HyNIC: Hybrid In-Network Inference for Line-Rate Anomaly Detection on SmartNICs

Aristide Tanyi-Jong Akem[†§] and Noa Zilberman[§]
[†]University of Southampton, United Kingdom, [§]University of Oxford, United Kingdom
a.t-j.akem@soton.ac.uk, noa.zilberman@eng.ox.ac.uk

***Abstract*—SmartNICs have emerged as a promising platform for in-network machine learning inference, yet existing approaches largely rely on stateless packet-level inference, off-path stateful inference or offloading flow-level analysis to the host, limiting performance. This creates a performance gap between line-rate inference capabilities in the data plane and the need for flow-aware context in security and monitoring applications. In this paper, we bridge this gap by exploiting the coexistence of programmable data planes and on-NIC processing cores on SmartNICs. We propose `HyNIC`, a hybrid in-network inference system that performs line-rate packet classification for intrusion and anomaly detection in the data plane while subsequently enriching inference with stateful flow-level context computed on SmartNIC cores and integrated back at runtime. `HyNIC` enables a seamless transition from stateless to flow-aware inference without diverting packets from the fast path. We implement `HyNIC` in P4 on an industry-grade SmartNIC and evaluate it on realistic IoT intrusion detection datasets, demonstrating significant accuracy improvements of up to 22% over a stateless-only baseline while preserving line-rate performance.**

***Index Terms*—In-network inference, SmartNIC, anomaly detection, machine learning, P4**

I. Introduction

Modern network systems are facing an unprecedented scale of security challenges due to increasingly complex architectures and the rapid growth of network traffic, particularly with the rise of the Internet of Things (IoT) and its heterogeneous devices. Traditional security frameworks, such as Intrusion Detection and Prevention Systems (IDS/IPS) running on general-purpose servers, struggle to keep pace and often become performance bottlenecks [1]. While machine learning (ML) has emerged as a powerful tool for detecting sophisticated attacks [2], ML-based solutions are typically deployed on dedicated servers or in the control plane, introducing latency and overheads that limit their efficacy in high-speed networks [3].

Programmable network devices such as SmartNICs enable ML inference directly in the data plane at line rate, supporting the emerging paradigm of *in-network ML* [4]. Existing approaches fall into two categories. *On-path* methods embed models in the data plane to achieve high throughput but are often limited to stateless features or require an initial observation window before flow-level features become available [5]–[9]. *Off-path* approaches leverage on-NIC cores to perform flow-aware inference but divert packets from the fast path, increasing latency and reducing throughput [10]–[13]. As a result, current systems must choose between classifying all packets at line rate without flow context, or running flow-aware inference (on/off-path) that excludes early packets, which can constitute a significant fraction of traffic in workloads dominated by short flows [14]. To the best of our knowledge, no existing work enables line-rate classification of all packets while dynamically incorporating flow-level features at runtime.

In this paper, we introduce `HyNIC`, a hybrid in-network ML inference framework for SmartNICs that combines line-rate data-plane inference with on-NIC stateful feature computation. `HyNIC` deploys a trained tree-based model as match-action table entries to perform per-packet inference on the fast path. During the early stage of each flow, packets are classified using only stateless features, while mirrored copies are processed on the on-NIC cores to incrementally compute flow statistics such as packet size variability, inter-arrival time summaries, and counters. Once sufficient flow context is available, the derived stateful features are installed into the data plane as flow-specific entries. Subsequent packets are then classified using both stateless and stateful features, enabling richer inference while keeping packets on the fast path. This progressive refinement avoids early commitment, enabling early detection and mitigation for short flows while localizing the impact of misclassifications and preserving line-rate performance.

Unlike prior approaches that operate purely on-path without flow context or off-path with higher latency, `HyNIC` bridges this gap by coordinating data-plane inference with asynchronous state construction on the NIC cores. This design preserves the determinism and throughput of on-path processing while overcoming the feature limitations of stateless methods and the latency overheads of off-path solutions. As such, `HyNIC` provides a practical and efficient foundation for flow-aware, in-network ML inference on modern SmartNIC architectures.

