide exploration. At each iteration, the surrogate model estimates the performance of untested allocations. The acquisition function then selects the next configuration by trading off exploration (sampling in high-uncertainty regions) and exploitation (refining allocations with high predicted performance). This process ensures that each evaluation contributes maximally to discovering an optimal allocation. The optimization workflow begins by proposing a feasible allocation $\mathbf{n}$ that respects SRAM and TCAM budgets. It then trains the corresponding experts on the training dataset, synthesizes gating labels based on expert confidences, and trains the gating network accordingly. The accuracy on a validation set is computed and fed back into the surrogate model for subsequent updates. This iterative process converges to an allocation $\mathbf{n}^*$ that provides near-optimal performance while satisfying hardware constraints.

C. Co-Optimization Module

The final module addresses the challenge of training the MoE components to operate as a cohesive whole. This presents two primary obstacles: *(i)* the tree-based models are non-differentiable, precluding standard end-to-end backpropagation, and *(ii)* a naive and isolated training of components creates a training-inference skew, where the training objective diverges from the model's deployment logic.

To overcome these challenges, *Monic* introduces the co-optimization Module, which enables joint refinement of the

gating network and experts through an iterative process. The module is based on the expectation-maximization (EM) framework. In our context, the latent variable is the identity of the optimal expert for a given sample. This framework serves as a surrogate for gradient-based learning, allowing the gating network and experts to co-adapt in a manner that explicitly aligns the training procedure with the final inference mechanism. The process alternates between the following two steps.

1) E-Step: Refining Gating Logic. In the E-step, the gating network G is retrained to learn an optimal distribution-aware routing policy. Rather than routing to the single most accurate expert, we train G on a more nuanced objective: to identify the expert whose probabilistic reasoning best aligns with that of the high-fidelity teacher model M_T. The training target for G for each instance x is the index of the expert that minimizes the Kullback-Leibler (KL) divergence to the teacher's distribution:

$$y_{\text{gating}}(\mathbf{x}) = \underset{i \in \{1,\ldots,k\}}{\operatorname{argmin}} \; D_{\text{KL}}(P_T(\mathbf{x}) \;||\; P_i(\mathbf{x})). \tag{3}$$

This formulation encourages the gating network to learn a routing strategy based on holistic distributional similarity, fostering a more intelligent collaboration.

2) M-Step: Weighted Expert Distillation with Adaptive Gating. In the M-step, each expert E_i is updated by minimizing a weighted composite loss function that balances direct supervision and knowledge distillation from the teacher model M_T. The loss for each sample $\mathbf{x}$ assigned to expert E_i is defined as:

$$\begin{aligned} L_i(\mathbf{x}) = w(\mathbf{x}, E_i) \cdot [(1-\alpha)H(y, P_i(\mathbf{x})) + \\ \alpha D_{\text{KL}}(P_T(\mathbf{x}) \| P_i(\mathbf{x}))]. \end{aligned} \tag{4}$$

Here, $H(y, P_i(\mathbf{x}))$ denotes the cross-entropy loss between the ground-truth label y and the expert's prediction $P_i(\mathbf{x})$, while $D_{\text{KL}}(P_T(\mathbf{x}) \| P_i(\mathbf{x}))$ measures the KL divergence between the teacher's output distribution $P_T(\mathbf{x})$ and that of the expert. The hyperparameter $\alpha \in [0, 1]$ controls the trade-off between fitting true labels and distilling knowledge from the teacher. In our evaluation, α is set to 0.3 to balance label supervision and distillation, improving expert generalization under constrained conditions.

The sample weight $w(\mathbf{x}, E_i)$ plays a crucial role in this adaptive framework. It is derived from the gating network's confidence and the expert's alignment with the teacher, ensuring that each expert focuses its capacity on the data samples for which it is most responsible. Specifically, if the gating confidence for an input exceeds a predefined threshold, a hard assignment is made and the expert receives a weight proportional to the distillation alignment. Otherwise, a soft assignment is applied: the Top-K experts with the highest gating scores share the responsibility, with weights normalized accordingly. This adaptive gating strategy effectively balances specialization and collaboration among experts. In our setting, the Top-1 confidence threshold is set to 0.8 to ensure high-certainty routing while allowing fallback to soft assignment in cases of uncertainty.

