Binocular: Dual-Plane Anomaly Detection with In-Switch Inference and Control-Plane Refinement

BINOCULAR: Dual-Plane Anomaly Detection with In-Switch Inference and Control-Plane Refinement

Simone Geraci * † Lorenzo Pappone† Alessio Sacco* Flavio Esposito†
† *Department of Computer Science*, Saint Louis University, USA
* *Department of Control and Computer Engineering*, Politecnico di Torino, Italy

***Abstract*—Modern networks increasingly operate at line rate, yet most Intrusion Detection Systems (IDSs) still rely on off-path servers to analyze mirrored traffic, resulting in latency, bandwidth overhead, and scalability bottlenecks. As high-throughput, low-latency infrastructures become the norm, there is a pressing need for intrusion detection that operates directly within the network fabric itself. In this work, we address the challenge of bringing intelligent anomaly detection into the data plane without sacrificing performance or adaptability. We present BINOCULAR, an in-network anomaly-detection framework that embeds a compact neural network within a programmable switch and augments it with a control-plane refinement loop to enable confidence-aware decisions and continuous model adaptation. Our results show that our solution sustains line-rate inference with microsecond-scale latency, achieves up to 95% F1 on standard intrusion-detection benchmarks, and leverages confidence-aware retraining to recover over 70% of lost F1 under traffic distribution shifts.**

I. INTRODUCTION

Mirroring traffic to external Intrusion Detection Systems (IDS) in high-speed networks, such as data centers, can introduce latency, consume bandwidth, and increase the risk of delayed reaction to attacks [1]–[4]. As link rates continue to grow and traffic mixes become more heterogeneous, the traditional *copy–export–analyze–react* loop struggles to keep pace with events happening directly on the data path. This raises a fundamental question: can anomaly detection be performed *in place*, directly within the network devices that forward the traffic, so that decisions are made as close as possible to the events of interest?

Recent advances in programmable switches offer a promising path [5]–[9]. By allowing developers to customize packet parsing, matching, and stateful actions at line rate, these devices enable offloading selected computations from servers into the data plane, an approach broadly referred to as in-network computing (INC) [7]. INC efforts fall into two broad categories: (i) accelerating machine learning (ML) training by exploiting the fast and distributed nature of network hardware, and (ii) performing inference within the network to reduce reaction time and avoid host overhead. For security applications such as anomaly detection, the latter direction is particularly attractive: an in-switch classifier can act on flows before they traverse the network, thereby reducing alert latency and preventing suspicious traffic from spreading.

Despite this potential, deploying neural models inside switches remains challenging. The protocol-independent switch architecture (PISA) exposes a constrained computational environment with no floating-point operations, limited ALUs per stage, bounded pipeline depth, and tight state [5], [10], [11]. As a result, conventional neural networks cannot be executed without significant restructuring. Prior efforts demonstrated that compression, quantization, and model partitioning can help adapt neural inference to PISA constraints; for example, recent work shows that compressed CNN architectures can run entirely inside a Tofino switch at microsecond-scale latency [10]. These works successfully establish *feasibility*. However, they do not fully address the *operational viability* of ML-based anomaly detectors, which must remain robust under distribution drift, emerging threats, and evolving traffic patterns.

In this work, we investigate whether an in-switch neural detector can provide accurate real-time anomaly detection while remaining adaptable over time. Our key premise is to jointly leverage the strengths of the data and control planes: the former offers deterministic, wire-speed execution, while the latter provides flexibility and computational capacity for learning and adaptation. To this end, we propose to couple *on-path inference* with *off-path adaptation*. The switch executes a compact neural network to classify flows in real-time, while the control plane selectively collects uncertain samples, refines the model, and updates the data plane as needed. The goal is to preserve line-rate guarantees without freezing the model at deployment time.

We design BINOCULAR, an in-network anomaly detection system that deploys a compact neural network (NN) inside a programmable switch and complements it with a lightweight, confidence-aware refinement loop in the control plane. [1] To enable deterministic, line-rate execution directly in the data plane, we convert a general NN into a binary NN (BNN) to tailor it to PISA constraints. The control plane program allows adjusting the BNN parameters on the switch to varying network traffic and conditions by means of a *confidence* metric

[1] BINOCULAR (**BI**narized **N**eural c**O**ntrol with adaptive **C**ontinual **U**pdating and **L**earning for **A**nomaly **R**ecognition) captures both the technical design and the operational intuition of our system. We choose the binocular metaphor as it reflects the system's dual perspective: fast, on-path inference inside the switch provides immediate visibility into traffic, while an off-path, control-plane refinement loop "reviews" low-confidence decisions to adapt the model over time. Together, these two "views" allow BINOCULAR to see network behavior more clearly, combining wire-speed responsiveness with continuous learning under traffic drift.

that determines the goodness of the installed BNN.

Our contributions can be summarized as follows:

- We introduce a split architecture that performs *on-path inference* in the data plane and *off-path adaptation* in the control plane, providing both real-time responsiveness and long-term adaptability.
- We design a neural network detector tailored to the constraints of commodity PISA switches, achieving line-rate execution and deterministic latency.
- We propose a confidence-driven mechanism that identifies low-certainty predictions and uses them to guide selective refinement, improving robustness without burdening the data plane.

We tested our approach in a simulated environment and over a software Tofino testbed. Results demonstrate that BINOCULAR sustains line-rate inference with microsecond-scale latency, improving inference time by 85% compared to state-of-the-art quantized models. Our system outperforms quantized baseline models up to 17.5% in terms of F1 score on traditional intrusion detection datasets in out-of-distribution scenarios.