By proposing `HyNIC`, we make the following contributions:

- We identify a gap in existing SmartNIC-based in-network IDS approaches, namely the inability to combine line-rate inference with both stateless features and stateful features introduced at runtime. We address this gap by proposing `HyNIC`, a hybrid in-network ML framework that leverages coordination between data-plane and on-NIC processing.
- We design a collaborative inference architecture in which the data plane performs line-rate classification while on-NIC cores derive flow-level features during early-flow operation and inject them back into the data plane, enabling adaptive stateful inference fully in the fast path.
- We implement `HyNIC` in P4 on an Intel IPU and evaluate it on publicly available IDS datasets. Results show that hybrid flow-aware inference improves detection performance over a stateless baseline by up to 22% in F1 score.

Work	Platform	On-Path Inference	Stateless Inference	Stateful Inference	Line Rate	CP-DP collaboration
[10]	BlueField-2	✗	✗	✓	✗	✗
[11]	BlueField-2/3	✗	✗	✓	✗	✗
[12]	BlueField-3 + GPU	✗	✗	✓	✗	✗
[7]	Netronome Agilio CX	✓	✗	✓	✓	✗
[6]	Netronome NFP / FPGA	✓	✗	✓	✓	✗
[5]	Netronome AgilioCX	✓	✓	✗	✓	✗
[16]	NetFPGA-Plus	✓	✓	✗	✓	✗
HyNIC	**Intel IPU**	✓	✓	✓	✓	✓

Table I: Comparison to related work. Columns depict: hardware platform on which each solution is deployed; if inference runs on-path; if inference relies on stateless or stateful features; if ML inference is at line rate; and if tight control-data plane (CP-DP) collaboration is employed.

II. Background and Motivation

A. In-network machine learning for network security

Recent work has shown that programmable packet processing pipelines, using domain-specific languages such as P4 [15], can host trained lightweight ML models for traffic classification and anomaly detection [5], [6], [8], [16]. Existing approaches can be broadly categorized based on the features they employ. Stateless inference relies on per-packet features extracted independently from each packet [5], [8], while stateful inference incorporates rich flow-level context computed across multiple packets of the flow [6], [11].

Although stateful features have been shown to improve classification accuracy for security workloads [14], [17], they require maintaining per-flow state and computing aggregate statistics. Most prior in-network ML systems target programmable switches, whose pipelines are optimized for simple, deterministic operations [18]. Limited memory and restricted support for arithmetic and temporal aggregation make it challenging to compute and maintain rich flow-level state. These constraints motivate more flexible state management, which SmartNICs enable by leveraging on-NIC cores instead of relying solely on pipeline-based processing.

B. SmartNIC architectures

SmartNICs integrate high-throughput packet processing engines with general-purpose compute resources, enabling network functions to be offloaded from the host, freeing compute resources, and improving packet-processing performance. As surveyed in [19], SmartNICs can separate fast, deterministic packet processing performed directly in the forwarding pipeline (on-path) from more expressive computation on NIC cores outside the forwarding path (off-path), providing greater flexibility than programmable switches, and enabling more efficient handling of stateful processing. Several commonly used SmartNIC platforms in the literature exemplify this design, including NVIDIA BlueField DPUs [20], Netronome Agilio SmartNICs [21], and, more recently, Intel Infrastructure Processing Units (IPU) [22], which expose a P4-programmable packet processing pipeline tightly coupled with on-NIC cores.

C. Related work

Prior work on in-network inference on SmartNICs for security applications explores a wide range of design trade-offs, particularly where inference takes place, supported features, and whether line-rate operation can be sustained [23]. These design choices directly affect both the complexity of the deployed classification model and inference latency.

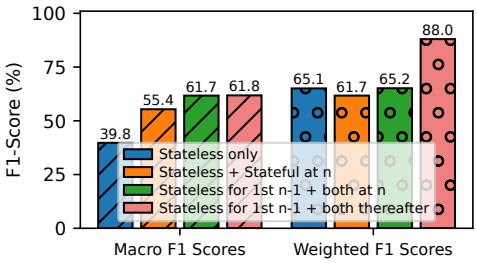


Figure 1: Effect of *when* and *how* stateless and stateful features are used on attack classification performance. Comparing four strategies: (i) stateless features for all packets; (ii) stateful features computed over the first n packets and combined with stateless features at packet n to tag the flow, with the result applied to subsequent packets; (iii) stateless features for the first $n-1$ packets and combined stateless-stateful inference at packet n, with the result applied to subsequent packets; and (iv) stateless features for the first n packets, followed by combined stateless-stateful inference for all subsequent packets using the state computed over the first n packets.