D. Neural Network-Based MoE Training

Monic also supports neural network-based MoE training via the two modules:

1) High-Fidelity Model PreTraining Module. We first train a full-precision (FP32) MoE model as a high-accuracy teacher model, which comprises a gating function and multiple experts trained jointly to capture task semantics and ensure balanced expert utilization. The training objective is:

$$\mathcal{L}_{\text{teacher}} = \mathcal{L}_{\text{task}} + \lambda \mathcal{L}_{\text{balance}}. \tag{5}$$

Here, $\mathcal{L}_{\text{task}}$ denotes the primary loss (e.g., cross-entropy), and $\mathcal{L}_{\text{balance}}$ encourages uniform routing across experts. The $\lambda \in [0, 1]$ is a tunable hyperparameter balancing accuracy and load distribution.

2) Hardware-Aware Mapping & Co-Optimization. To meet hardware constraints, we construct a quantized student model with low-bit variants via quantization-aware training [22]. To bridge the accuracy gap, we apply knowledge distillation from the full-precision teacher:

$$\mathcal{L}_{\text{student}} = \alpha \mathcal{L}_{\text{task}} + (1-\alpha)\mathcal{L}_{\text{KD}}\left(\sigma\left(\frac{z_t}{\tau}\right), \sigma\left(\frac{z_s}{\tau}\right)\right). \tag{6}$$

Here, $\mathcal{L}_{\text{KD}}$ denotes the KL divergence between the teacher (z_t) and student (z_s) logits with temperature τ. The hyperparameter $\alpha \in [0, 1]$ controls the trade-off between ground-truth supervision and teacher guidance.

Due to the page limit, we leave training details and accuracy enhancement strategies to future work.

V. *Monic* Design in Data Plane

A. Modular Design for MoE Model

The structure of MoE model, composed of a gating network and multiple experts, naturally lends itself to modularization. We adopt this insight by decomposing the MoE model into two reusable P4 components: a *Gating Unit*, responsible for expert selection, and a set of *Expert Unit*, each encapsulating a single expert. This modular design not only mirrors the model's structure but also simplifies deployment in the data plane, allowing independent development, optimization, and reuse of core inference components.

The modular architecture enables flexible resource utilization within constrained switch environments. Since the trained MoE model is tailored to hardware resource budgets, it rarely saturates all available pipeline stages. As a result, more Expert Units can be opportunistically instantiated on switches with residual capacity, co-existing with existing Expert Unit logic. This minimizes inter-switch communication and reduces packet replication overhead along the data path. By decoupling model structure from fixed hardware layout, the design supports efficient and scalable deployment without requiring global reconfiguration.

B. P4 Design for a Probabilistic RF

Prior works focus on mapping tree models to P4 match-action tables (MATs), whose design is limited to a single model with deterministic class prediction accuracy. This is

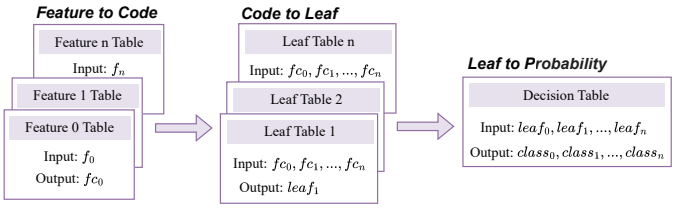


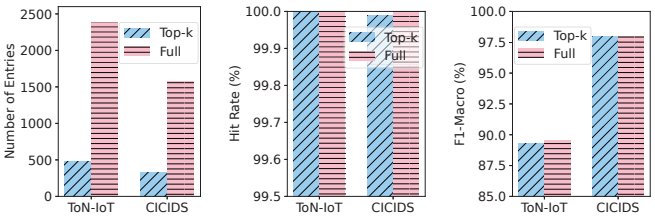
Fig. 4. Implement Probability RF in the P4 Switch.

insufficient for the implementation of the two fundamental units (i.e., the Gating-Unit and the Expert-Units). Thus, a more sophisticated building block is required to produce a full class probability distribution, enabling weighted voting and nuanced decision-making. *Monic* introduces a novel two-phase P4 implementation that transforms a standard RF into a probabilistic inference engine.

1) Feature Coding to Signature Mapping. In the general implementation of tree models, the final action at a leaf node directly assigns the class label with the most votes. In *Monic*, this action is repurposed to return a unique and pre-assigned `leaf_id` that represents the specific path the packet traversed through that tree. For an RF with T trees, the packet is processed in parallel by T logical tree pipelines. The collective output is a tuple of leaf identifiers, $(l_1, l_2, \ldots, l_T)$, where l_t is the `leaf_id` from tree t. This tuple forms a unique signature that holistically represents the packet's characteristics as judged by the entire forest. This design is illustrated in Fig. 4. Unlike Planter's direct feature to class mapping [9], *Monic* maps features to an intermediate and high-dimensional representation.

