Between Packets and Predictions: Analyzing the Overhead of In-Network Machine Learning in Programmable Switches

Caio Luiz L. T. Silva Icaro M. da Silva Thiago Gouveia e Leandro C. de Almeida

[1]Academic Unit of Informatics – Federal Institute of Paraíba (IFPB)
João Pessoa – PB – Brazil

{caio.luiz, icaro.silva}@academico.ifpb.edu.br,

{thiago.gouveia,leandro.almeida}@ifpb.edu.br

Abstract. *Recent advances in programmable data planes have sparked significant interest in executing machine learning inference directly within network devices. Prior work demonstrates the feasibility of this paradigm across diverse applications, ranging from Network Security to Quality of Service. However, the network-level performance overhead and effects in perceived quality remain largely unexplored. This paper assesses the impact of executing machine learning inference inside network devices on key metrics such as throughput, queueing delay, and application quality. Through controlled experiments covering web traffic, bulk file transfer, and video streaming, the performance of machine-learning-enabled pipelines is contrasted with standard forwarding. Specific attention is given to adaptive video delivery, where MPEG-DASH metrics indicate that while adaptive mechanisms preserve playback stability, the integration of additional processing features leads to a saturation of quality gains and a marked increase in tail latency. The findings provide valuable insights into the systemic costs of embedding intelligence into smart-switch data planes.*

1. Introduction

Network devices such as switches and routers have historically been designed with a single, well-defined goal: forwarding packets efficiently and deterministically. Their architectures prioritize simple operations, deep pipelining, and massive parallelism in order to sustain high throughput and low latency under heavy traffic loads. As a result, complex processing and decision-making tasks have traditionally been offloaded to external systems, typically residing in the control plane or on dedicated servers.

Recent advances in programmable data planes have fundamentally altered this design paradigm. Languages such as P4 [Bosshart et al. 2014] enable fine-grained control over packet-processing logic, allowing developers to implement custom functionalities directly within the forwarding pipeline. This flexibility has sparked growing interest in executing Machine Learning (ML) inference inside network devices [Boutaba et al. 2018], a paradigm commonly referred to as In-Network Machine Learning (In-Network ML) [Xiong and Zilberman 2019]. By enabling real-time data-plane decisions, In-Network ML provides faster reaction times and reduced dependence on centralized processing.

However, ML models are inherently computationally intensive, particularly during the training phase. In contrast, network devices operate under strict resource constraints and rely on highly specialized hardware architectures. Consequently, most In-Network ML approaches adopt a split-execution model in which training is performed offline in the control plane, while inference is deployed online in the data plane [Zheng et al. 2024a]. This separation enables the practical deployment of ML models, but requires significant adaptation to fit the constraints of programmable switches.

To accommodate these constraints, recent work [Xiong and Zilberman 2019, Zhang et al. 2024, Fatur Lessa et al. 2025] maps trained ML models onto match-action table (MAT) pipelines, representing inference logic as a sequence of table lookups and actions. This mapping may introduce additional processing stages that every packet must traverse, regardless of whether inference is strictly necessary.

In this context, these additional stages alter the standard packet-processing path. Each MAT associated with the ML model increases the depth of the pipeline, introducing an additional processing delay and internal queueing. Although such overheads may appear negligible in isolation, their cumulative effect can interfere with the switch's core forwarding function, especially under high traffic loads. This raises a critical concern regarding the execution of ML within devices originally designed for packet forwarding. The overhead introduced by In-Network inference can increase device-level resource consumption and negatively impact network performance, affecting key metrics such as throughput, latency, and round-trip time (RTT). These effects are particularly harmful to latency-sensitive applications, including real-time media streaming and interactive services.

To better understand these trade-offs, this paper evaluates the performance overhead introduced by In-Network Machine Learning in programmable switches. Rather than focusing solely on model accuracy, we quantify the impact of in-network inference on network-level performance across multiple application scenarios, including web browsing, file transfer, and video streaming. Our goal is to characterize the true cost of embedding intelligence into the data plane and to evaluate whether the benefits of In-Network ML outweigh its impact on network performance.

To this end, this work presents an exploratory study on the impacts of executing ML inference in programmable data planes. We evaluated the network-level overhead introduced by In-Network ML by quantifying its influence on throughput, queue delay, and application performance under different traffic scenarios. By contrasting ML-enabled pipelines with baseline forwarding behavior, we provide empirical insights into the trade-offs between model execution and packet forwarding, thereby helping to clarify when and to what extent ML should be deployed within the data plane.

The remainder of this work is organized as follows. Section 2 presents the problem statement and outlines the challenges associated with executing machine learning inference in programmable data planes. Section 3 reviews related work on In-Network ML and programmable switch architectures. Section 4 describes the experimental methodology and evaluation results, as well as the impact of inference on network-level performance. Finally, Section 5 reflects on the experimental outcomes and concludes the paper with a discussion on future work.

2. Problem Statement

As introduced previously, the dominant approach for deploying ML models in programmable data planes relies on translating a pre-trained model into a set of match-action tables and instructions that can be executed in the switch pipeline. Under this strategy, training is performed offline using conventional ML frameworks, and the resulting decision logic is statically mapped onto the data plane.