Table I highlights these trade-offs across representative systems. Several approaches perform inference off the data path using NIC-attached CPUs or accelerators, enabling richer models but sacrificing on-path, line-rate processing [10]–[13]. In contrast, systems that embed inference directly in the data plane achieve line-rate performance [5]–[9], but typically restrict inference to either stateless or stateful features, while also offering limited or no coordination between the data plane and the control plane for better performance.

As a result, existing designs invariably forgo at least one key capability: on-path inference, combined use of stateless and stateful features, sustained line-rate operation, or tight control-data plane collaboration. As shown in Table I, no prior system simultaneously satisfies all these requirements, motivating a closer examination of how stateful feature computation and inference should be structured on SmartNICs.

D. Motivating example

Consider an attack classification task based on the ToN-IoT dataset [24] (details in §IV-B). Stateless features enable per-packet inference but do not provide flow-level context, while stateful features computed over the first n flow packets capture flow dynamics but delay classification and often ignore early packets during feature computation phases. Figure 1 shows that combining both stateful and stateless features improves performance. However, existing systems typically infer only until a reliable flow-level result is reached (at packet n), then apply this prediction to all subsequent packets, ignoring their individual stateless features which could be informative.

We observe that continuing inference on every packet (beyond n), using their individual stateless features alongside the computed stateful features, yields significantly better overall

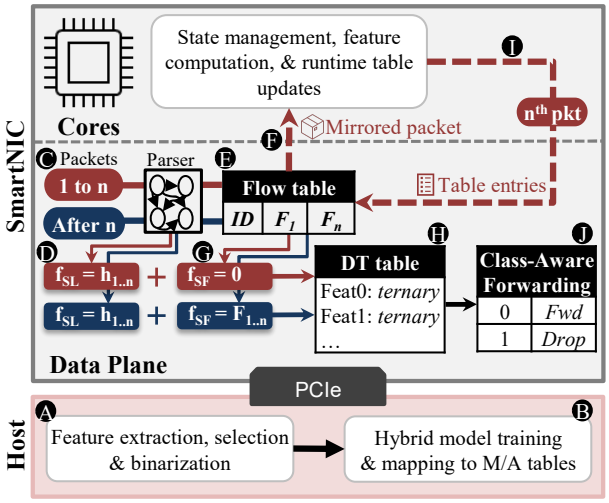


Figure 2: Overview of the HyNIC system. Steps A to J depict the sequence of events. f_{SL} and f_{SF} respectively denote stateless and stateful feature sets, while $h_{1..n}$ and $F_{1..n}$ are respectively the stateless packet header fields and the stateful features obtained from the flow table.

performance, as shown in Figure 1. Since state is difficult to compute on-path due to the inherent constraints of programmable data planes, on-NIC processing cores can be used to compute and inject the stateful features at runtime, while keeping inference in the data plane.

III. Hybrid In-Network Inference on SmartNICs

We design HyNIC as a framework for hybrid in-network inference on SmartNICs, combining stateless and stateful features. We assume a SmartNIC architecture with a P4-programmable data plane and on-NIC cores executing the control plane. SmartNIC data planes impose key constraints, including limited pipeline resources, restricted table key widths, and limited support for stateful processing, which hinder maintaining rich per-flow state entirely in the data plane. To address these limitations, HyNIC performs line-rate stateful inference in the P4 pipeline for established flows, while classifying the first n packets using only stateless features.

A. System overview

Figure 2 summarizes the HyNIC workflow, spanning offline model preparation and runtime inference in the data plane. In the offline phase (Ⓐ–Ⓑ), stateless features are extracted from packet traces and used to derive stateful features over the first n packets of each flow. A feature selection pipeline identifies relevant features and model parameters, after which features are binarized to enable efficient deployment. The trained decision tree is encoded as match-action (M/A) table entries, where each root-to-leaf path is represented as a ternary (value, mask) match with an associated class label.

At runtime, packets (Ⓒ) are parsed to extract stateless features (Ⓓ) and matched against the flow table using their five-tuple (Ⓔ). For flows without installed state, packets follow the stateless path: they are classified using stateless features while being mirrored to the on-NIC cores for stateful feature computation. The cores incrementally build per-flow state and, after observing n packets, install the corresponding stateful features into the flow table in the data plane (Ⓘ).

Once state is available, subsequent packets follow the stateful path, combining stateless and stateful features for inference. Classification outcomes are finally applied via a class-aware forwarding table to forward or drop packets (Ⓙ).