2) Signature to Probability Distribution Mapping. The second phase maps `leaf_id` signature to a final probability vector. This is achieved with a final MAT (called decision table), where the match key is the tuple of `leaf_id`s.

However, this fusion introduces a critical challenge of a combinatorial table explosion. The theoretical size of this decision table is the Cartesian product of the number of leaves in each tree, $|L_1| \times |L_2| \times \cdots \times |L_T|$. For even a moderate-sized forest, this number can grow exponentially, creating a table far too large to fit within the stringent memory constraints of programmable switches. The strawman implementation, therefore, introduces significant computational overhead.



(a) Entry Number. (b) Hit Rate. (c) Accuracy.

Fig. 5. Top-K vs. Full in *Monic*.

We introduce a domain-aware compression strategy: Top-K signature compression. This heuristic is motivated by the observation that network traffic patterns are not uniformly distributed; a small subset of behaviors (and thus feature combinations) typically accounts for the vast majority of traffic. During offline training, we analyze the frequency of each `leaf_id` combination in its training dataset and cache only the K most frequently occurring combinations, along with their corresponding probability distributions, in the hardware decision table. This approach yields a highly favorable trade-off between resource consumption and inference accuracy. As demonstrated by our evaluation (Fig. 5), the Top-K strategy dramatically reduces the memory footprint of the decision table. In the most demanding case (the expert on the ToN-IoT dataset), our method reduces the required table entries from 2,384 to just 475, achieving a 80% resource saving by compressing the table size by a factor of 5. This substantial resource saving comes at a negligible cost: the hit rate for the cached entries remains above 99.97% across all experiments, resulting in a Macro F1-score degradation of less than 0.25%. For the rare cases where a `leaf_id` combination misses (e.g. average 0.03%) in the cache, the packet is forwarded to the control plane for handling, ensuring correctness without compromising real-time inference in the data plane.

C. Analysis of Top-K Signature Compression

The effectiveness of this heuristic is not coincidental. The features used for inference (e.g., packet sizes, TCP flags, etc.) are direct reflections of underlying network behaviors and application patterns. It is a well-documented principle in networking literature that various traffic characteristics (e.g., flow sizes, file popularity, and content access frequencies) follow heavy-tailed, power-law distributions [23], [24]. Since each packet's path through a decision tree determines a `leaf_id` combination, the frequency distribution of these combinations, denoted $P(c)$, inherits this skew. We therefore model $P(c)$ using Zipf's law [25], which accurately captures such long-tailed behavior.

Let the $|C|$ unique `leaf_id` combinations be sorted by their frequency rank, where $P(c_i)$ denotes the probability of the i-th most frequent combination:

$$P(c_i) = \frac{1/i^s}{\sum_{j=1}^{|C|} 1/j^s}. \tag{7}$$

Here, the exponent $s > 0$ determines the skewness of the distribution: larger s values yield distributions with more dominant top entries.

To assess the compression-accuracy trade-off, we consider the cumulative inference error introduced by retaining only the top K combinations and discarding the remaining $|C| - K$. The key result is summarized below.

Theorem 1. *Given a Zipfian distribution over* `leaf_id` *combinations with exponent $s > 1$, the number of cached entries K required to bound the total inference error within ϵ scales sub-linearly with $1/\epsilon$. Specifically,*

$$K = \mathcal{O}((\frac{1}{\epsilon})^{\frac{1}{s-1}}). \tag{8}$$

Proof. Let $\text{Error}_{\max}$ denote the cumulative probability mass of the discarded entries:

$$\text{Error}_{\max} = \sum_{i=K+1}^{|C|} P(c_i) = \frac{\sum_{i=K+1}^{|C|} 1/i^s}{\sum_{j=1}^{|C|} 1/j^s}. \tag{9}$$

For large $|C|$, the denominator converges to the Riemann zeta function $\zeta(s)$, while the numerator can be upper-bounded via an integral:

$$\sum_{i=K+1}^{\infty} \frac{1}{i^s} \leq \int_K^{\infty} \frac{1}{x^s} dx = \frac{K^{1-s}}{s-1}. \tag{10}$$