Once deployed, these match-action tables become part of the forwarding pipeline and are executed for every packet traversing the device. Unlike control-plane mechanisms that operate selectively or asynchronously, data-plane inference logic is applied uniformly, regardless of traffic type or application requirements. As a result, all packets are subjected to additional processing stages introduced to support machine learning inference. In a baseline configuration, where the switch performs only traditional forwarding operations, the per-packet processing cost can be modeled as

$$C_{\text{base}} = C_{\text{parser}} \oplus (C_1 \oplus C_2 \oplus \cdots \oplus C_n) \oplus C_{\text{deparser}} , \tag{1}$$

where $\oplus$ stands for an abstract operator that combines two costs; C_{parser} and C_{deparser} represent the parsing and packet reconstruction costs, respectively, and $C_1 \cdots C_n$ denotes the processing costs of the match-action stages required for packet forwarding, such as CPU and memory usage, queueing delay, and throughput. This pipeline is dimensioned to sustain high throughput while ensuring deterministic latency and minimal buffering.

Each additional ML-related table increases the number of programmable blocks that packets must traverse internally. Although each block is designed to operate at line rate, their cumulative effect increases pipeline depth and reduces processing slack. This accumulation of stages is expected to embed multiple micro decision-making processes into the packet forwarding path, altering the characteristics of the switch pipeline. With In-Network ML enabled, the per-packet processing cost becomes

$$C_{\text{ML}} = C_{\text{base}} \oplus (C^*_{n+1} \oplus C^*_{n+2} \oplus \cdots \oplus C^*_{n+m}) , \tag{2}$$

where m denotes the number of additional match-action stages introduced by the machine learning model, and C^*_i represents the processing cost of the i-th ML-related stage. In the switch architecture, these stages consume limited pipeline slots and execution budget, directly increasing pipeline depth.

Figure 1 illustrates the conventional paradigm for ML integration in programmable data planes and identifies the sources of overhead inherent to this architecture. Figures 1(a–e) characterize the offline phase, which includes data collection and preprocessing (a), model training (b–c), conversion to the P4 language (c–d), compilation, and static mapping onto match-action tables (e). Conversely, Figure 1(f) depicts the online inference phase, where each packet traverses the parser, a sequence of MATs, and the deparser. Within this pipeline, the base MATs handle standard packet forwarding, while the ML-specific MATs are dedicated to inference logic. Consequently, the total resources required increase due to the cumulative execution of these ML-specific stages, as formally defined in Equations (1) and (2).

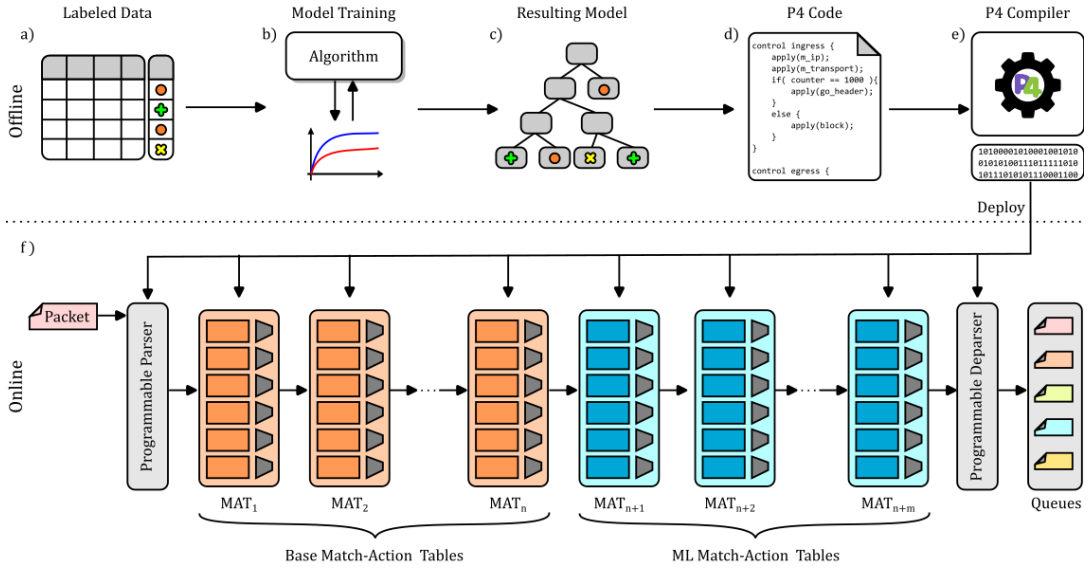


Figure 1. Detailed overview of the problem: (a) input data, (b) model training, (c) resulting model, (d) P4 code, (e) P4 deployment, and (f) the resulting pipeline.

The core challenge addressed in this work is to understand how this accumulation of in-network inference stages translates into measurable overhead. Specifically, we aim to quantify how these additional decision points affect packet processing time, internal queueing behavior, and transmission delays. When the associated costs are scalars, this overhead is expressed as a percentage as follows:

$$100 \times (C_{\text{ML}} - C_{\text{base}})/C_{\text{base}} \ . \tag{3}$$

While prior work has demonstrated that such pipelines are functionally correct and capable of delivering accurate predictions, their impact on forwarding performance remains insufficiently understood.