B. Hybrid model preparation

As in prior work on offloading ML inference to the data plane [4], [18], model training in HyNIC is performed offline (*e.g.*, on the host or an ML server). IP and transport-layer headers are extracted from PCAP files using Tshark [25]. We use 16 stateless features, including IP length, protocol, and TTL; TCP ports, window size, data offset, and flags; UDP ports and length. Flows are identified via the standard 5-tuple.

Following prior work on stateful inference [14], [26], we compute stateful features over the first n packets of each flow. We evaluate $n \in [2, 20]$ and select the smallest value preserving accuracy. The 18 stateful features include statistics (max, min, mean, sum, std) of IP length and inter-arrival time, TCP flag counts, and UDP length extrema. These are assigned to subsequent packets, depicting the flow state once populated in the data plane. Thus, the first n packets use stateless features only, while subsequent packets use stateless + stateful features.

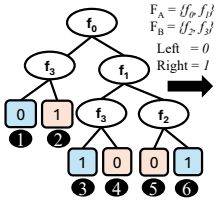
Using this extracted data, we perform feature and model selection (step Ⓐ in Figure 2). We adopt decision tree (DT) models as in previous work [8], [14], [17] due to their efficient mapping to M/A tables and strong performance on tabular data [27]. Using Scikit-learn [28], we train an initial DT and rank features by mean decrease in impurity (MDI). Features are then incrementally added in descending order of importance, retraining until the smallest subset preserving maximum accuracy is found, yielding a compact model.

We then binarize features before final training: each feature requiring a maximum of N bits is converted into N binary features. This facilitates mapping to constrained P4 targets by ensuring binary decision thresholds. Using the transformed features, we train the final DT (Ⓑ), allowing full growth while limiting the number of leaf nodes to 1000 to bound model size.

C. Model mapping to match-action table entries

We illustrate the DT-to-M/A mapping using the example in Figure 3. The DT operates on two features, F_A and F_B, binarized into $\{f_0, f_1\}$ and $\{f_2, f_3\}$, respectively. Each root-to-leaf path maps to a conjunction of constraints over these features and is encoded as a bit-level value-mask pair, where mask bits of 1 denote constrained bits and 0 denote wildcards.

The table in Figure 3 lists encodings for all leaf nodes. For example, Leaf 3 constrains $f_0 = 1$, $f_1 = 0$, and $f_3 = 0$, leaving f_2 unconstrained. Using the bit order (f_0, f_1, f_2, f_3), this path is encoded as value 1000 with mask 1101. While these are used internally, data-plane matching operates on the original features F_A and F_B. Thus, encodings are mapped back to per-feature ternary matches by splitting value-mask pairs along



Leaf	Value	Mask	(F_A, Mask)	(F_B, Mask)	Class
1	0000	1001	(00, 10)	(00, 01)	0
2	0001	1001	(00, 10)	(01, 01)	1
3	1000	1101	(10, 11)	(00, 01)	1
4	1001	1101	(10, 11)	(01, 01)	0
5	1100	1110	(11, 11)	(00, 10)	0
6	1110	1110	(11, 11)	(10, 10)	1

Figure 3: Tree mapping to ternary match-action table entries

feature boundaries. For Leaf 3, this yields $(F_A, \text{mask}) = (10, 11)$ and $(F_B, \text{mask}) = (00, 01)$, which match directly against extracted feature values. This enables representing the DT within a single M/A table executed in one stage, improving over prior techniques requiring at least two stages [16], [29].

Each row corresponds to an M/A entry with per-feature ternary matches and the leaf class. To ensure correct matching with overlapping rules, entries are prioritized by the number of constrained bits, so more specific paths match first.

D. HyNIC data plane operation

The core of HyNIC lies in its data plane operation, which enables low-latency inference through a hybrid stateless/stateful design. As summarized in Algorithm 1 and illustrated in Figure 2 (Ⓒ–Ⓘ), each arriving packet p is first parsed to extract stateless features $F_{\text{stateless}}(p)$. The packet 5-tuple is then used to query the flow state table to determine whether stateful features are available. If a matching entry exists, the retrieved F_{stateful} are combined with $F_{\text{stateless}}(p)$ and passed to the model for inference. The resulting prediction y is then matched on the class-aware forwarding table to either forward or drop the packet. For packets belonging to new or short flows, no stateful features are available. In this case, inference is performed using only stateless features, while packets are mirrored to on-NIC cores for stateful processing. Once sufficient packets have been observed, the computed stateful features are installed into the flow state table, enabling subsequent packets to follow the full inference path with both sets of features.