Thus,

$$\text{Error}_{\max} \leq \frac{K^{1-s}}{(s-1)\zeta(s)}. \tag{11}$$

To ensure $\text{Error}_{\max} \leq \epsilon$, we solve:

$$\frac{K^{1-s}}{(s-1)\zeta(s)} \leq \epsilon \Rightarrow K \geq (\epsilon(s-1)\zeta(s))^{\frac{1}{1-s}} \propto \left(\frac{1}{\epsilon}\right)^{\frac{1}{s-1}}. \tag{12}$$

Hence, the number of cached entries grows sub-linearly with $1/\epsilon$ when $s > 1$, completing the proof. □

The analysis justifies our Top-K Table Compression heuristic as a solution to the combinatorial table explosion in probabilistic RF design. This approach significantly reduces SRAM usage while preserving accuracy, making it suitable for implementation on programmable switches.

VI. *Monic* Implementation in Data plane

A. ML model implementation in the P4 switch

We introduce two sub-units to support tree- and neural network-based inference: FLD-subUnit (Feature, Leaf, Decision Table) and CAP-subUnit (Convolutional, Activation and Pooling Layer) following the approach in Quark [8]. The Gating-Unit and Expert-Units instantiate the appropriate sub-unit based on the selected MoE model, either FLD-subUnit or CAP-subUnit.

An FLD-subUnit can implement a random forest using a combination of feature, leaf, and decision tables, each occupying a single pipeline stage. To maximize resource utilization and processing throughput, multiple instances of these tables are deployed in parallel within each stage, allowing the switch to host multiple FLD-subUnits concurrently. This parallel design minimizes inter-unit communication overhead and supports the deployment of multiple Expert-Units with low latency and high throughput, enabling the system to scale efficiently to a large number of experts.

B. Implementation of MoE across the data plane

We implement *Monic* on Intel Tofino ASIC using P4_16, with the logic organized into the Gating-Unit and the Expert-Unit, which can be deployed on the same or separate switches. Collaborative inference is orchestrated through a custom header that carries inference state and leverages native hardware features for efficient packet-level prediction. The header is compact and hardware-friendly, designed to encode inference metadata with minimal overhead.

The inference process commences within the ingress pipeline of the switch functioning as the Gating Unit. Upon arrival, a packet is identified and has a custom *Monic* protocol header prepended. The Gating-Unit then applies its ML process logic using the 16-bit or 8-bit widths {`Feature0...FeatureN`} fields, populating the `BitMap` and 8-bit widths {`Prob0`, ..., `ProbK`} fields in the `monic_h header`. The `BitMap`, where each bit corresponds to one expert, is used as a key to a table that maps it to a pre-configured hardware multicast group ID, enabling line-rate and one-to-many packet replication by the switch fabric.

The replicated packets are subsequently processed by their designated Expert-Units. To conserve resources, an Expert-Unit's pipeline is only executed if its corresponding bit is set as 1 in the `BitMap`. The selected Expert-Unit first applies its own ML process logic to compute a local class probability distribution.

To apply the gating calculation weight to the corresponding expert's probability distribution, we implement multiplication via a pre-computed MAT to avoid computation burden in the hardware pipeline. This table takes the gating weight and a class probability as a composite key and returns the result of the pre-calculated product. The Expert-Unit uses these tables to compute its weighted probability distribution, which it then writes into its dedicated 16-bit width fields {`Class0`,...,`ClassN`} in the `monic_h` header before forwarding the packet to the aggregation point.

The collaborative inference culminates at a designated aggregation point. Its task is simplified because the complex multiplication has already been distributed to the expert switches. The aggregation pipeline simply sums the pre-weighted probability vectors from each participating expert. A final sequence of max operations then identifies the class with the highest cumulative score, which is taken as the final prediction result. This result is written to the 8-bit widths `Class` field, and a corresponding forward policy is applied.

VII. Evaluation

A. Experiment Setup

Testbed: Our hardware testbed comprises two hardware programmable switches (Flnet S9180-32X with Intel Tofino ASIC [1]), running Intel SDE versions 9.7.0, and two servers are equipped with Intel Core i7-4771 CPUs (8 cores @3.5 GHz) and Mellanox ConnectX-3 40 GbE NICs, as both sender and receiver. All server-to-switch links are 40 Gbps fiber optical connections. For the software environment, we utilize Mininet [26] to emulate a network topology consisting of eight switches and two hosts acting as traffic sources and sinks. The switches employ the P4 behavioral model v2 (BMv2) SimpleSwitch target [27].

