
# FENIX: Enabling In-Network DNN Inference with FPGA-Enhanced Programmable Switches

*Xiangyu Gao*[§, ⋆] *Tong Li*[‡] *Yinchao Zhang*[§] *Ziqiang Wang*[†, §]
*Xiangsheng Zeng*[•] *Su Yao*[§, ◇, *] *Ke Xu*[§, ⋆, *]
[§]*Tsinghua University* [⋆] *Zhongguancun Laboratory* [‡]*Renmin University of China*
[†]*Southeast University* [◇]*BNRist* [•]*Huazhong University of Science and Technology*

## Abstract

Machine learning (ML) is increasingly used in network data planes for advanced traffic analysis, but existing solutions (such as FlowLens, N3IC, BoS) still struggle to simultaneously achieve low latency, high throughput, and high accuracy. To address these challenges, we present FENIX, a hybrid in-network ML system that performs feature extraction on programmable switch ASICs and deep neural network inference on FPGAs. FENIX introduces a Data Engine that leverages a probabilistic token bucket algorithm to control the sending rate of feature streams, effectively addressing the throughput gap between programmable switch ASICs and FPGAs. In addition, FENIX designs a Model Engine to enable high-accuracy deep neural network inference in the network, overcoming the difficulty of deploying complex models on resource-constrained switch chips. We implement FENIX on a programmable switch platform that integrates a Tofino ASIC and a ZU19EG FPGA directly, and evaluate it on real-world network traffic datasets. Our results show that FENIX achieves microsecond-level inference latency and multi-terabit throughput with low hardware overhead, and delivers over 90% accuracy on mainstream network traffic classification tasks, outperforming the state of the art.

## 1 Introduction

Machine learning (ML) is increasingly transforming networked systems. Recent work demonstrates that both traditional ML techniques—such as decision trees, random forests, and support vector machines—and deep learning (DL) approaches can significantly improve networking tasks, including malicious traffic detection [24, 33, 40, 71] and application traffic classification [12, 64, 71]. These advances have been enabled by the emergence of programmable network devices, including P4 switches [13], NetFPGA [41], and SmartNICs [23]. By integrating data-driven learning models with traditional rule-based mechanisms, these systems enable more adaptive and responsive in-network traffic analysis.

Early intelligent network designs typically split responsibilities between planes: the programmable data plane extracts traffic features, while the control plane performs ML inference. Although this leverages existing programmable infrastructure, it introduces communication latencies—often milliseconds [12, 54]. Such delays are especially problematic for time-sensitive applications like intrusion detection, as analysis may lag behind real-time traffic (see § 7 for details).

To address these latency challenges, researchers have explored moving ML inference closer to the data path. One promising approach is implementing inference directly on programmable devices such as SmartNICs, as demonstrated by systems like N3IC [50]. This offloads computation from endpoint CPUs and significantly reduces inference latency. However, new bottlenecks arise: for example, N3IC achieves maximum throughput of only 40 Gbps, and even the latest commercial SmartNICs are limited to 400 Gbps [42]. Such throughput is often insufficient for core networks, where switch ASICs handle 4 to 12Tbps [30, 31], far exceeding current SmartNIC capabilities. While deploying multiple NICs can scale throughput, this approach introduces significant costs and new scheduling challenges, as packet scheduling consumes significant CPU resources [26].

Recent advances in in-network ML have led researchers to implement ML models directly on programmable switch ASICs. For example, systems such as Leo [33] and NetBeacon [71] deploy decision tree and random forest models on Tofino switches using Match Action Tables (MAT), while BoS [64] demonstrates the feasibility of running binary recurrent neural networks (RNNs) on similar hardware. These approaches can potentially enable line-rate ML inference at significantly higher throughput. However, they are typically subject to stringent computational resource constraints, which limit their effectiveness for complex multi-classification tasks (see § 7 for details).