E. State management in the on-NIC cores

Due to the limited support for complex stateful operations in the data plane, HyNIC offloads flow state management to the on-NIC cores. Packets belonging to flows without existing state are processed using stateless inference in the data plane while being mirrored to the cores, limiting core involvement to early-stage flow packets. The cores receive mirrored packets via a NIC-local interface and maintain per-flow state indexed by the 5-tuple. This state is incrementally updated with each packet, tracking counters and summary statistics required for stateful feature construction (*e.g.*, packet counts, length aggregates, and protocol-specific indicators).

Once $N_{\text{threshold}}$ packets of a flow have been observed, the cores derive the stateful feature vector and install it into the data plane flow table via the runtime API. Subsequent packets of that flow can then directly leverage stateful features without further mirroring. Because updates are asynchronous, packets arriving during the transition may still follow the stateless path. Flow entries are installed atomically and associated with inactivity timeouts to prevent stale state, enabling dynamic state population while keeping inference in the data plane.

Algorithm 1 Hybrid inference workflow

Input: Packet p with 5-tuple $(src_{ip}, dst_{ip}, src_{port}, dst_{port}, proto)$
Data Plane Processing:
if packet p matches existing flow in the state tables **then**
  Extract stateless features $F_{\text{stateless}}(p)$
  Retrieve stateful features F_{stateful} from P4 state table
  Inference: $y = Model(F_{\text{stateless}}, F_{\text{stateful}})$
  Forward or drop based on y
else
  Extract stateless features $F_{\text{stateless}}(p)$ only
  Stateless inference: $y = Model(F_{\text{stateless}}, \emptyset)$
  Forward or drop based on y
  Mirror packet to on-NIC cores via data plane mirroring session
end if
On-NIC Core Processing (in parallel to data plane):
if mirrored packet received **then**
  Parse headers and extract flow key
  Update flow state: $S_{\text{flow}} = \{pkt\_count, max\_len, min\_len, ...,\}$
  if $S_{\text{flow}}.pkt\_count = N_{\text{threshold}}$ **then**
    Compute flow features: $F_{\text{stateful}} = f(S_{\text{flow}})$
    Install via runtime API: $state\_table.add\_with\_assign\_sf(F_{\text{stateful}})$
  end if
end if

Unlike prior approaches that fix a flow's classification result after the first n packets, HyNIC classifies every packet, avoiding early commitment. By combining stateless features with progressively available stateful context, it localizes misclassifications: false positives affect only individual packets, while false negatives can be corrected as evidence accumulates, improving robustness without sacrificing accuracy.

F. Hardware implementation

HyNIC is implemented on an Intel IPU E2100, with a P4-programmable data plane and on-NIC Arm cores with runtime control over M/A tables. The inference workflow is mapped to three main M/A tables: an exact-match flow table storing stateful features, a ternary M/A table encoding the DT, and a class-aware forwarding table enforcing classification outcomes. Flow table misses trigger packet mirroring to the on-NIC cores, which maintain per-flow state and compute stateful features. Once sufficient packets are observed, features are installed into the data plane via atomic runtime updates. This ensures that packets observe either no state or fully installed entries, avoiding partial updates. All flow state is maintained in tables rather than mutable pipeline registers, enabling safe concurrent processing without data-plane synchronization.

IV. Experimental Evaluation

A. Testbed setup

All experiments are conducted on an Intel IPU E2100-CCQDA2 [22], a PCIe 4.0 ×16 SmartNIC with two 100 GbE ports and a P4-programmable pipeline capable of line-rate processing. The IPU integrates on-NIC compute (16 Arm cores) and high-bandwidth memory, enabling concurrent data-plane and control-plane execution. The IPU is hosted on a multi-core Dell R760 server with 375 GB of DDR4 memory and 100 GbE connectivity. A second server with similar capabilities is used for traffic generation and is connected to the

Class	Baseline	HyNIC
DDoS	0.3462	**0.9237**
Injection	0.3774	**0.5542**
Password	0.1406	**0.5137**
Ransomware	0.2623	**0.4081**
Scanning	0.0191	**0.2071**
XSS	0.6580	**0.7464**
Benign	**0.9795**	0.9743
Macro Avg	0.3976	**0.6182**
Weighted Avg	0.6507	**0.8799**