Datasets and Use Cases: We evaluate *Monic* on three challenging public datasets to demonstrate its accuracy and generalizability across diverse network scenarios. All datasets

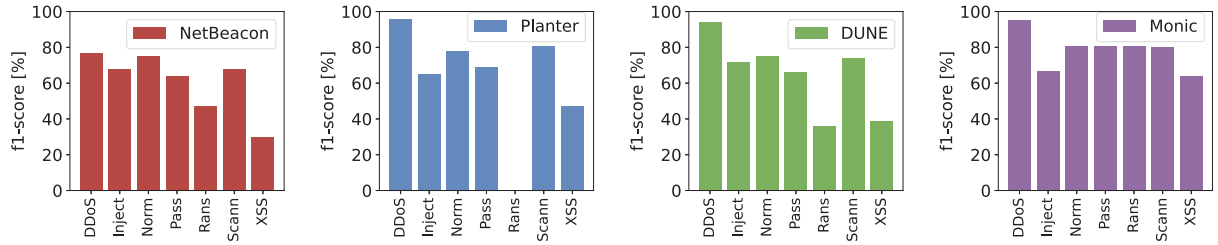


Fig. 6. Per-class F1-score comparison on the ToN-IoT dataset (DDoS, Injection, Normal, Password, Ransomware, Scanning, XSS).

are split into 60% training, 20% validation, and 20% testing. For application-level classification, we use ISCX-Tor-2016 [28], which includes traffic from five categories such as *Browsing*, *File Transfer*, *P2P*, *Streaming*, and *VoIP*. For IoT/IIoT attack detection, we construct a balanced subset of ToN-IoT [29] by sampling flows evenly across seven classes, including *DDoS*, *Injection*, *Password*, *Ransomware*, *Scanning*, and *XSS*. For general-purpose intrusion detection, we adopt CICIDS-2017 [30] and classify flows into attack types such as *Botnet*, *DDoS*, and *DoS*, with majority-class undersampling applied to reduce training bias.

Comparison Schemes: We benchmark *Monic* against three representative in-network ML systems: NetBeacon [4], Planter [9], and DUNE [6]. Both NetBeacon and Planter are *monolithic* methods, while DUNE offers a *distributed* framework that conducts a complex inference task across multiple programmable devices.

B. Model Performance Comparison

Accuracy: Table I presents the results on three datasets using Micro, Macro, and Weighted F1-scores. *Monic* (with two experts and one gating) consistently outperforms NetBeacon, Planter, and DUNE across all metrics.

On the ToN-IoT dataset, *Monic* attains a Macro F1-score of 78.32%, representing a 13.2% improvement over DUNE (65.12%), the next-best model. Similar trends are observed on the CICIDS and ISCX-Tor datasets, where *Monic* achieves improvements of 7.81% and 3.32% in Macro F1-score, respectively, compared to DUNE and NetBeacon. These results demonstrate the robustness of *Monic* in maintaining high accuracy across diverse traffic patterns. With an offline training time of 14.19s on the ISCX-Tor dataset, *Monic* offers a practical trade-off between accuracy and efficiency, making it suitable for real-world deployment.

TABLE I
F1-SCORE COMPARISON ACROSS THREE DATASETS.

Dataset	F1-Score	NetBeacon	Planter	DUNE	Monic
CICIDS	**Micro**	97.77%	96.81%	95.76%	**98.48%**
	Macro	81.11%	80.00%	90.28%	**98.09%**
	Weighted	97.59%	96.61%	95.88%	**98.48%**
ToN-IoT	**Micro**	64.32%	71.21%	68.11%	**77.52%**
	Macro	61.44%	62.15%	65.12%	**78.32%**
	Weighted	63.20%	69.83%	68.67%	**77.99%**
ISCX-Tor	**Micro**	89.62%	86.13%	67.33%	**91.62%**
	Macro	70.97%	60.54%	54.33%	**74.29%**
	Weighted	88.82%	84.39%	74.22%	**90.79%**

To further investigate the source of *Monic*'s superior aggregate performance, we analyze per-class F1-scores on the ToN-IoT dataset, as shown in Fig. 6. This analysis highlights *Monic*'s architectural advantage in achieving consistent performance across diverse attack categories.

While other systems perform well on certain classes, they exhibit significant variability on others. For example, DUNE achieves high F1-scores on *DDoS* and *Injection* attacks but drops below 40% on *Ransomware*. Such inconsistencies may lead to undetected threats in practical deployments. In contrast, *Monic* maintains F1-scores above 60% across all categories and exceeds 80% on most. This balanced performance results from its MoE architecture, where the gating network dynamically selects the most appropriate expert(s) for each flow.