These observations raise several key questions: Why do existing switch ASICs struggle to support more complex ML tasks? Beyond the programmable devices discussed above, what other platforms can balance accuracy and latency for

*Ke Xu and Su Yao are the corresponding authors.

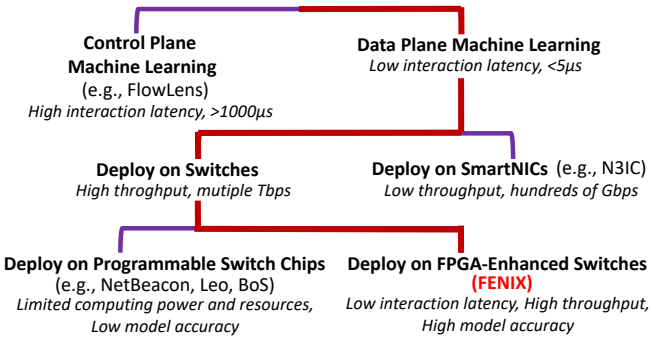



Figure 1: Design space in Intelligent Network.

in-network ML? In this paper, we present FENIX (FPGA-Enabled Neural Inference eXecution for network switches), a system designed to achieve low latency, high throughput, and high accuracy for in-network intelligent traffic analysis.

Figure 1 surveys intelligent network deployment methods and positions FENIX within this landscape. Unlike Control Plane approaches such as FlowLens [12], which incur high interaction latency (>1000$\mu$s), FENIX adopts data plane Machine Learning to achieve sub-microsecond latency (<5$\mu$s). Within the data plane category, FENIX is implemented on switches rather than SmartNICs (e.g., N3IC [50]), supporting multi-terabit-per-second throughput compared to only hundreds of Gbps with SmartNIC-based solutions. Furthermore, FENIX leverages FPGA-enhanced switches instead of solely relying on programmable switch ASICs (e.g., NetBeacon [71], Leo [33], BoS [64]), thereby overcoming their computational and resource limitations and enabling higher model accuracy while retaining low latency and high throughput.

FENIX consists of two main components: the Data Engine and Model Engine, which enable in-network traffic analysis on the data plane. The Data Engine extracts features from high-throughput traffic, while the Model Engine performs ML inference on these features. This design enables FENIX to achieve three key goals: *low latency* by avoiding software processing, *high throughput* through efficient feature extraction, and *high accuracy* by supporting full DNN models with minimal quantization loss.

**Contributions.** The main contribution of this paper is the design, implementation, and evaluation of FENIX, the first system to leverage FPGA augmentation to extend the capabilities of programmable switches for in-network machine learning. Over six months, we designed and manufactured a custom switch, integrating a programmable Tofino switch chip and a ZU19EG FPGA on Printed Circuit Board (PCB). We evaluate FENIX on challenging tasks, including VPN encrypted traffic classification and malware detection. Experimental results show that FENIX achieves up to 537× lower inference latency compared to control plane-based approaches, and improves classification accuracy by up to 21% over SOTA. Our prototype shows low hardware resource overhead, indicating that FPGA-enhanced switches are practical for deployment in high-speed network environments.

## 2 BACKGROUND AND MOTIVATION

This section reviews the evolution of DNN models for networking, examines the limitations of programmable switches, highlights the promise of FPGA-assisted networking, and outlines the integration challenges that motivate our hybrid system design.

**DNN Learning Models for Networking.** Classification tasks such as malicious traffic detection [24, 33, 40, 71] and application traffic classification [12, 64, 71] are fundamental for network intelligence.Prior work has demonstrated the feasibility of in-network AI inference by deploying lightweight models such as decision trees [33], random forests [71], XGBoost [71], binarized MLPs [50], and simplified RNNs [64] on programmable switches and SmartNICs. While these models enable practical deployment, their limited representational power often makes it difficult to capture complex or dynamic network behaviors. In contrast, more complex DNNs offer stronger nonlinear modeling capabilities and can adapt more effectively to evolving network conditions, leading to improved accuracy on diverse networking tasks. As a result, supporting DNN inference in network devices has the potential to further enhance the flexibility and precision of in-network intelligence, especially in scenarios where lightweight models face limitations.