3. Related Work

Recent advances in programmable data planes have enabled the execution of ML inference directly within network devices, giving rise to In-Network ML. Previous work has demonstrated the feasibility of mapping trained ML models into P4 pipelines, achieving high classification accuracy across a range of applications, despite the tight constraints of programmable switching hardware. However, these studies primarily focus on model performance and resource efficiency, leaving the network-level performance overhead of in-network inference unexplored, a gap that motivates this work.

The early work in this domain focused on basic viability. Specifically, [Xiong and Zilberman 2019] investigate the feasibility of deploying pre-trained machine learning models directly on commodity switches, introducing IIsy, a framework that maps multiple models trained with Scikit-learn[1] into P4 match-action pipelines. By demonstrating in-network classification in practice, the study provides insights into the resource constraints and operational limitations of programmable switches.

[1]https://scikit-learn.org/stable/

Building on this foundation, to address model diversity, [Zheng et al. 2024b] extends IIsy to support a broader set of ML models, including K-means, Naive Bayes, Random Forests, SVM, and XGBoost, targeting off-the-shelf programmable switches. The work proposes a hybrid In-Network ML architecture in which most decisions are handled by a lightweight model in the data plane, while low-confidence or ambiguous cases are deferred to a more complex backend. While this hybrid approach optimizes decision-making, it still faces the challenge of fitting complex logic into rigid hardware. Addressing this, INQ-MLT [Zhang et al. 2024] utilizes quantization-aware training to ensure that ML models are natively optimized for the bit-width constraints of intelligent data planes.

However, even with optimized individual models, the overhead of running multiple independent tasks in data plane remains a bottleneck. To mitigate this cost, [Zhang et al. 2025b] introduces MUTA, a multi-task learning framework that enables concurrent inference tasks through shared model layers mapped onto programmable switches under the PISA architecture. Its distributed extension [Zhang et al. 2025a] further allows model layers to be offloaded across multiple switches, improving resource efficiency and scalability without hardware modifications.

Expanding the scope from stateless packet inspection to stateful flow analysis, [Akem et al. 2023] introduce the framework Flowrest, which enables flow-level Random Forests on commercial switches by integrating hardware constraints directly into the model design phase. Nevertheless, purely flow-based models leave initial packets unclassified while the state is gathered. To address this, [Akem et al. 2024] propose Jewel, a hybrid framework that trains a single model to handle both stateless and stateful contexts. Jewel utilizes packet-level features for early traffic and seamlessly transitions to flow-level features as the state accumulates, achieving higher accuracy and consistency than disjoint strategies.

Following the establishment of the architectural foundations for In-Network ML, concurrent studies have assessed its efficacy across domain-specific applications. In the context of Network Security, [Musumeci et al. 2022] evaluates ML-based DDoS-attack detection deployed directly on programmable switches. The study considers both standalone and hybrid detection architectures, achieving high accuracy and low classification latency, demonstrating the benefits of in-data-plane feature extraction for real-time detection. Similarly, [Nguyen et al. 2024] propose a comprehensive framework to offload ML inference tasks from central servers to P4-capable devices, addressing the latency and privacy challenges inherent in data-intensive monitoring, particularly within IoT environments.

In the domain of Quality of Service (QoS), [Shirmarz et al. 2025] deploy Random Forest models on a Tofino ASIC to classify augmented reality and cloud gaming traffic, integrating with L4S mechanisms via ECN and DSCP marking. The evaluation shows minimal overhead and effective prioritization, while highlighting potential QoS impacts from misclassified or unclassified traffic. Finally, whereas most previous work targets plaintext traffic, [Akem et al. 2025] address encrypted-traffic classification by fully integrating tree-based models into the data plane. Evaluated on real-world testbeds with Tofino switches, the proposed solution achieves near-optimal accuracy on QUIC traffic with sub-microsecond latency and modest hardware resource consumption.

In summary, existing work demonstrates that In-Network ML is feasible and effective under hardware constraints, with a strong focus on accuracy and resource efficiency. Further, this paradigm has been successfully applied in domains such as network security, QoS, and IoT. However, the network-level performance impact of in-network inference remains largely unexplored, which this work addresses through a systematic overhead evaluation, given in Section 4.

4. Evaluation

This section describes the experimental artifacts and system components used in the evaluation performed, to enable controlled experimentation and ensure reproducibility.[2]. The testbed is instantiated using virtual machines orchestrated with VirtualBox[3], Vagrant[4] and Ansible[5]. The network topology comprises three virtual machines: two end hosts acting as client and server, and a central node running a programmable P4 switch (BMv2)[6]. Hereafter, these nodes are referred to as client, switch, and server, respectively.

Building on this testbed, the evaluation is structured into two experiments considering three traffic classes: *web traffic*, *file transfer*, and *video streaming*. First, the impact of In-Network ML inference on different application classes is examined, with an emphasis on network-level performance metrics. The second experiment focuses exclusively on the adaptive video streaming service MPEG-DASH and evaluates the impact of In-Network ML under increasing model complexity. This analysis aims to characterize the trade-off between model complexity and video quality, highlighting how possible gains in accuracy may come at the cost of degraded quality of service.