Performance under Strict Resource Constraints. To evaluate performance under resource scarcity, we constrain each model to a maximum of four features per switch, reflecting the limited memory and compute budgets of programmable data planes. The results under this constraint are shown in Table II. The data shows that *Monic*'s performance advantage becomes even more pronounced. On the ToN-IoT dataset, *Monic* achieves a Macro F1-score of 77.85%, now leading the next-best system (DUNE) by over 15%. Notably, *Monic* sustains F1-scores above 70% across all classes, whereas baselines exhibit significant drops, with several classes falling below 60%.

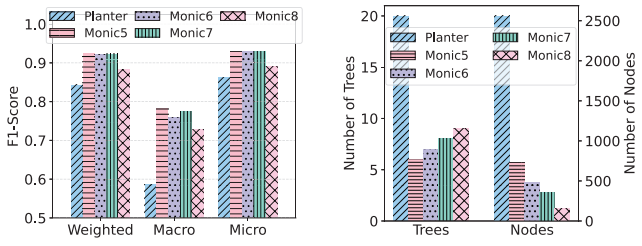
TABLE II
F1 SCORE UNDERING STRICT RESOURCE CONSTRAINTS.

Dataset	F1-Score	NetBeacon	Planter	DUNE	*Monic*
ToN-IoT	**Micro**	62.48%	67.26%	66.51%	**77.09%**
	Macro	56.20%	55.87%	62.73%	**77.85%**
	Weighted	57.22%	62.86%	66.29%	**77.51%**

This performance arises from *Monic*'s resource-aware MoE design, which selectively allocates feature budgets and compute across experts. Unlike monolithic architectures bound to globally fixed feature sets, *Monic* allows each expert to focus on the most informative features for its subtask. This adaptive strategy enables robust classification under severe hardware constraints, achieving graceful degradation and superior efficiency in resource-scarce environments.

Scalability of *Monic*. We evaluate *Monic*'s scalability by increasing the number of experts while reducing each tree's depth. Configurations include *Monic5*, *Monic6*, *Monic7*, and *Monic8*, where the suffix denotes the number of experts on

the ISCX-Tor dataset. Their respective tree heights are 7, 6, 5, and 4.

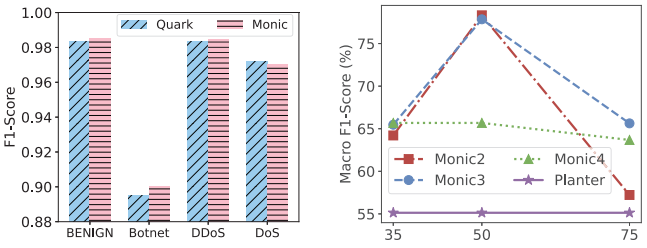


(a) Accuracy. (b) Tree size.

Fig. 7. The Scalability of *Monic*.

As shown in Fig. 7a, *Monic* maintains high accuracy across all configurations and consistently outperforms the Planter baseline. We further evaluate *Monic*'s resource efficiency by comparing the number of trees and nodes required under each configuration (Fig. 7b). While Planter uses 20 trees and 2560 nodes, *Monic5–Monic8* reduce node counts to 736, 483, 356, and 153, respectively. These results confirm that *Monic* can scale to larger expert ensembles without compromising performance, supporting flexible deployments under diverse hardware constraints.

Neural network inference in *Monic*. To demonstrate the architectural flexibility of *Monic*, we integrate it with Quark [8]. Each expert is a three-layer 7-bit quantized CNN with two hidden layers and a 4-class output layer. The gating network is implemented as a single-layer perceptron that maps an 8-dimensional feature vector to expert selection logits. The results, presented in Fig. 8a, show that *Monic* achieves performance that is overall comparable to the baseline Quark model. Notably, *Monic* improves detection of the minority *Botnet* class, achieving an F1-score of 0.9004 compared to 0.8953 for Quark, representing a 0.5% improvement.



(a) Quark (CNN) as experts. (b) Different feature overlap rates.

Fig. 8. The flexibility of *Monic*.