**Programmable Network Data Plane Limitations.** Despite the promise of in-network DNN inference, the resource constraints of programmable data planes present significant challenges for deploying sophisticated models. PISA-based architectures are effective for basic machine learning tasks [33, 71], but encounter fundamental limitations with complex neural networks. The PISA instruction set supports only simple operations (e.g., addition, subtraction, bit-shifting, and logical operations), and lacks native support for floating-point arithmetic, multiplication, division, and complex conditionals—operations essential for modern DNNs. Hardware resources are also limited: for example, commercial switches such as Barefoot Tofino 1 offer only 12 pipeline stages and constrained memory resources (120 Mbit SRAM, 6.2 Mbit TCAM) [64], which are insufficient for parameter-rich neural networks. Furthermore, restrictions such as atomic register access prevent efficient implementation of iterative computations required by DNNs. Collectively, these constraints make it impractical to support DNN inference on programmable switches while maintaining line-rate performance, underscoring the need for alternative approaches to bring advanced neural network capabilities into the data plane.

**FPGA-assisted Networking.** FPGAs offer a promising solution to the computational limitations of programmable data planes while preserving high-performance networking capabilities. These reconfigurable platforms can interface directly with programmable switch ASICs, providing microsecond-level latency for DNN-based processing via cus-

tomized hardware acceleration [1]. FPGAs strike a favorable balance among energy efficiency, computational flexibility, and performance—surpassing CPUs in energy efficiency and providing more versatile analytical capabilities than fixed-function switch ASICs [66]. Their support for partial dynamic reconfiguration enables network operators to update analytical functions without disrupting service [20], offering valuable operational flexibility. In hybrid architectures with programmable switch ASICs, these advantages are particularly valuable: conventional packet processing remains in the high-speed switch pipeline, while computation-intensive neural network inference is offloaded to specialized FPGA hardware. This clear division of labor enables the practical deployment of DNN models even in resource-constrained network environments.

**Mismatch between FPGA and Switch.** Despite their complementary capabilities, fundamental architectural differences between programmable switches and FPGAs present significant integration challenges. The two components operate in fundamentally different ways: switches process packets in deterministic pipelines with nanosecond precision, whereas FPGAs perform variable-latency neural computations requiring sophisticated synchronization. More critically, there is a substantial throughput gap between these chips: modern programmable switches operate at multi-terabit-per-second rates [30], while even the latest commercial FPGAs sustain only hundreds of Gbps [1]. This order-of-magnitude disparity makes it infeasible to offload all switch traffic directly to FPGAs for processing. As a result, a central challenge in designing FPGA-assisted networking systems is to efficiently extract and route only the most relevant feature information from high-speed network streams to the FPGA for deep processing, while preserving overall throughput and low latency.

**Motivation.** Current research in-network traffic analysis mainly follows two directions: adapting machine learning models to fit data plane constraints (e.g., transforming decision trees into match-action table representations) [33, 35, 60, 64, 71], or sampling data for offline analysis on servers [12, 34, 69]. However, both approaches inevitably compromise either model accuracy or introduce significant latency, and thus fall short of modern network requirements. The core challenge is to deploy powerful deep neural networks (DNNs) in high-speed network environments. While DNNs excel at traffic classification and anomaly detection, they face a critical bottleneck: programmable switches offer limited computational resources and restricted instruction sets, making complex DNN execution infeasible; meanwhile, hardware accelerators such as FPGAs, though efficient for neural network computation, provide processing bandwidth orders of magnitude lower than the Tbps-level throughput of switches. Even the most advanced FPGAs cannot directly process network traffic from switching chips. Therefore, it is necessary to select key feature information for the FPGA and implement rate control to adapt to different traffic rates.