4.1. Description of Experiment 1

The first experiment evaluates how different application classes are affected by the presence of In-Network ML. To this end, a baseline configuration, in which packet forwarding is performed without in-network inference, is compared against a configuration where In-Network ML is enabled. The evaluation considers three scenarios with different application classes:

Web traffic: This scenario is implemented by deploying an *Nginx*[7] HTTP server on the destination host, which serves a directory of interconnected HTML pages. To emulate user browsing behavior, the client host executes an automated shell script that uses the `curl` utility to retrieve web content. Starting from an initial landing page, the script parses the document's anchor tags to identify available hyperlinks and recursively traverses the website structure, generating realistic web traffic patterns.

File transfer: This scenario is implemented by configuring the server as a file repository containing datasets categorized by file type (plain text and binary) and size (1 MB, 2 MB, and 20 MB). In addition, all test files are aggregated into a single compressed archive using the `tar` utility. On the client side, an automated script issues a sequence of Secure Copy Protocol (`scp`) commands to download individual files as well as the aggregated archive from the server, generating representative encrypted bulk transfer traffic.

[2]https://github.com/ifpb/In-NetworkML
[3]https://www.virtualbox.org/
[4]https://developer.hashicorp.com/vagrant
[5]https://docs.ansible.com/
[6]An adapted version for higher throughput. Available in: https://github.com/p4lang/behavioral-model/blob/main/docs/performance.md
[7]https://nginx.org/

Video streaming: This scenario is implemented by configuring the server to stream an MP4 video file using the `ffmpeg` tool over the Real-Time Messaging Protocol (RTMP). The stream is configured with the *stream_loop* option, enabling continuous playback by looping the video upon reaching the end of the file. On the client side, `ffmpeg` is used to receive and consume the stream, while simultaneously generating a log of statistical metadata throughout the streaming session. This setup allows us to capture video-related performance indicators under varying network conditions.

4.2. Design of Experiment 1

As discussed previously, Experiment 1 evaluates the impact of In-Network ML by comparing a baseline configuration, in which packets are forwarded through a conventional P4 pipeline without inference logic, against an intelligent configuration that embeds ML-based classification into the data plane. In the intelligent setup, model training is conducted offline in the control plane using labeled traffic traces, while the trained model is deployed for online inference within the programmable switch. We adopt a Decision Tree classifier due to its structural compatibility with match-action tables (MATs), which enables a direct and efficient mapping of decision nodes to P4 tables. Further, in a broader ML landscape, Decision Trees serve as the foundational learners for widely adopted ensemble methods such as Random Forests, XGBoost, and LightGBM.

The ML component is developed following a two-phase pipeline implemented in Python. In the first phase, one `pcap` file from each traffic scenario is processed to build an unified dataset in CSV format. Packets without IPv4 or TCP headers are excluded. For each remaining packet, fields from the Ethernet, IPv4, and TCP headers are extracted as features, and a categorical label is assigned according to the traffic class: *web browsing* (0), *file transfer* (1), or *video streaming* (2). The model follows a hybrid inference design that combines packet-level and flow-level features, leveraging the strengths of both approaches, as discussed in Section 3. The inference is driven by a joint feature set that includes per-packet fields, type of service (ToS), and packet size, as well as flow-derived features such as TCP window size and inter-arrival time (IAT).

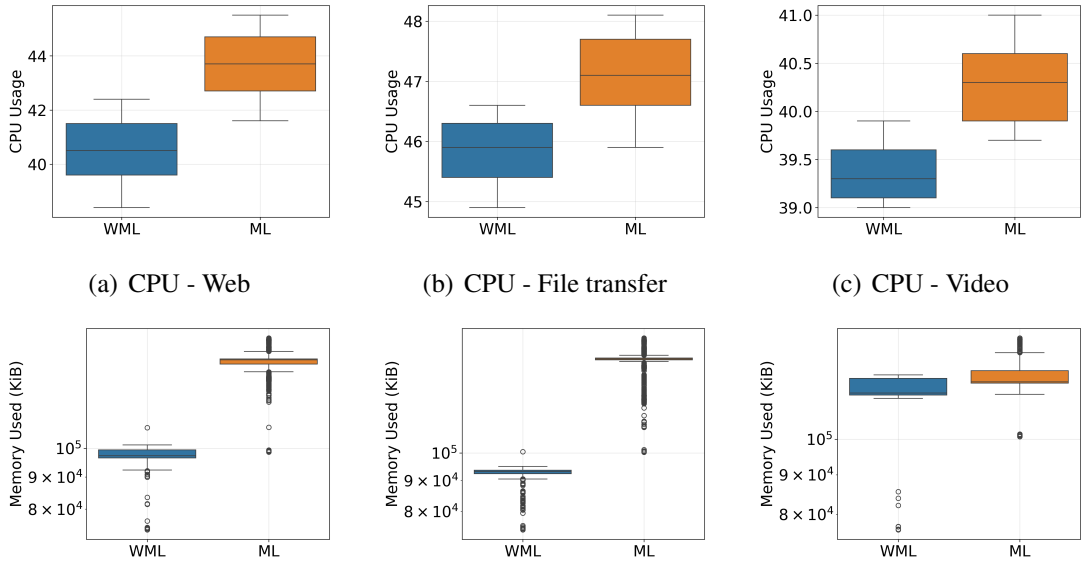
In the second phase, the dataset is cleaned by removing records with missing values and eliminating redundant entries. The resulting dataset is used to train a Decision Tree classifier using the *Scikit-learn* library. The trained model is then translated into P4-compatible match-action rules and deployed in the data plane to enable online inference during the evaluation. After training, the model is translated into a P4-compatible representation by extracting the selected features and their corresponding thresholds. These parameters are processed by a custom Python script in the control plane, which automatically generates both the P4 source code and the MAT entries required for deployment on the BMv2 switch.