The rest of the paper is organized as follows. Section II reviews prior work on executing machine learning inference in programmable data planes and highlights the gap between feasibility and long-term operational adaptability. Section III presents our system design, including BINOCULAR's dual-plane architecture, feature extraction pipeline (Section III-B), confidence-aware control-plane refinement (Section III-C), and data-plane deployment on programmable switches (Section III-E). In Section IV we present our evaluation results both in simulation and on a software Tofino testbed, quantifying inference latency, resource utilization, detection accuracy, and robustness under distribution shift. Finally, in Section V we conclude the paper.

II. RELATED WORK

The way machine learning is applied to networks is undergoing a major shift. Host-based approaches to networking tasks such as traffic classification, security monitoring, and congestion control [12]–[14] demonstrated reliable performance but have inherent delays. Decisions are made far from the data flow, relying on aggregate metrics that can lead to out-of-date information, unsuitable for real-time or microseconds-scale networking applications. To overcome this, researchers are now moving inference into the network fabric. By placing lightweight models directly into switches, frameworks can perform analysis and make decisions at line-rate, enabling real-time traffic classification [10], [15]–[20] and congestion control [6], [21]. However, deploying ML models directly to the network fabric is inherently limited by the programmable data plane architectures. Prior approaches can be broadly categorized by the learning model employed, neural networks or decision trees, and by where decision logic is placed across the data and control planes.

Neural Network Inference in the Data Plane. Because programmable switches lack floating-point support and offer limited arithmetic capabilities, early efforts relied on simplified neural models. Binarized neural networks (BNNs) [22], which restrict weights and activations to binary values, emerged as a natural fit. Systems such as N2Net [23], BoS [8], and N3IC [24] demonstrated that binarized layers can be executed using bitwise operations in programmable hardware. N3IC targets SmartNICs, while BoS deploys binary RNNs on Tofino switches. To improve accuracy, subsequent work explored quantized neural models. INQ-MLT [15] applied quantization-aware training to CNN and MLP models on software switches, while Quark [10] demonstrated quantized CNN inference on Tofino hardware through model compression and packet recirculation, enabling periodic retraining to handle domain shift. Other approaches increased model capacity through distribution or hardware offloading: NetNN [19] partitions DNNs across multiple switches, while Taurus [17] leverages external FPGA blocks to accelerate dot-product computations. These systems improve expressiveness but often incur higher latency, resource usage, or architectural complexity.

Decision Tree–Based Inference. A complementary line of work maps decision trees and ensemble models onto match–action pipelines. IIsy [18] encodes decision trees, SVMs, and Naive Bayes classifiers via table lookups, achieving line-rate inference by avoiding arithmetic operations. Planter [25] generalizes this mapping to a broader range of models, emphasizing flexibility and rapid prototyping. NetBeacon [20] introduces a multi-phase decision-tree architecture that compresses feature thresholds using ternary tables, while ART [6] distills deep reinforcement learning policies into decision trees for congestion-aware routing. While efficient and deployable, these approaches often trade accuracy and adaptability for simplicity.

Decision Placement Across Control and Data Planes. Beyond model choice, prior work also differs in where decisions are taken and how responsibility is split between the data and control planes. Some systems place all decision logic in the data plane, treating the control plane primarily as a configuration mechanism (e.g., NetBeacon [20], Planter [25]), achieving low latency but limited adaptability. Others retain inference and learning in the control plane, using the data plane mainly for measurement or feature extraction (e.g., Sonata [26], Quark [10], Mutant [13]), improving flexibility at the cost of latency and overhead. Hybrid designs such as BaNaNa-Split [16] partially bridge this gap by splitting execution across planes or devices, but still rely on static boundaries or heavyweight retraining cycles.

In contrast, BINOCULAR jointly addresses both dimensions by combining hardware-compatible neural inference in the data plane with a confidence-driven control-plane feedback loop. This design preserves deterministic, line-rate execution while enabling continuous adaptation under traffic drift, a combination not achieved by existing in-network IDS solutions.

III. System Overview and Design

At the core of the BINOCULAR, there is a dual-plane architecture that couples the deterministic, line-rate execution of programmable switches with a control-plane mechanism for adaptive learning-based intrusion detection. In particular, the data plane executes a binarized neural network (BNN) tailored to the PISA pipeline; the control plane runs training, calibration, model deployment, and applies refinements phases periodically.

Fig. 1 illustrates the end-to-end workflow of our adaptive intrusion detection framework. Network flow statistics are first extracted in the data plane and passed through the BNN executor, where simple operations (i.e., bitwise XNOR, popcount, and $SIGN$) are executed to perform real-time inference while meeting PISA constraints. The classification results and confidence scores are then analyzed in the control plane to determine whether samples are confidently classified or marked as critical for retraining. Critical samples are then labeled by a complete model (teacher MLP), which computes SHAP-based feature importance, and guides the student (in-switch BNN) during retraining. In the remainder of the section, we detail the diverse components.

A. BINOCULAR *Overview*

The co-design of in-switch processing with control-data plane communication enables fast in-network inference and continuous off-path learning, allowing the system to continuously adapt to evolving network behaviors. Packets belonging to a bidirectional flow are observed and summarized by the switch, which maintains a lightweight per-flow state until the flow reaches a "maturity" threshold, at which point it is classified. For classification, BINOCULAR aggregates a compact set of flow-level statistics and performs inference directly in the data plane using a BNN tailored to the PISA execution model. The BNN is trained offline through a combination of quantization-aware learning and teacher–student collaboration [27], where the BNN model (the student) learns from a set of MLP-labeled (the teacher) and original samples. This procedure allows the final student model to operate with binary weights and activations while preserving most of the accuracy of its full-precision teacher. This produces a compact neural representation that we then further modify to make it compliant with the PISA architecture through bitwise operations and lookup-based popcount approximation, while respecting the strict resource and timing constraints of programmable switches.