(a) ToN-IoT

Class	Baseline	HyNIC
DDoS HTTP Flood	0.7639	**0.9068**
DDoS ICMP Flood	1.0000	1.0000
DDoS TCP SYN Flood	**1.0000**	0.9997
DDoS UDP Flood	1.0000	1.0000
Password attacks	0.9232	**0.9703**
SQL injection	0.5434	**0.7457**
Uploading attack	**0.7254**	0.7041
Vulnerability scan	0.9519	**0.9712**
XSS attacks	0.3159	**0.6944**
Benign	1.0000	0.9999
Macro Avg	0.8224	**0.8992**
Weighted Avg	0.9866	**0.9935**

(b) Edge-IIoT

Table II: Per-class F1 scores, with best scores shown in bold.

IPU host server via an Intel Tofino switch to emulate realistic network conditions. We replay traces using `tcpreplay` [30] and generate high-speed traffic with `PktGen-DPDK` [31].

B. Datasets and metrics

We evaluate `HyNIC` using two widely used IoT/IIoT intrusion detection datasets that capture realistic traffic conditions and diverse attack behaviors. We focus on IoT settings due to their security relevance and heterogeneous, resource-constrained environments, though our approach is dataset-agnostic and can be applied to other traffic traces.

ToN-IoT [24] includes benign and attack traffic collected from a medium-scale testbed spanning cloud, fog, and edge environments. It covers multiple attack types (*e.g.*, ransomware, scanning, injection, denial-of-service), and we perform 7-class classification (6 attacks + benign).

Edge-IIoT [32] consists of a comprehensive mix of IoT/IIoT traffic generated from a heterogeneous testbed with diverse devices and protocols. It includes several attack categories (*e.g.*, denial-of-service, scanning, injection, malware), and we perform 10-class classification (9 attacks + benign).

For both datasets, we randomly split flows into 75% for training and 25% for testing, ensuring that packets from the same flow do not appear in both sets. We evaluate the classification performance of `HyNIC` using the F1 score computed as: $\text{F1} = \frac{2TP}{2TP+FP+FN}$. We report the per-class scores, as well as the macro and weighted averages.

C. Classification Performance

To evaluate classification performance under realistic traffic conditions, we replay test PCAP traces using `tcpreplay` at their original rates, preserving temporal characteristics and flow-level dependencies. Tables IIa and IIb report the per-class performance of `HyNIC` compared to a stateless-only baseline.

On ToN-IoT, `HyNIC` improves classification performance for nearly all attack classes, with gains of up to 22% in macro F1 and similar improvements in weighted F1. Except for benign traffic, `HyNIC` achieves higher F1 scores on all classes, indicating that incorporating runtime context improves both accuracy and class balance, particularly for under-represented attacks. On Edge IIoT, both approaches achieve similarly high weighted F1 scores (~ 0.99) due to near-perfect performance on four well-represented classes. However, differences emerge in the remaining six classes, where stateless inference is more challenging. Here, `HyNIC` achieves significant improvements, including over 37% for XSS, more than 20% for SQL injection, and around 15% for DDoS HTTP Flood, with moderate gains for other classes. These improvements result in a higher macro average F1, reflecting more balanced performance.

Finally, we perform an ablation study to assess the contribution of on-core feature computation by considering the scenario where stateful features are unavailable. In this case, `HyNIC` falls back to stateless inference. On ToN-IoT, macro F1 drops from 0.62 to 0.32 and weighted F1 from 0.88 to 0.59, yielding performance slightly below but close to the stateless baseline, which was optimized for stateless inference. This shows that while stateful context is key to peak accuracy, `HyNIC` degrades gracefully when it is unavailable.

D. System performance

Throughput. We use `PktGen-DPDK` to generate high-speed traffic while varying packet sizes to characterize throughput and packet-rate capability, measured via DPDK port statistics at the IPU. No packet loss is observed in any experiment. The system consistently achieves near line-rate throughput for larger packet sizes: with 1024 B packets, 97–99 Gbps at $\sim$12 Mpps, and with 512 B packets, 95–99 Gbps at $\sim$24 Mpps, indicating full utilization of the 100 Gbps link.