The code generation process is based on a modified version of the IIsy framework [Xiong and Zilberman 2019], leveraging its software-based Decision Tree implementation as a template. For each feature used by the model, the script generates a corresponding MAT with range-based matches that encode the model thresholds. The outcomes of these tables are stored in metadata and subsequently combined by a final classification table to determine the final classification of the packet. The generated MATs are populated at runtime using the P4 switch API, allowing seamless alignment between the trained model and the data-plane behavior during evaluation.

4.3. Results of Experiment 1

This section presents the results of Experiment 1, which evaluates the system-level impact of integrating ML inference directly into the programmable data plane. The In-Network ML approach (ML) is compared against a conventional P4 baseline without inference logic (WML), focusing on key system metrics to quantify the overhead introduced by data-plane classification.

Figures 2(a), 2(b), and 2(c) show the switch CPU utilization under the evaluated workloads. Relative to the WML baseline, In-Network ML increases the median CPU usage by 7.3% for web traffic, 2.5% for file transfer, and 2.4% for video streaming. Figures 2(d), 2(e), and 2(f) report the switch memory utilization. Compared to WML, In-Network ML introduces median memory overheads of 29.6% for web traffic, 34.9% for file transfer, and 3.29% for video streaming.



(a) CPU - Web (b) CPU - File transfer (c) CPU - Video

(d) Memory - Web (e) Memory - File transfer (f) Memory - Video

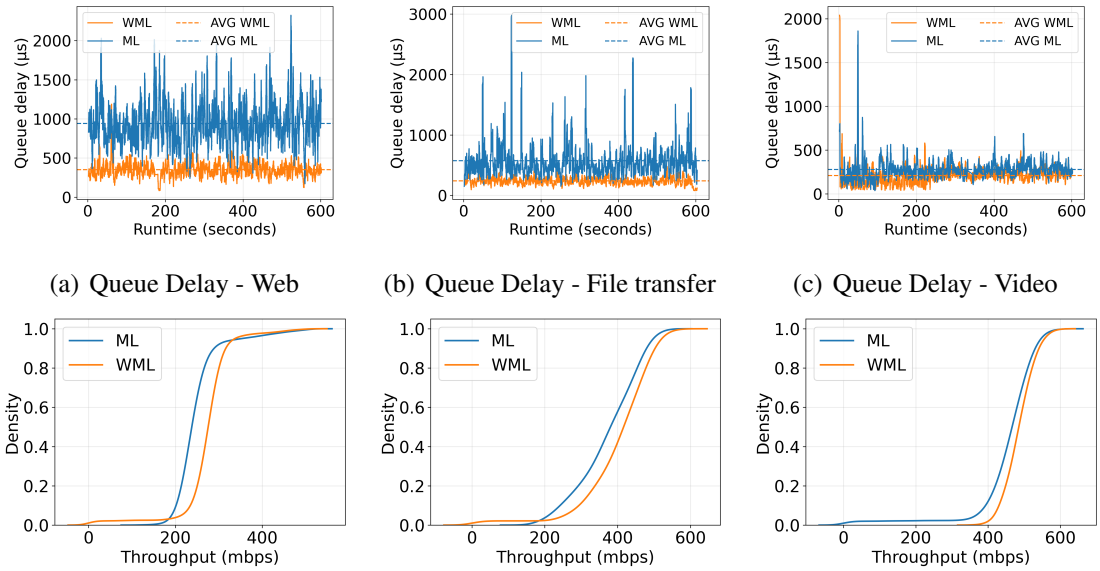
Figure 2. Device-level impact on CPU (a-c) and memory indicators (d-f) for web, file transfer, and video scenarios, respectively.

We now turn our attention to network-level performance metrics, focusing on queueing delay and throughput. While previous results quantified the system overhead of executing In-Network ML within the data plane, this analysis aims to understand how in-network inference affects the network's behavior itself. By examining delay accumulation at the switch queues and the achieved end-to-end throughput for web browsing, file transfer, and video streaming traffic, we evaluate if the additional processing introduced translates into measurable performance degradation from the perspective of network flows and applications.

Figures 3(a), 3(b), and 3(c) report the queueing delay observed at the programmable switch for web, file transfer, and video streaming traffic, respectively. When compared to the WML baseline, the In-Network ML configuration introduces a substantial increase in median queueing delay, with relative differences of 70.4% for web traffic, 51.4% for file transfer, and 31.3% for video streaming. Also, the coefficient of

variation increases substantially. These results indicate that, although the computational overhead of in-network inference at the device level is moderate, the additional stages in the data-plane pipeline significantly affect packet residence time in switch queues.