Once deployed, the data plane provides deterministic, line-rate classifications and attaches the predicted label to subsequent packets of the same flow, avoiding repeated inference. To remain effective in response to evolving traffic behavior, however, BINOCULAR does not freeze the model after deployment. The switch computes a lightweight confidence score for each decision and asynchronously reports the low-confidence cases to the control plane. This approach allows the switch to handle complex out-of-distribution cases.

We build a refinement pipeline that, through continuous teacher labeling of new inference samples, re-trains a new student model when necessary, and reinstalls refreshed binary weights into the switch. Because training and refinement occur off-path, BINOCULAR preserves predictable dataplane performance while still enabling the model to evolve over time.

This closed-loop process combines *(i)* a quantized BNN architecture that fits the data plane, *(ii)* a confidence-based training workflow with a quantization-aware strategy that produces lightweight deployable binary models, and *(iii)* an online feedback mechanism that maintains high accuracy under distributional drift.

B. Feature Extraction

BINOCULAR extracts flow-level features directly in the data plane by maintaining lightweight state keyed on the 5-tuple and flow direction. As packets belonging to the same bidirectional flow traverse the switch, we update a set of counters and statistical indicators as in Table I, where mean packet size features are derived in the control plane (by *spkts*, *sbytes*, and their destination counterparts), while all other features are updated directly in the data plane. Once the flow reaches a maturity threshold of k packets, these measurements are grouped into a fixed-length feature vector, ensuring that inference is triggered only when a sufficient number of packets have been observed. Since programmable switches offer limited memory and arithmetic capabilities, we further reduce the dimensionality of this feature vector offline via eXplainable AI (XAI).

Feature	**Description**	**Size (b)**
sbytes / dbytes	src / dst total bytes	14 / 14
spkts / dpkts	src / dst packet count	7 / 7
smeansz / dmeansz	src / dst mean packet size	14 / 14
smaxbytes / dmaxbytes	src / dst max packet size	14 / 14
sminbytes / dminbytes	src / dst min packet size	14 / 14
TCP flag count	Cumulative counts for TCP flags FIN, SYN, ACK, PSH, RST, and ECE	6×7

TABLE I: Flow-level features, along with the size, used by the inference pipeline.

Prior to training, all input features are first discretized and binarized to match the execution constraints of the target BNN. The resulting binary values are then concatenated into a unified bit-level feature vector, which is used to train a full-precision teacher model. After training, we analyze the teacher model using SHAP [28] to estimate the contribution of each bit-level feature to anomaly classification. Fig. 2 shows SHAP importance scores for the top 10 features. The visualization reveals that different binary values of the same feature bit contribute consistently to the model's output, as indicated by clusters of similarly colored points with comparable SHAP values. Based on this ranking, we retain only the most influential bit-features, producing a pruned binary input vector. This

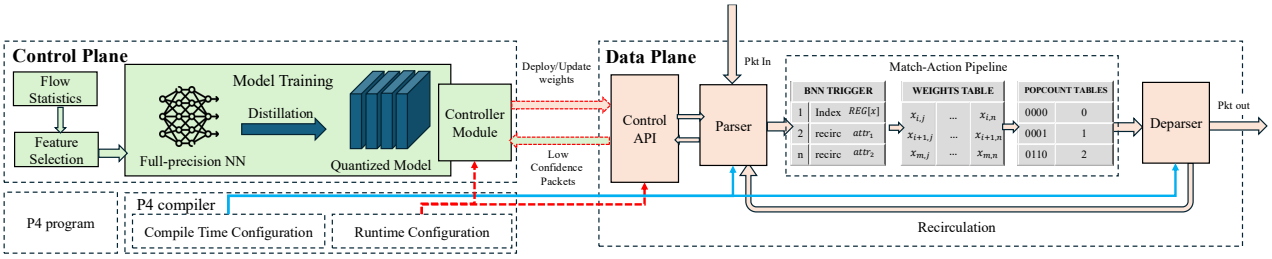


Fig. 1: BINOCULAR high-level architecture and workflow across the data plane and control plane, from feature extraction to BNN inference and confidence-driven refinement. Fast on-path decisions remain in-switch, while only low-confidence cases are escalated for retraining to sustain accuracy under drift without sacrificing line-rate performance.

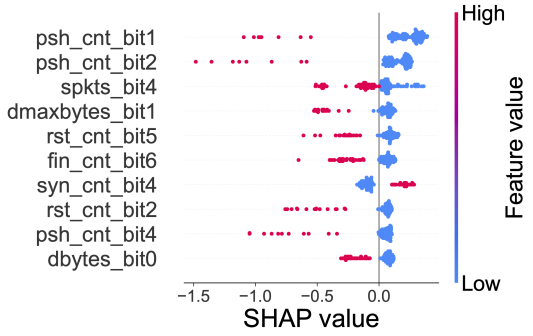


Fig. 2: SHAP beeswarm plot of the top feature-bits used by the teacher model; position indicates contribution to the prediction and color denotes the binary bit value (suffix `_bitN` is the N-th bit). The key takeaway is that a small subset of bits consistently dominates decisions, motivating SHAP-based pruning to reduce dataplane footprint with minimal accuracy loss.

pruning significantly reduces the dataplane footprint, attenuates feature noise, and enables deployment on programmable switches while remaining compatible with SRAM capacity and pipeline-stage constraints. This produces the binarized activation vector consumed by the BNN, utilizing only simple compare and bitset operations that naturally fit within the match–action pipeline.

C. Control Plane

While the data plane performs fast inference, the control plane supervises the learning process and maintains the long-term model accuracy. It collects *low-confidence* samples, leverages the teacher's stronger generalization to label them, re-trains the student, and deploys updated weights back to the switch. By running all adaptation off-path, the control plane enables BINOCULAR to evolve with traffic dynamics without interrupting line-rate forwarding.