[1] An FPGA can be directly connected to a switching ASIC as an external module, as implemented in our system; this approach also applies to switching ASICs other than Tofino.

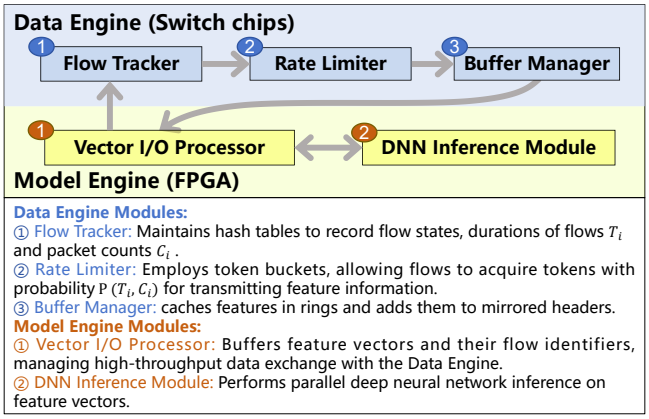



Figure 2: The architecture of FENIX.

**Goals.** Our research aims to develop FENIX, an intelligent network data plane system that simultaneously achieves three core technical goals:

• **Low latency:** FENIX uses FPGA parallelism and direct interfacing with programmable switching ASICs to support neural network inference with microsecond-level latency, without involving software stacks or system buses. This hardware-based approach minimizes processing overhead and maximizes data throughput. As a result, FENIX ensures real-time responsiveness for latency-sensitive network applications.

• **High throughput:** FENIX achieves high throughput in two aspects. First, programmable switching ASICs process all network traffic at line rate, performing initial feature extraction without affecting forwarding performance. Second, a novel feature control mechanism optimizes the communication channel between switches and FPGAs by dynamically adjusting per-flow sampling rates. This approach maximizes effective bandwidth utilization while respecting hardware constraints.

• **High accuracy:** FENIX implements different DNNs (e.g. CNN, RNN) on FPGAs using fixed-point quantization, rather than simplified or binarized models. This approach maintains accuracy close to offline analysis and enables support for advanced pattern recognition and temporal analysis tasks.

## 3 FENIX Design Overview

Figure 2 shows the architecture of FENIX, which is designed to address two main challenges in integrating deep learning inference with programmable switches. The first challenge is enabling the data plane to execute deep neural network models for fine-grained traffic control. The second challenge is bridging the throughput gap between FPGAs and programmable switching ASICs.

To address the throughput mismatch, FENIX designs a Data Engine that manages feature caching and transmission control. The Flow Tracker is motivated by the need to enable

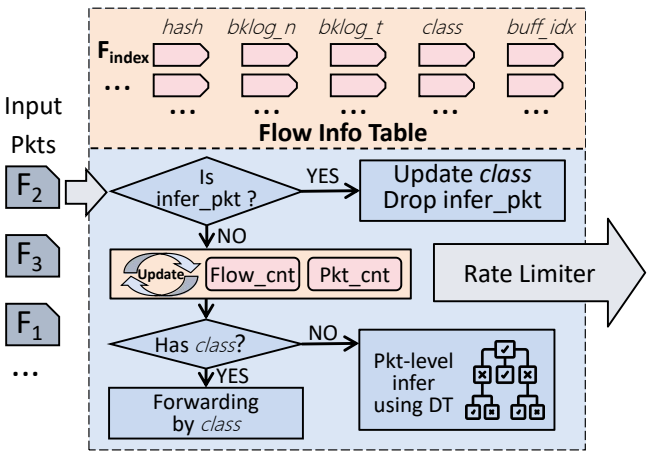



Figure 3: Workflow of Flow Tracker.