Figures 3(d), 3(e), and 3(f) present the Cumulative Distribution Functions (CDFs) of throughput for web browsing, file transfer, and video streaming, respectively. Consistent with the increased queueing delay, the In-Network ML configuration leads to a reduction in median throughput of 17.1% for web traffic, 10.2% for file transfer, and 4.1% for video streaming when compared to the WML baseline. This throughput degradation directly follows from the longer packet residence times induced by the expanded data-plane pipeline, which lowers the service rate of the switch under load.



(a) Queue Delay - Web (b) Queue Delay - File transfer (c) Queue Delay - Video

(d) Throughput - Web (e) Throughput - File transfer (f) Throughput - Video

Figure 3. Network-level impact on queueing delay (a-c) and throughput indicators (d-f) for web, file transfer, and video scenarios, respectively.

We conjecture that this overhead stems from the additional data-plane stages and MATs required to execute inference logic per packet. While the Decision Trees increase the number of tables traversed, the resulting CPU impact remains modest, whereas the memory requirements show a moderate increase across web and file transfer scenarios. The influence of increasing model complexity examined in Experiment 2.

4.4. Description of Experiment 2

Experiment 2 extends the evaluation of In-Network ML by investigating whether increasing model complexity affects the switch system metrics and the quality perceived by users of a latency-sensitive application, MPEG-DASH[8]. Specifically, this experiment examines the implications of progressively more complex models, implemented as decision trees with a growing number of features, on CPU, memory usage, and end-to-end application performance. At this point, it is important to recall how the hyperparameters that control the complexity of decision trees, specifically the number of nodes and the number of features, map into the P4 pipeline, as shown in Figure 4.

[8]https://www.mpeg.org/standards/MPEGDASH/

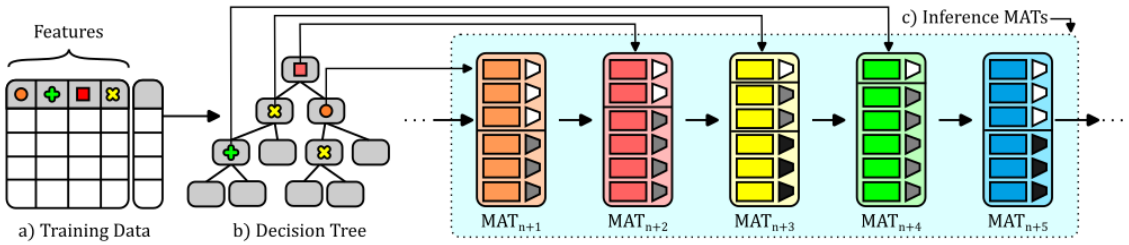


Figure 4. Decision Trees: from data to MATs. (a) Training data. (b) Resulting decision tree. (c) Pipeline stages that implement the model logic.

Figure 4(a) illustrates the input data and highlights the feature set. After training, Figure 4(b) presents the resulting decision tree, which utilizes all 4 features and has a depth of 3 with 11 nodes. Finally, Figure 4(c) depicts the pipeline stages that implement the model logic, where a dedicated MAT is assigned to each feature. Each MAT encodes the input ranges of its corresponding feature into auxiliary variables, and a final MAT aggregates these variables to produce the classification decision. It is worth noting that the impact of the number of nodes is negligible as the number of stages depends solely on the number of features of the tree.

With the architectural implications and system-level impact established, Experiment 2 investigates whether the gains in model expressiveness translate to improved video playback quality or whether excessive complexity adversely impacts the user experience. To this end, the relationship between model complexity and playback stall behavior is analyzed as a function of the number of features.

4.5. Design of the Experiment 2

The setup adopted in Experiment 2 largely follows the experimental workflow defined in Experiment 1 (Section 4.2), preserving the network topology, In-Network ML configuration, and measurement procedures to ensure comparability. The main difference lies in the workload generation: first, a controlled traffic is generated using `ffmpeg`, followed by a real application-driven load produced by the WAVE framework [Beuttenmuller et al. 2025].

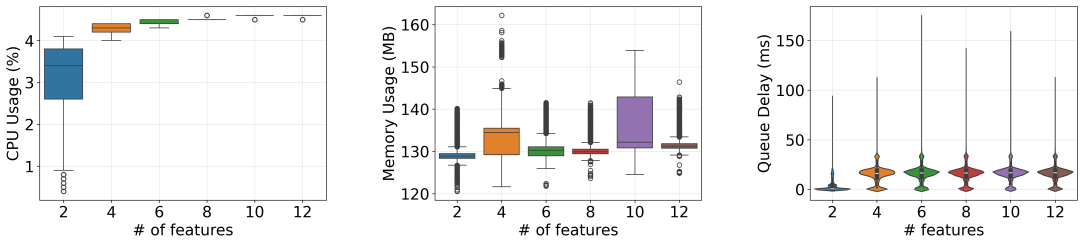
In this experiment, WAVE is configured to generate a sinusoidal load pattern, characterized by alternating peak and valley periods, which emulates realistic fluctuations in user demand. The wave function controls the number of active clients watching video simultaneously. This dynamic load deliberately stresses the network and forces the MPEG-DASH to react by adjusting the video quality. Follows the configuration of WAVE: $\{amplitude{=}15; period{=}5; \lambda{=}5; duration{=}10\}$, resulting in two sine wave cycles centered on 30 video clients, varying from 15 to 45 clients over a 10-minute interval.