Training workflow. Using the teacher–student approach, a full-precision MLP is first trained on labeled intrusion datasets using squared-hinge loss, as it directly optimizes a margin-based objective consistent with sign-based inference. This model serves as a full-precision oracle: it provides (i) feature attributions used for SHAP-based feature selection (Section III-B) and (ii) soft targets that guide the training of the binarized student. After feature selection, the student BNN, which matches the compact architecture deployable in the switch, is trained with quantization-aware learning so that it is exposed to binarization effects during training, rather than as a post-hoc conversion. We use a straight-through estimator (STE) [29] to backpropagate through the $\mathrm{sign}(\cdot)$ activation, enabling stable optimization despite the discrete nature of the representation.

Quantization-aware training. A simple post-training binarization would cause severe accuracy loss, as the student would be optimized in a continuous space and only later forced into discrete $\{-1, +1\}$ weights and activations. To avoid this mismatch, we adopt quantization-aware training (QAT) [10], which injects binarization directly into the forward pass. During training, each layer uses binary weights and the $\mathrm{sign}(\cdot)$ activation, while the backward pass employs the straight-through estimator (STE), which approximates the gradient of the sign function by an identity mapping within a clipped range. This enables the optimizer to account for quantization effects throughout the entire optimization process, resulting in smoother convergence and a student model whose behavior at training time closely matches its deployment behavior in the switch. In practice, QAT significantly reduces the accuracy gap typically observed in binarized networks, thereby maintaining acceptable detection performance under the tight arithmetic and memory constraints of the PISA pipeline.

Retraining Feedback Loop. One of the main challenges of deploying a quantized model in a programmable switch is to maintain an acceptable accuracy over time and deal with unseen traffic patterns. In BINOCULAR, we tackle this challenge by codesigning the switch control and data plane with a retraining feedback loop. When the control plane collects enough low-confidence samples, we leverage the teacher's robustness under evolving traffic distributions to perform a new student QAT round on the reduced feature set. To preserve previously learned behavior while adapting to traffic drift, the retraining set is built by augmenting the initial training dataset with a recent-flow dataset. The latter

consists of a fixed-size window of recent samples, composed of a combination of randomly selected flows and critical flows identified via low-confidence predictions, with a 40% random and 60% critical sample ratio. The total number of recent flows included in each refinement round is determined empirically (herein omitted for brevity). This approach avoids retraining from scratch while allowing BINOCULAR to track traffic drift over time. After each refinement, the binarized weights are exported, packed into bit arrays aligned with the switch register size, and transferred to the data plane (see Section III-E).

D. Online Confidence-Aware Feedback Loop

Even after careful offline training, traffic distributions evolve over time, and a static in-switch model eventually loses accuracy as workloads, applications, and attack patterns drift. However, continuously exporting flows to a remote classifier would defeat the purpose of in-network inference by adding latency and control-plane load. The challenge, therefore, is to preserve line-rate predictions for the vast majority of flows while selectively invoking an off-path model refinement only when needed. To meet this goal, BINOCULAR couples dataplane inference with a lightweight confidence scoring system that flags uncertain decisions for review and enables the control plane to correct and retrain the model without interrupting the *on-path* execution.

Confidence Score Mechanism. After each in-switch inference, BINOCULAR assigns a confidence score to decide whether the prediction should be trusted or re-examined by the control plane. Because binarized activations prevent the use of traditional probabilistic outputs (e.g., Softmax), BINOCULAR employs lightweight structural proxies to estimate prediction certainty directly in the data plane. We compute confidence either as the fraction of active neurons in the penultimate layer,

$$c = \frac{1}{n_{L-1}} \sum_{i=1}^{n_{L-1}} \mathbb{1}[h_{L-1,i} = +1],$$

or as a margin derived from the signed distance to the final decision threshold (obtained from the last accumulator s before comparison). Both metrics require only counters and comparisons, making them inexpensive to compute in the data plane.

If $c < \tau_c$ (a percentile-based threshold set during validation), the corresponding feature vector and minimal metadata are exported to the control plane. To avoid overload, BINOCULAR rate-limits exports to $\rho_{\max}$ samples/s and switches to randomized downsampling when the limit is exceeded. In all other cases (i.e., when confidence is high), the data plane performs the classification locally, and the control plane is not involved.

Refinement and Update Triggers. The control plane stores the exported samples in a sliding window and periodically checks whether enough uncertain cases have accumulated or a timeout $\Delta t_{\max}$ has elapsed. When either condition is met, a refinement round is launched: the student is re-trained, and the resulting model is validated against a hold-out set. If the new student improves accuracy or meets a fixed quality threshold, BINOCULAR prepares it for redeployment.

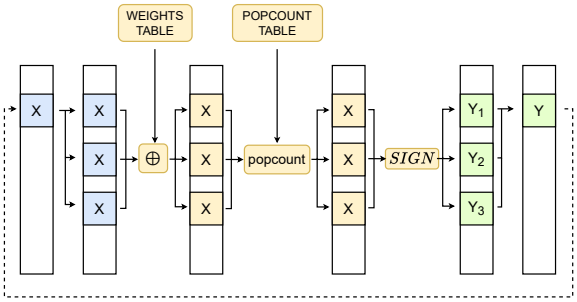


Fig. 3: In-switch BNN execution pipeline: features are duplicated for parallelism, processed via XNOR-based match-action lookups, and mapped through popcount tables with recirculation to complete deep inference.

E. Data Plane

The data plane implements real-time packet classification directly on the Tofino ASIC. It consists of *(i)* a feature extraction pipeline that aggregates flow statistics at line rate and triggers inference once a flow reaches maturity, and *(ii)* a BNN executor pipeline that performs a forward pass using bitwise operations and bounded recirculation. Through this design, we enable on-path inference under strict PISA constraints: no floating-point operations, no loops, a restricted number of ALU operations per pipeline stage, and a bounded Packet Header Vector (PHV) state, *i.e.,* a fixed-size container that carries packet headers and required metadata through the ASIC pipeline.