As packet sizes decrease, packet rate increases while maintaining high throughput. With 256 B packets, the IPU sustains 90–96 Gbps at $\sim$44 Mpps with no loss, showing that near line-rate performance is achievable at moderate packet sizes typical of IoT/IIoT traffic. Using a benign ToN-IoT PCAP trace, the system also sustains close to 100 Gbps, confirming near line-rate performance on realistic traffic.

Resource Usage. `HyNIC` uses an exact-match flow table to maintain per-flow state, allocating approximately 32,000 entries to track up to 32,000 concurrent flows at line rate using about 50 MB of on-chip SRAM. This footprint is modest and limited to active flows, with the possibility for larger or long-lived state to be backed by DRAM, ensuring SRAM capacity does not constrain scalability. Classification is implemented using a single ternary-match table with 1,024 entries, encoding the entire model and bounding inference to a single lookup stage. This supports DT models with up to 1,000 leaf nodes while avoiding the multi-stage ternary pipelines required by prior in-network inference approaches [8], [14], [16], [26].

V. Discussion

Technical novelty. `HyNIC` introduces a hybrid in-network inference framework that integrates data-plane execution with on-NIC cores interaction while preserving line-rate execution within a single data-plane stage. The model combines stateless and stateful features, leveraging stateless features for the first n packets of each flow, after which stateful context is computed and introduced at runtime from the cores, enabling both early and context-aware classification. We map the model to a single M/A table, avoiding the multi-stage designs in prior work.

Model choice and hardware constraints. `HyNIC` adopts DT models to meet SmartNIC data-plane constraints, where

inference must execute at line rate with tightly bounded resources. DTs enable compact, deterministic inference that maps naturally to a single match-action table, avoiding multi-stage pipelines or recirculation required by more complex models. While other programmable data planes (*e.g.*, switches) support richer models using deeper pipelines, SmartNIC approaches typically offload them to the host or control plane. In contrast, `HyNIC` enables fully on-path inference.

Latency and datapath execution. All inference in `HyNIC` executes on the data path, ensuring bounded and predictable latency. Classification is performed in a single lookup without recirculation or multi-pass processing, and packet forwarding is not stalled by NIC core activity. During feature updates by the cores, packets can still be classified using stateless features, allowing continued operation without interrupting traffic.

Generality. `HyNIC` is not tied to a specific dataset, application, or SmartNIC platform. Given a programmable data path and on-device cores, its hybrid stateless-stateful inference pattern and single-table model mapping are portable across similar platforms. Beyond intrusion and anomaly detection, `HyNIC` can support traffic classification, policy enforcement, QoS marking, and lightweight telemetry, where early packet features enable coarse decisions refined as flow context becomes available, without off-path inference.

Limitations and future work. `HyNIC` is constrained by SmartNIC data-plane resources, limiting model size and complexity compared to off-path approaches. Our evaluation relies on offline datasets and trace replay, which may not capture dynamic conditions or closed-loop behavior in live deployments, and comparison is limited to a stateless baseline. While data-plane inference reduces exposure to off-path attacks, the system may remain sensitive to adversarial traffic patterns (*e.g.*, short-flow floods) targeting early classification.

Future work will include extending `HyNIC` beyond anomaly detection to broader traffic analysis tasks and environments, and performing more comprehensive evaluations of control-plane/data-plane interaction, including update latency, coordination overhead, and scalability. Additional directions include leveraging on-NIC cores for selective refinement with more complex models, and exploring model drift detection, runtime adaptation, and efficient deployment of alternative models.

VI. Conclusion

We proposed `HyNIC`, a hybrid in-network inference system that combines line-rate data-plane classification with runtime flow-level context derived on SmartNIC cores. Our design demonstrates how coordinated data-plane and on-NIC core processing enables accurate, flow-aware inference without sacrificing performance, and establishes a practical foundation for deploying stateful ML in programmable network devices.

Acknowledgement

This work was partly funded by the Leverhulme Trust and EU Horizon SmartEdge (101092908, Innovate UK 10056403).

References

[1] H. Haugerud et al. A dynamic and scalable parallel network intrusion detection system using intelligent rule ordering and network function virtualization. *Future Gener. Comput. Syst.*, 124:254–267, 2021.

[2] K. Shaukat et al. A survey on machine learning techniques for cyber security in the last decade. *IEEE Access*, 8:222310–222354, 2020.