4.6. Results of Experiment 2

The initial investigation of Experiment 2 (Figure 5) utilizes a fixed load produced with `ffmpeg` to evaluate how the decision tree feature count affects switch resource overhead. Detailed breakdowns of CPU load, memory footprint, and queuing delay are provided in Figures 5(a), 5(b), and 5(c), respectively.

Observe that as the number of features grows, the CPU load increases only about

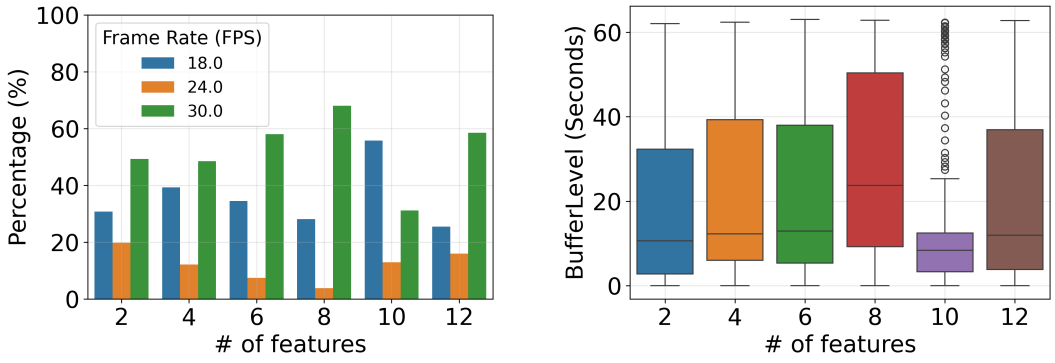
1%, stabilizing around 4.5%. Despite a wider range of values, a similar trend is evident in memory usage, which is centered around 130 MB. The behavior is slightly different with 4 and 10 features, probably due to buffer fluctuations. Finally, despite the long tails, queueing delay remains negligible regardless of the feature count. These results corroborate the findings of Experiment 1, demonstrating that even complex trees incur only a marginal increase in CPU load and moderate memory consumption.



(a) CPU usage (b) Memory usage (c) Queueing delay

Figure 5. Device-level impact of a fixed workload generated using ffmpeg. (a) CPU load. (b) Memory footprint. (c) Queueing delay.

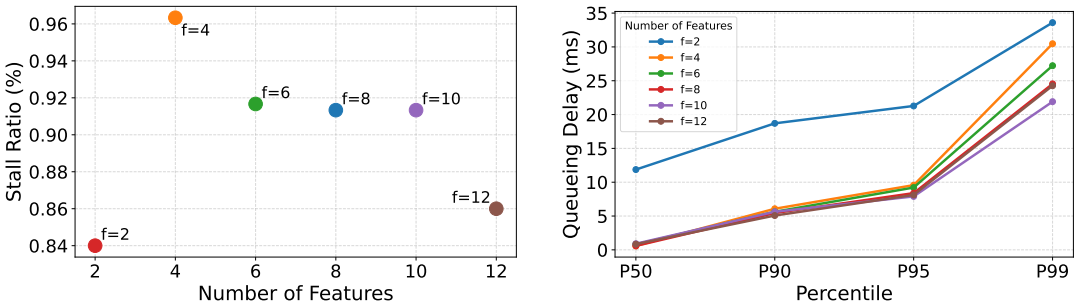
Shifting focus to video metrics, specifically FPS (Frames per Second) and buffer level, Figure 6 details how the feature count influences perceived video quality. It is important to recall that MPEG-DASH dynamically switches between three quality configurations (30, 24, and 18 FPS), what directly affects other metrics. Figure 6(a) reports the percentage of time spent in each FPS configuration. Due to WAVE-induced network fluctuations, the video transitioned among all three quality levels. However, playback occurred predominantly at maximum quality across all feature counts, except for the 10-feature configuration. In turn, Figure 6(b) details the client buffer level. These values indicate that MPEG-DASH successfully maintained buffer stability by adapting playback quality. Specifically, with the 10-feature tree, the system maintained the video at minimum quality for an extended duration to replenish the buffer.



(a) Frames per second (b) Buffer level

Figure 6. Quality-level impact on MPEG-DASH video streams under a sinusoidal workload generated using WAVE. (a) Frames per second (FPS). (b) Buffer level.

While FPS and buffer metrics provide insight into adaptation dynamics, they do not directly capture playback interruptions. To quantify this aspect of QoE, Figure 7(a)



(a) Stall Ratio (b) Queue Delay

Figure 7. Impact of the number of features on QoE and tail latency. (a) Playback stall ratio versus number of features, showing non-linear behavior and saturation of QoE gains. (b) Queueing delay percentiles (P50–P99), indicating that additional features primarily increase tail latency.

shows the playback stall ratio as a function of the number of features used by the model. The results show that increasing the feature set does not lead to consistent improvements in QoE. Although intermediate configurations slightly reduce the stall ratio, the gains quickly saturate, and larger feature sets may even degrade playback stability. This trend indicates that adding features beyond a moderate set offers limited benefit and underscores the importance of careful feature selection rather than indiscriminately increasing input dimensionality.