Feature Extraction Incoming packets are hashed on their 5-tuple and associated with lightweight per-flow state. As packets traverse the ingress pipeline, the switch updates counters, flags, and coarse timing signals (as per Table I) until the flow reaches a maturity threshold of k packets. At that point, the feature extractor emits a compact 168-bit vector encoding flow-level behavior. Simple boolean and counter-based features are computed directly in P4, while more complex statistics are offloaded to the control plane and reintegrated before binarization. The mature feature vector is then attached to a trigger packet via a custom BNN header, which initiates the inference pipeline.

BNN Inference. The executor processes one layer at a time. Given an input activation a_{l-1}, the data plane evaluates a batch of neurons in parallel using only bitwise operations. Binary multiplication is implemented with XNOR, and each partial dot product is accumulated through a popcount lookup table stored in SRAM (Fig. 3). After all chunks of a neuron's weight vector have been processed, the accumulated sum is compared against a threshold to produce a 1-bit output. The resulting activations form a_l, which is passed to the next layer without leaving the pipeline.

Algorithm 1 BNN Inference.

Input: B-bit activation vector a_0, binary weights W, parallelism factor P
Output: Classification result a_L.

```
for l = 1 to L do                                   ▷ Layer iteration
    for n = 1 to ⌈N_L/P⌉ do                         ▷ Batching neurons
        for k = ⌈dim(W_{l,n})//B⌉ to 1 do           ▷ Weight streaming
            a_l ← a_l ‖ Sign(a_{l-1,k} ⊕ W_{l,n,k,i})
                for i ∈ {1 ... P}                   ▷ Parallel XNOR-Popcount
        end for
    end for
end for
```

Recirculation for Multi-Layer Execution. To complete inference without violating timing constraints, BINOCULAR implements the hybrid recirculation strategy detailed in Algorithm 1. BINOCULAR interleaves neuron-wise and weight-wise recirculation to overcome hardware limits on per-stage ALUs and PHV capacity. The outer loops of Algorithm 1 group neurons into batches ($n \in \{1 \dots \lceil N_L/7 \rceil\}$) that fit in the PHV, enabling parallel evaluation. Simultaneously, the inner loop handles weight-wise recirculation by streaming weight vectors in fixed-width chunks (e.g., 14 bits). For each chunk, the data plane performs an XNOR $\rightarrow$ popcount $\rightarrow$ accumulate sequence, recirculating the packet to resume execution from the stored accumulator until the weight-batching index k is exhausted. Once all groups and chunks are processed, the assembled activation vector a_l serves as the input for the subsequent layer, repeating until the final classification.

BNN Executor Implementation Fig. 4 outlines the $P4_{16}$ logic for the BINOCULAR data plane. The inference process is divided into the following steps: *(i)* Using the `layer_no` field, the switch fetches initial input features via `get_bnn_input` into the `nr1` register. *(ii)* For subsequent layers, the switch performs on-the-fly reconstruction by selecting 7-bit activation pairs from the header (e.g., `l0_out_1`, `l0_out_2`) based on the `pop_recirc` index. *(iii)* These are concatenated into 14-bit inputs and processed by the corresponding weight tables (`l1_weights.apply()`). This iterative execution allows the switch to map multi-layer BNNs onto the hardware by reusing pipeline stages across successive passes.

```
    /* SET ACTIVATIONS AND WEIGHTS */
if(hdr.bnn_pkt.layer_no == 0) {
    nr1 = get_bnn_input.execute(input_offset);
    COPY()
    l0_weights.apply();
} else if(hdr.bnn_pkt.layer_no == 1){
    if(hdr.bnn_pkt.pop_recirc == 0) {
        nr1 = (hdr.bnn_pkt.l0_out_1 << 7) |
              hdr.bnn_pkt.l0_out_2;
    } else if (hdr.bnn_pkt.pop_recirc == 1) {
        nr1 = (hdr.bnn_pkt.l0_out_3 << 7) |
              hdr.bnn_pkt.l0_out_4;
    } else if (hdr.bnn_pkt.pop_recirc == 2) {
        nr1 = (hdr.bnn_pkt.l0_out_5 << 7) |
              hdr.bnn_pkt.l0_out_6;
    }
    nr2 = nr1;
    l1_weights.apply();
}
```

Fig. 4: $P4_{16}$ code of the BNN executor showing layer selection, activation reconstruction across recirculations, and weight-table application.

IV. EVALUATION

A. Evaluation Settings

We evaluate BINOCULAR on a simulation environment comprising an Intel Tofino (v1) programmable software switch. Experiments are conducted on a commodity server equipped with an NVIDIA Quadro RTX 6000 GPU and an Intel Xeon CPU. To generate the switch-deployable model, we train the student BNN using Knowledge Distillation and Quantization-Aware Training (QAT). We employ a Straight-Through Estimator (STE) to approximate gradients for the non-differentiable sign activation during backpropagation.

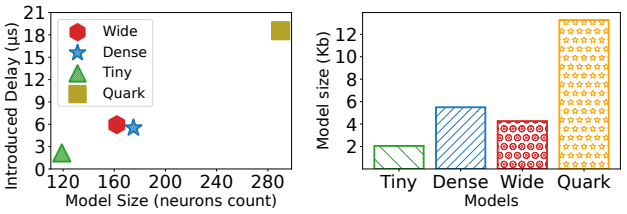
Teacher and student training in BINOCULAR is carried out with a batch size of 2048, using the squared hinge loss optimized with Adam and a fixed learning rate of 10^{-3}. The number of training epochs varies across models: the teacher is trained for 10 epochs, whereas student models are trained for 25 epochs. For SHAP-based feature selection on the student model, we use a background set of 32 samples and compute explanations over 128 samples, following standard practices to balance explanation fidelity and computational cost. We evaluated three BNN variants tailored to the switch pipeline constraints: *Tiny*: $[98, 21, 2]$ neurons (optimized for low latency). *Dense*: $[133, 42, 2]$ neurons (balanced). *Wide*: $[128, 32, 8, 2]$ neurons (maximum capacity). We compare our solution against two models for in-switch ML inference: Quark [10], representing quantized NN architectures, and NetBeacon [20], representing DT-based classification.