[3] K. He et al. Measuring control plane latency in SDN-enabled switches. In *Proc. of ACM SIGCOMM SOSR*, NY, USA, 2015. ACM.

[4] C. Zheng et al. In-network machine learning using programmable network devices: A survey. *IEEE Commun. Surv. Tutor.*, 26(2), 2024.

[5] R. Kapoor et al. ML-NIC: accelerating machine learning inference using smart network interface cards. *Front. Comput. Sci.*, 6:1493399, 1 2024.

[6] G. Siracusano et al. Re-architecting traffic analysis with neural network interface cards. In *Proc. of USENIX NSDI*, pp. 513–533, 2022.

[7] B. M. Xavier et al. Programmable switches for in-networking classification. In *Proc. of IEEE INFOCOM*, pp. 1–10, 2021.

[8] C. Zheng et al. Iisy: Hybrid in-network classification using programmable switches. *IEEE/ACM Trans. Netw.*, 32(3):2555–2570, 2024.

[9] M. F. Sada et al. Real-time in-network machine learning on P4-programmable FPGA SmartNICs with fixed-point arithmetic and taylor approximations. In *Proc. of PEARC*, New York, NY, USA, 2025.

[10] P. Castoldi et al. Programmable packet-optical network security and monitoring using DPUs with embedded GPUs [Invited]. *J. Opt. Commun. Netw.*, 17(2):A178–A195, Feb 2025.

[11] S. Elizalde et al. Accelerated anomaly detection on IoT traffic using SmartNICs. In *Proc. of IEEE GlobeCom*, 2025.

[12] R. A. Bakar et al. Next-generation intrusion prevention system using hardware-accelerated data processing units (DPUs). In *Proc. of IEEE ICC (ICC Workshops)*, pp. 1438–1443, 2025.

[13] K. Tasdemir et al. An investigation of machine learning algorithms for high-bandwidth SQL injection detection utilising BlueField-3 DPU technology. In *Proc. of IEEE SOCC*, pp. 1–6, 2023.

[14] G. Zhou et al. An efficient design of intelligent network data plane. In *32nd USENIX symposium on security*, 2023.

[15] P. Bosshart et al. P4: programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[16] C. Zheng et al. Planter: Rapid prototyping of in-network machine learning inference. *SIGCOMM Comput. Commun. Rev.*, 54(1), 2024.

[17] A. T.-J. Akem et al. Practical and general-purpose flow-level inference with random forests in programmable switches. *IEEE Transactions on Networking*, 33(5):2489–2506, 2025.

[18] R. Parizotto et al. Offloading machine learning to programmable data planes: A systematic survey. *ACM Comput. Surv.*, 56(1), August 2023.

[19] E. F. Kfoury et al. A comprehensive survey on SmartNICs: Architectures, development models, applications, and research directions. *IEEE Access*, 12:107297–107336, 2024.

[20] Bluefield data processing unit (DPU). https://www.nvidia.com/en-gb/networking/products/data-processing-unit/, 2024.

[21] Agilio CX SmartNICs. https://netronome.com/agilio-smartnics/.

[22] Intel Infrastructure Processing Unit (IPU). https://www.intel.com/content/www/us/en/products/details/network-io/ipu/adapter-e2100.html.

[23] S. Elizalde et al. A survey on security applications with SmartNICs: Taxonomy, implementations, challenges, and future trends. *Journal of Network and Computer Applications*, 242:104257, 2025.

[24] A. Alsaedi et al. TON_IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems. *IEEE Access*, 8:165130–165150, 2020.

[25] Tshark. https://www.wireshark.org/docs/man-pages/tshark.html.

[26] A. T.-J. Akem et al. Jewel: Resource-efficient joint packet and flow level inference in programmable switches. In *IEEE INFOCOM*, 2024.

[27] L. Grinsztajn et al. Why do tree-based models still outperform deep learning on typical tabular data? In *In Proc. of NeurIPS*, 2022.

[28] F. Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2011.

[29] G. Xie et al. Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation. In *IEEE INFOCOM 2022*, pp. 1938–1947.

[30] A. Turner and F. Klassen. Tcpreplay. https://tcpreplay.appneta.com/.

[31] K. Wiles. Pktgen. https://pktgen-dpdk.readthedocs.io/en/latest/.

[32] M. A. Ferrag et al. Edge-IIoTset: A new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning. *IEEE Access*, 10:40281–40306, 2022.