Different Feature Overlap in *Monic*. To validate *Monic*, we analyze the impact of its core parameters: the number of experts and their feature overlap. Fig. 8b plots the Macro F1-Score for 2, 3, and 4-expert configurations across varying overlap levels, benchmarked against Planter. The results show a clear performance peak at 50% feature overlap, where the 2-expert *Monic* achieves the highest F1-score of nearly 78%. At low overlap (35%), experts are overly specialized and lack shared context, degrading performance. At high overlap (75%), redundancy emerges, diminishing ensemble benefits. The 50% setting balances diversity and shared knowledge.

C. Performance in the P4 Hardware Switch

Resource usage. Table III reports the hardware resource consumption of *Monic* versus monolithic and distributed baselines. As a distributed system, *Monic* partitions its resource footprint across switches. The Gating-Unit, which orchestrates expert collaboration, incurs the highest cost in managing expert selection. Despite this, *Monic* demonstrates strong efficiency. In SRAM, its per-switch usage (e.g., 9.00%) is far lower than Planter's peak of 23.25%. These results highlight *Monic*'s favorable trade-off: a manageable resource investment yields state-of-the-art accuracy while preserving ample capacity for other data plane tasks.

TABLE III
RESOURCE CONSUMPTION COMPARISON.

		Benchmarks		Dune			Monic		
Dataset	**Resource**	NetBeacon	Planter	Sub1	Sub2	Sub3	EP1	EP2	Gating
ISCX-Tor	**TCAM**	4.86%	4.86%	4.17%	4.86%	4.17%	4.17%	4.17%	16.66%
	SRAM	5.75%	23.25%	8.25%	8.5%	4.25%	9.00%	9.00%	4.0%
ToN-IoT	**TCAM**	5.90%	5.90%	3.82%	4.17%	4.51%	4.17%	4.17%	8.33%
	SRAM	3.75%	14.00%	5.00%	5.00%	3.75%	9.00%	9.00%	2.5%

* Each sub-model (Sub) or expert (EP) is evaluated in one switch.

Latency comparison. Table IV reports end-to-end processing latencies (in ns) for *Monic* and baselines across two switches where one of those can implement multiple sub-models or experts. The No Inference case at ∼543 ns reflects pure forwarding latency. *Monic*'s total latency of 690∼692 ns equates to an extra negligible ∼50 ns per expert relative to inference on average.

TABLE IV
LATENCY ANALYSIS (NANOSECOND).

Dataset	No Inference	NetBeacon	Planter	DUNE	*Monic*
ISCX-Tor	543	572	602	656	692
ToN-IoT	543	573	604	653	690

VIII. CONCLUSION

This paper presents *Monic*, a collaborative in-network ML framework that leverages multiple experts for joint inference in programmable data planes. Experiments show that *Monic* outperforms existing methods in accuracy and resource efficiency, while maintaining scalability and deployment flexibility. This work highlights the potential of collaborative inference for advancing intelligent and high-performance network systems.

ACKNOWLEDGEMENT

This work is partially supported by National Natural Science Foundation of China (NSFC) under Grant 62172189, 62572217 and 62272050; the Innovate UK grants 10098627, 10186103 and 600648; Guangdong Basic and Applied Basic Research Foundation under Grant 2024A1515011869; Zhuhai Science-Tech Innovation Bureau under Grant 2320004002772 and the Interdisciplinary Intelligence Super Computer Center of Beijing Normal University (Zhuhai).

REFERENCES

[1] "Intel Tofino switch ASIC," 2025, https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino.html, accessed on Jul. 30, 2025.

[2] W.-X. Liu, C. Liang, Y. Cui, J. Cai, and J.-M. Luo, "Programmable data plane intelligence: Advances, opportunities, and challenges," *IEEE Network*, vol. 37, no. 5, pp. 122–128, 2023.

[3] Z. Zhang, Z. Luan, Q. Li, Z. Qi, K. Li, Y. Jiang, and Z. Yuan, "Sentinelx: A lightweight malicious traffic detection system based on programmable switches," in *IEEE Conference on Computer Communications (IEEE INFOCOM)*. IEEE, 2025, pp. 1–10.

[4] 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)*, 2023, pp. 6203–6220.

[5] 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*, vol. 18, no. 4, pp. 4353–4364, 2021.

[6] B. Bütün, D. D. A. Hernandez, M. Gucciardo, and M. Fiore, "DUNE: Distributed inference in the user plane," in *IEEE Conference on Computer Communications (IEEE INFOCOM)*. IEEE, 2025, pp. 1–10.

[7] G. Siracusano, S. Galea, D. Sanvito, M. Malekzadeh, G. Antichi, P. Costa, H. Haddadi, and R. Bifulco, "Re-architecting traffic analysis with neural network interface cards," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2022, pp. 513–533.