B. Experimental Datasets

We evaluate our system using two datasets: CICIDS-2017 [30] contains benign traffic mixed with the most up-to-date common attacks, including Brute Force (FTP/SSH), DoS, DDoS, Web Attacks, Infiltration, Botnet, and Heartbleed, captured over a 5-day period; UNSW-NB15 [31] encompasses nine distinct attack families: Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, and Worms. We used CICIDS2017 for initial training and validation. To assess robustness against distribution drift, we introduce samples from the UNSW-NB15 to simulate evolving traffic patterns.

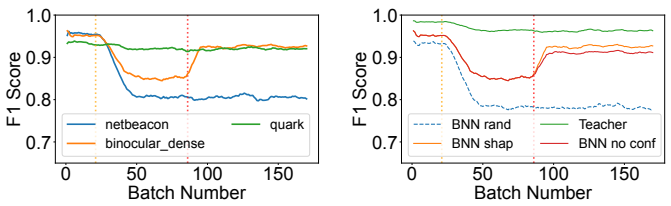
C. Performance Analysis of BNN Executors

Before assessing BINOCULAR as a complete intrusion detection system, it is important to isolate and quantify the cost of *data-plane neural inference alone*, independent of feature extraction and control-plane interaction. Programmable switches operate under strict timing constraints, and in-switch



(a) (b)

Fig. 5: (a) Introduced inference latency. Lower is better for on-path detection. (b) Model size. Smaller models reduce SRAM pressure and recirculation.



(a) Accuracy under shift. (b) Ablation of variants.

Fig. 6: (a) Accuracy under gradual distribution shift comparing BINOCULAR to Quark and NetBeacon. (b) Ablation comparing variants without confidence-based sampling (`BNN no conf`) and without SHAP feature selection (`BNN rand`). BINOCULAR limits degradation under drift and recovers performance through selective retraining.

inference must preserve deterministic forwarding behavior. This analysis, therefore, focuses on the execution overhead of the BNN executor pipeline, measuring the latency and resource cost introduced by neural inference relative to a baseline L3 forwarding program. We evaluate three representative architectures that explore different points in the design space in terms of depth, width, and neuron parallelism. By comparing their inference delay, recirculation requirements, and memory footprint, this analysis highlights the impact of neural architecture choices on the feasibility of the data plane.

Model overhead. We first analyze the additional latency of the in-switch inference pipeline for three different variants of our solution: `Tiny`, `Dense`, and `Wide` (see Section IV-A). The inference time is measured from the moment the *k-th* packet of a flow is received to the transmission of the resulting inference to the control plane. It is important to note that because the data and control planes operate asynchronously, this process does not degrade the overall switch throughput. Nevertheless, this analysis is critical as it quantifies the system's detection "agility", *i.e.,* how rapidly the switch can identify and report an anomalous flow. Fig. 5 shows the inference time as a function of model size, measured in microseconds assuming a standard Tofino data-plane clock of 1.22 GHz. We compare the three BINOCULAR architectures with the state-of-the-art quantize model Quark [10]. As expected, `Tiny` achieves sub-3 μs inference due to its small model size and limited recirculation requirements, corresponding to approximately $50\times$ the processing time of a baseline L3 switch. Although slightly more expensive, larger BINOCULAR variants remain within a few microseconds, without substantially degrading the performance compared to the tiniest model. In contrast, Quark [10] incurs significantly higher latency ($\sim$18 μs, $\sim$ $400\times$ L3). The observed overhead for the `Tiny` model ($\sim$ 3 μs) is operationally viable because: (1) it remains within the Tofino's microsecond-scale timing budget, and (2) the cost is amortized across the flow, as the result is cached to allow subsequent packets to bypass the BNN.

Our results show that the delay increases with both model size and recirculation depth, confirming that pipeline traversal frequency is the dominant latency factor.

Fig. 5b compares the model size of the three BNN alternatives and two state-of-the-art models: Quark [10], which uses a 1D-CNN architecture and fixed-point quantization. Quark presents a significant increase in weight size compared to our BNN variants, due to the deeper network and wider bit-width parameter quantization (7b).

TABLE II: Model-wise resource utilization for one packet traversal over the Tofino switch.

Model	SRAM [%]	MaxStageSRAM [%]	Recirculation
`Tiny`	5.3	35	25
`Dense`	5.3	35	64
`Wide`	7.6	65	69
`Quark`	24.3	100	102
`BoS`	26.3	63	-
`Netbeacon`	19.4	53.8	-

1) P4 Hardware Utilization Analysis: Since programmable switches must simultaneously support standard forwarding logic (e.g., L2/L3 routing, ACLs), minimizing the footprint of the inference engine is critical for real-world deployment feasibility. Table II summarizes the hardware resource consumption (*i.e.,* average consumption of SRAM, SRAM used from the most demanding stage of the pipeline, and number of recirculations) on the Tofino target for each BNN architecture, evaluated during a single packet traversal within the data plane.

SRAM usage is highest in the `Wide` variant due to its 16-bit popcount tables; conversely, `Tiny` and `Dense` employ 14-bit tables. Although they share a similar architectural design, `Dense` incurs a higher total inference cost than `Tiny` because it requires more recirculation cycles to process its wider layers. We note that total execution cost scales with these recirculations, unlike TCAM usage, which is constant at 0.3. This confirms that BINOCULAR is lightweight enough to be deployed alongside complex forwarding planes without causing resource contention.