To understand the root cause of these stalls, we analyze the network-level queueing behavior. Figure 7(b) reports the queueing delay percentiles (P50, P90, P95, and P99) for different feature configurations. The median delay (P50) remains relatively stable across all cases, suggesting that the average queueing behavior is largely unaffected by the number of features. However, higher percentiles exhibit a markedly different pattern: delays at P95 and P99 increase substantially as more features are added, revealing a clear amplification of tail latency. This divergence between median and tail behavior indicates that additional features primarily affect rare but critical congestion events rather than typical operating conditions.

5. Conclusions

In this work, we presented a quantitative evaluation of the overhead introduced by executing machine learning inference directly in programmable data planes. We evaluated the In-Network ML approach against conventional forwarding behavior, analyzing how inference impacts system-level metrics, network performance, and service-level indicators. Rather than focusing on model accuracy, our study emphasized the cost of embedding intelligence into the packet-processing path and its implications under realistic traffic conditions.

The results indicate that the initial overhead of In-Network ML, measured across three distinct classes of applications, is moderate when inference logic is first introduced into the forwarding pipeline. In this preliminary evaluation, we observed a noticeable but limited increase in switch resource utilization and in the consumption of available network bandwidth. As model complexity increases, however, the additional overhead becomes marginal, suggesting that data-plane inference execution scales favorably within

the evaluated range. Moreover, the experiments show that traffic load and workload dynamics have a greater impact on performance degradation than model complexity itself. Under fluctuating and high-load conditions, queueing behavior and tail latency are primarily driven by the offered load rather than by the number of features or decision stages introduced by the model.

A natural path for future works is to extend this evaluation to additional classes of ML models, including ensemble and neural-network-based approaches, in order to generalize our findings beyond decision trees. Furthermore, while this study relied on software-based programmable switches, an important next step is to validate these results on production-grade programmable hardware, such as Tofino ASICs and SmartNICs, to evaluate if the observed trends persist under hardware-accelerated data-plane execution.

Acknowledgments

This work was partially supported by CNPq under grant no. 404509/2025-8 and IFPB.

References

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

Akem, A. T.-J., Fraysse, G., and Fiore, M. (2025). Real-time encrypted traffic classification in programmable networks with p4 and machine learning. *International Journal of Network Management*, 35(1):e2320.

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

Beuttenmuller, D., Valério, M., Silva, C., da Silva, I., Jr., P. D. M., and de Almeida, L. (2025). A new wave: Exploring new load pattern models for experimentation in computer networks. In *Anais Estendidos do XLIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 11–22, Porto Alegre, RS, Brasil. SBC.

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., and Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):1–99.

Fatur Lessa, J. V., Marques, J. A., Parizotto, R., and Gaspary, L. P. (2025). In-network inference with neuralp4: Auto-generating full-fledged nn models for pdps. In *NOMS 2025-2025 IEEE Network Operations and Management Symposium*, pages 1–10.

Musumeci, F., Fidanci, A. C., Paolucci, F., Cugini, F., and Tornatore, M. (2022). Machine-learning-enabled ddos attacks detection in p4 programmable networks. *Journal of Network and Systems Management*, 30(1):21.

Nguyen, H. N., Nguyen, M.-D., and Montes de Oca, E. (2024). A framework for in-network inference using p4. In *Proceedings of the 19th International Conference on Availability, Reliability and Security*, pages 1–6.

Shirmarz, A., Bragatto, M. N., Verdi, F. L., Singh, S. K., Rothenberg, C. E., Patra, G., and Pongrácz, G. (2025). In-network ar/cg traffic classification entirely deployed in the programmable data plane: Unlocking rtp features and l4s integration. In *2025 IEEE 11th International Conference on Network Softwarization (NetSoft)*, pages 477–485.

Xiong, Z. and Zilberman, N. (2019). Do switches dream of machine learning? toward in-network classification. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, HotNets '19, page 25–33, New York, NY, USA. Association for Computing Machinery.

Zhang, K., Samaan, N., and Karmouch, A. (2024). A machine learning-based toolbox for p4 programmable data-planes. *IEEE Transactions on Network and Service Management*, 21(4):4450–4465.

Zhang, K., Zheng, C., Samaan, N., Karmouch, A., and Zilberman, N. (2025a). Design, implementation, and deployment of multi-task neural networks in programmable data-planes. *IEEE Transactions on Network and Service Management*, pages 1–1.

Zhang, K., Zheng, C., Samaan, N., Karmouch, A., and Zilberman, N. (2025b). Muta: Enabling multi-task neural network inference in programmable data-planes. In *2025 IEEE 26th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–6.

Zheng, C., Hong, X., Ding, D., Vargaftik, S., Ben-Itzhak, Y., and Zilberman, N. (2024a). In-network machine learning using programmable network devices: A survey. *IEEE Communications Surveys & Tutorials*, 26(2):1171–1200.

Zheng, C., Xiong, Z., Bui, T. T., Kaupmees, S., Bensoussane, R., Bernabeu, A., Vargaftik, S., Ben-Itzhak, Y., and Zilberman, N. (2024b). Iisy: Hybrid in-network classification using programmable switches. *IEEE/ACM Transactions on Networking*, 32(3):2555–2570.