[8] M. Zhang, L. Cui, X. Zhang, F. P. Tso, Z. Zhen, Y. Deng, and Z. Li, "Quark: Implementing convolutional neural networks entirely on programmable data plane," in *IEEE Conference on Computer Communications (IEEE INFOCOM)*. IEEE, 2025, pp. 1–10.

[9] C. Zheng, M. Zang, X. Hong, L. Perreault, R. Bensoussane, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "Planter: Rapid prototyping of in-network machine learning inference," *ACM SIGCOMM Computer Communication Review*, vol. 54, no. 1, pp. 2–21, 2024.

[10] J. Lin, Q. Li, G. Xie, Y. Jiang, Z. Yuan, C. Jiang, and Y. Yang, "In-forest: Distributed in-network classification with ensemble models," in *IEEE 31st International Conference on Network Protocols (ICNP)*. IEEE, 2023, pp. 1–12.

[11] K. Zhang, C. Zheng, N. Samaan, A. Karmouch, and N. Zilberman, "Muta: enabling multi-task neural network inference in programmable data-planes," 2025.

[12] S. Lin, S. Xu, B. He, H. Liu, D. Kong, X. Chen, D. Zhang, C. Wu, M. Li, X. Liu *et al.*, "NDIF: A distributed framework for efficient in-network neural network inference," *Computers & Security*, p. 104593, 2025.

[13] K. Razavi, S. D. Fard, G. Karlos, V. Nigade, M. Mühlhäuser, and L. Wang, "NetNN: Neural intrusion detection system in programmable networks," in *29th IEEE Symposium on Computers and Communications (ISCC)*, 2024.

[14] W. Cai, J. Jiang, F. Wang, J. Tang, S. Kim, and J. Huang, "A survey on mixture of experts in large language models," *IEEE Transactions on Knowledge and Data Engineering*, vol. 37, no. 7, pp. 3896–3915, 2025.

[15] 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.

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

[17] A. T.-J. Akem, B. Bütün, M. Gucciardo, and M. Fiore, "Jewel: Resource-efficient joint packet and flow level inference in programmable switches," in *IEEE Conference on Computer Communications (IEEE INFOCOM)*. IEEE, 2024, pp. 1631–1640.

[18] H. Kim, S. Yoon, C. Bae, S. Lee, and S. Pack, "Tiniee: Traffic-aware adaptive in-network intelligence via early-exit strategy," in *21st Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 2024, pp. 10–18.

[19] E. M. Loiola, N. M. M. De Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido, "A survey for the quadratic assignment problem," *European journal of operational research*, vol. 176, no. 2, pp. 657–690, 2007.

[20] X. Zhang, L. Cui, F. P. Tso, and W. Jia, "Compiling service function chains via fine-grained composition in the programmable data plane," *IEEE Transactions on Services Computing*, vol. 16, no. 04, pp. 2490–2502, 2023.

[21] T. Li, Z. Wei, X. Chen, and Z. Zhu, "Mrgrecmp4: Efficient stateful sfc recompilation in pdp switches with flexible table merging," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2025, pp. 1–10.

[22] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[23] M. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: evidence and possible causes," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 835–846, 1997.

[24] J. Kepner, K. Cho, K. Claffy, V. Gadepally, P. Michaleas, and L. Milechin, "Hypersparse neural network analysis of large-scale internet traffic," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, 2019, pp. 1–11.

[25] X. Gabaix, "Zipf's law for cities: An explanation*," *The Quarterly Journal of Economics*, vol. 114, no. 3, pp. 739–767, 08 1999. [Online]. Available: https://doi.org/10.1162/003355399556133

[26] "mininet/mininet," https://github.com/mininet/mininet, accessed on Jul. 30, 2025.

[27] "p4lang/behavioral-model," https://github.com/p4lang/behavioral-model, accessed on Jul. 30, 2025.

[28] E. Biglar Beigi, H. Hadian Jazi, N. Stakhanova, and A. A. Ghorbani, "Towards effective feature selection in machine learning-based botnet detection approaches," in *IEEE Conference on Communications and Network Security*, 2014, pp. 247–255.

[29] N. Moustafa, "A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets," *Sustainable Cities and Society*, vol. 72, p. 102994, 2021.

[30] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani *et al.*, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *International Conference on Information Systems Security and Privacy (ICISSp)*, vol. 1, 2018, pp. 108–116.