To evaluate robustness against dynamic network conditions, we tested the BNN models on a sequence of batches exhibiting controlled distribution drift. The sequence starts with pure

TABLE III: Ablation Study across all phases. F1-score, Precision, and Recall reported before distribution shift (Pre-Shift), after shift before retraining (Post-Shift Pre-Retrain), and after confidence-aware retraining (Post-Shift Post-Retrain).

Model Size	Variant	Pre-Shift			Post-Shift (Pre-Retrain)			Post-Shift (Post-Retrain)		
		F1	Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.
Dense	Teacher (Oracle)	0.987	0.990	0.977	0.963	0.991	0.937	0.962	0.991	0.936
	BINOCULAR	**0.961**	**0.989**	**0.917**	**0.861**	**0.968**	**0.760**	**0.925**	**0.967**	**0.888**
	Only SHAP (No Conf)	0.961	0.989	0.917	0.861	0.968	0.760	0.912	0.978	0.853
	Random Features*	0.948	0.992	0.881	0.811	0.983	0.651	0.808	0.983	0.649
Tiny	Teacher (Oracle)	0.987	0.992	0.977	0.962	0.992	0.935	0.961	0.992	0.933
	BINOCULAR	**0.948**	**0.994**	**0.879**	**0.853**	**0.946**	**0.764**	**0.907**	**0.974**	**0.846**
	Only SHAP (No Conf)	0.948	0.994	0.879	0.853	0.946	0.764	0.899	0.965	0.839
	Random Features*	0.942	0.957	0.901	0.773	0.903	0.635	0.770	0.909	0.629
Wide	Teacher (Oracle)	0.987	0.992	0.977	0.962	0.993	0.935	0.962	0.992	0.936
	BINOCULAR	**0.951**	**0.962**	**0.918**	**0.825**	**0.940**	**0.714**	**0.919**	**0.986**	**0.859**
	Only SHAP (No Conf)	0.951	0.962	0.918	0.825	0.940	0.714	0.891	0.942	0.846
	Random Features*	0.950	0.992	0.888	0.787	0.980	0.608	0.787	0.982	0.610

*Static baseline: no retraining performed.

CICIDS2017 traffic, followed by a gradual injection of CIC-UNSW-NB15 samples starting at the *distribution shift* point and reaching 40% of the total volume. This mixture emulates natural traffic evolution, testing the system's ability to handle unseen data distributions

To evaluate the system's resilience against evolving network threats, we analyze the impact of gradual distribution drift on the BINOCULAR dense model. In Fig. 6a, we report the accuracy performance of BINOCULAR compared to state-of-the-art baselines Quark [10] and NetBeacon [20]. Initially, the injection of new traffic causes a performance dip, with accuracy declining from a baseline of 0.952 to 0.852 at peak shift (40% mix). However, the control plane intervention effectively mitigates this degradation. The teacher MLP—which remains robust against the shift—labels low-confidence samples to trigger targeted retraining. This adaptation restores the student BNN's accuracy to an average of 0.926 (peaking at 0.94). Our confidence-based loop limits total degradation to 10%, significantly outperforming the 15% drop observed in the randomized feature baseline. The observed 3.9% post-retraining gain confirms that BINOCULAR can autonomously recover from distribution drift.

D. Ablation Study

To isolate the contributions of our core design choices, we conduct an ablation study to study the impact of SHAP-based feature selection and the confidence-aware refinement loop. We tested the BNN models on a sequence of batches, starting with pure CICIDS2017 traffic and gradually adding CIC-UNSW-NB15 samples. We compare the variants of our BINOCULAR architecture—which comprises a binarized student model and a confidence-aware full-precision Teacher—against the following baselines: (i) `Rand` serves as a static baseline utilizing randomly selected features without a retraining process; (ii) `BNN shap` represents our full BINOCULAR architecture where features are selected via SHAP analysis; and (iii) `BNN no conf` performs the retraining process using the Teacher's labels but selects samples uniformly at random, omitting the confidence score logic. We report results for the `Tiny`, `Dense`, and `Wide` variants of our model in Table III.

Effectiveness of SHAP-based Feature Selection. We first evaluate the quality of the features selected via SHAP analysis. In Fig. 6b, we compare the pre-shift performance; the SHAP-based models achieve a higher F1-score (0.961) compared to the random feature baseline (0.945), confirming that the selected subset captures more relevant traffic characteristics. Furthermore, the SHAP features demonstrate better robustness to distribution shift. When the foreign traffic is injected, the *Rand* baseline suffers a performance drop to 0.811. In contrast, the SHAP-based model maintains a higher F1 of 0.842 even before any retraining takes place.

Efficacy of Confidence-Aware Retraining Loop. Fig. 6b shows that while the `Rand` baseline remains static, both BINOCULAR and the `BNN no-conf` variant undergo adaptation starting from the same post-shift F1-score (0.861). The results highlight the importance of the confidence-aware mechanism in the retraining loop. In the `BNN no-conf` setting, the model is retrained on samples selected uniformly at random, without leveraging the Teacher's confidence metrics. Under this configuration, the `Dense` model achieves a mean F1-score recovery of only 0.912. In contrast, the full BINOCULAR architecture leverages the full-precision Teacher to selectively identify and label the low-confidence samples for the retraining set. This refinement allows BINOCULAR to achieve accuracy recovery, reaching a mean F1-score of 0.925. This relative gain (+6.4% *vs.* +5.1%) proves that confidence-aware filtering is essential.

V. CONCLUSION

In this work, we introduced BINOCULAR, an in-network anomaly-detection framework that reconciles the deterministic

performance of programmable data planes with the need for runtime adaptability. By co-designing a compact Binarized Neural Network (BNN) specifically for the PISA architecture and pairing it with a confidence-aware control-plane refinement loop, we addressed the fundamental limitations of static in-switch classifiers. Our methodology, which integrates SHAP-based feature selection and teacher-student distillation, allows the system to operate within strict hardware memory and timing constraints without sacrificing detection accuracy.

Experimental evaluation on a hardware testbed confirms that BINOCULAR sustains line-rate inference and achieves an F1-score of up to 95% on standard intrusion detection benchmarks. We show that our solution quickly adapts to distribution drift using a retraining feedback loop.

ACKNOWLEDGMENT

This work has been supported by NSF awards #2201536 and #2133407.

REFERENCES

[1] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better netflow," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 245–256, 2004.

[2] L. Castanheira, R. Parizotto, and A. E. Schaeffer-Filho, "FlowStalker: Comprehensive Traffic Flow Monitoring on the Data Plane using P4," in *ICC 2019 - IEEE International Conference on Communications*. IEEE, 2019, pp. 1–6.

[3] H. Mostafaei and S. Afridi, "P4Flow: Monitoring Traffic Flows With Programmable Networks," *IEEE Communications Letters*, vol. 25, no. 11, pp. 3546–3550, 2021.

[4] Y. Chen, S. Layeghy, L. D. Manocchio, and M. Portmann, "P4-nids: High-performance network monitoring and intrusion detection in p4," in *Intelligent Computing-Proceedings of the Computing Conference*. Springer, 2025, pp. 355–373.

[5] A. Sacco, F. Esposito, and G. Marchetto, "On control and data plane programmability for data-driven networking," in *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2021, pp. 1–6.

[6] A. Angi, A. Sacco, F. Esposito, and G. Marchetto, "Routing with art: Adaptive routing for p4 switches with in-network decision trees," in *GLOBECOM 2024-2024 IEEE Global Communications Conference*. IEEE, 2024, pp. 3291–3296.

[7] R. Parizotto, B. L. Coelho, D. C. Nunes, I. Haque, and A. Schaeffer-Filho, "Offloading machine learning to programmable data planes: A systematic survey," *ACM Computing Surveys*, vol. 56, no. 1, pp. 18:1–18:34, Aug. 2023.

[8] J. Yan, H. Xu, Z. Liu, Q. Li, K. Xu, M. Xu, and J. Wu, "Brain-on-Switch: Towards Advanced Intelligent Network Data Plane via NN-Driven Traffic Analysis at Line-Speed," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, 2024, pp. 419–440.

[9] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," in *Proceedings of the 18th ACM workshop on hot topics in networks (HotNets '19)*, 2019, pp. 25–33.

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

[11] A. Angi, A. Sacco, F. Esposito, G. Marchetto, and A. Clemm, "Load profiling via in-band flow classification and p4 with howdah," *IEEE Transactions on Network and Service Management*, vol. 21, no. 1, pp. 295–309, 2023.

[12] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *ACM Queue*, vol. 14, no. 5, pp. 20–53, 2017.

[13] L. Pappone, A. Sacco, and F. Esposito, "Mutant: Learning congestion control from existing protocols via online reinforcement learning," in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, 2025, pp. 1507–1522.

[14] A. Angi, A. Sacco, F. Esposito, G. Marchetto, and A. Clemm, "NAIL: A Network Management Architecture for Deploying Intent into Programmable Switches," *IEEE Communications Magazine*, vol. 62, no. 6, pp. 28–34, 2023.

[15] K. Zhang, N. Samaan, and A. Karmouch, "A machine learning-based toolbox for p4 programmable data-planes," *IEEE Transactions on Network and Service Management*, vol. 21, no. 4, pp. 4450–4465, 2024.

[16] D. Sanvito, G. Siracusano, and R. Bifulco, "Can the network be the ai accelerator?" in *Proceedings of the 2018 Morning Workshop on In-Network Computing*, ser. NetCompute '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 20–25.

[17] T. Swamy, A. Rucker, M. Shahbaz, I. Gaur, and K. Olukotun, "Taurus: a data plane architecture for per-packet ml," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22)*, ser. ASPLOS '22. Association for Computing Machinery, 2022, p. 1099–1114.

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

[19] K. Razavi, S. D. Fard, G. Karlos, V. Nigade, M. Mühlhüser, and L. Wang, "Netnn: Neural intrusion detection system in programmable networks," in *2024 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2024, pp. 1–8.

[20] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 6203–6220.

[21] Z. Dong *et al.*, "A reinforcement learning approach to internet congestion control," in *International Conference on Machine Learning (ICML)*, 2018, pp. 1285–1294.

[22] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognition*, vol. 105, p. 107281, 2020.

[23] G. Siracusano and R. Bifulco, "In-network neural networks," *arXiv preprint arXiv:1801.05731*, 2018.

[24] 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 22)*. Renton, WA: USENIX Association, Apr. 2022, pp. 513–533.

[25] 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*, 2024.

[26] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven streaming network telemetry," in *Proceedings of the 2018 conference of the ACM special interest group on data communication (SIGCOMM '18)*. Association for Computing Machinery, 2018, pp. 357–371.

[27] Z. Meng, M. Wang, J. Bai, M. Xu, H. Mao, and H. Hu, "Interpreting Deep Learning-Based Networking Systems," in *Proc. of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20)*, 2020, pp. 154–171.

[28] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in neural information processing systems (NeurIPS '17)*, vol. 30. Curran Associates, Inc., 2017.

[29] P. Yin, J. Lyu, S. Zhang, S. Osher, Y. Qi, and J. Xin, "Understanding straight-through estimator in training activation quantized neural nets," *arXiv preprint arXiv:1903.05662*, 2019.

[30] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *International Conference on Information Systems Security and Privacy*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:4707749

[31] H. Mohammadian, A. Habibi Lashkari, and A. A. Ghorbani, "Poisoning and evasion: Deep learning-based nids under adversarial attacks," in *2024 21st Annual International Conference on Privacy, Security and Trust (PST)*, 2024, pp. 1–9.