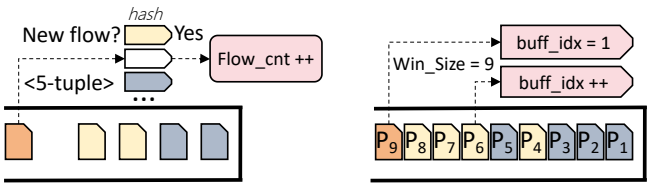
per-flow precise inference, providing the foundation for applying DNN models to individual network flows. The Rate Limiter employs a token bucket mechanism with probabilistic control to regulate the rate at which each flow sends feature information, preventing the FPGA from being overwhelmed by excessive feature data. The Buffer Manager uses a ring buffer structure with mirrored headers, enabling the data engine to pass stateful feature information to the model engine for processing.

To support DNN inference on the data path, FENIX designs a Model Engine responsible for both flow identification and model execution. The Vector I/O Processor is designed to support high-throughput, low-latency data exchange between the data engine and model engine, and caches flow identifiers so that the ASIC can accurately identify inference results. The DNN Inference Module performs parallel deep neural network inference on feature vectors, maximizing FPGA resource utilization.

With this design, FENIX achieves high-throughput, low-latency, and accurate DNN inference for in-network traffic analysis. The following sections detail each module design.

# 4 Data Engine

As shown in Figure 2, the Data Engine of FENIX is implemented on programmable switch ASICs and consists of three key modules working in concert. The Flow Tracker maintains complex hash table structures to accurately record state information for each network flow, including flow duration $T_i$ and packet count $C_i$, which serve as the basis for subsequent analysis. The Rate Limiter uses an enhanced token bucket algorithm to control the transmission frequency of feature information according to the computed probability function $P(T_i, C_i)$, effectively addressing throughput mismatch between the switch and FPGA, and dynamically adapting to current traffic rate. This probabilistic sampling method, based on flow features, helps maintain system stability under high-traffic conditions while ensuring quality of inputs for model




(1) Flow Counter (2) buff_idx Update

Figure 4: Details in Flow Tracker.

inference. To address FPGA inference latency that could potentially allow some malicious packets to pass undetected in per-packet inference scenarios, FENIX integrates lightweight decision trees as a complementary mechanism. For flows awaiting FPGA inference results, the system temporarily uses decision tree inference; once FPGA results become available, the flow processing strategy is immediately updated. The Buffer Manager employs ring buffer techniques to temporarily store feature data, attaches data to mirrored packet headers, and efficiently forwards traffic features to the Model Engine for further processing.

## 4.1 Flow Tracker

**Overview.** The Flow Tracker is a critical component of the Data Engine, responsible for tracking network flows and making per-flow decisions. As illustrated in Figure 3, it maintains a Flow Info Table in the switch's Static Random-Access Memory (SRAM), using truncated hash values of five-tuples (source IP, destination IP, source Port, destination Port, and Protocol) as unique flow identifiers. For each flow, the table records several key fields: the flow hash value (`hash`) for identifying new flow arrivals and handling table collisions; backlog packet count (`bklog_n`) and backlog timestamp (`bklog_t`) for tracking intervals between feature transmissions; classification results (`class`) for storing inference outcomes from the Model Engine; and buffer index (`buff_idx`), which records the flow's position in the Buffer Manager's ring buffer via modulo operation.

Upon packet arrival, the Flow Tracker first determines whether it is an inference packet from Model Engine. If so, it checks whether the packet belongs to a new flow or is the result of a hash collision, and then initializes or updates the corresponding flow entry. For flows with existing classification results, packets are forwarded based on these results. For flows without a classification, a lightweight decision tree implemented on the switch ASIC provides packet-level preliminary inference.

**Flow Counting Mechanism.** We define the start time of a flow as the instant when its first packet arrives at the switch, and count the number of flows that send packets within each timeout interval $T_w$. This statistical approach, which focuses on newly arrived flows within each interval, helps mitigate bias arising from missing the exact start or end times of flows [53]. The mechanism operates as illustrated in Figure 4 (1): the flow counter detects new flows by checking

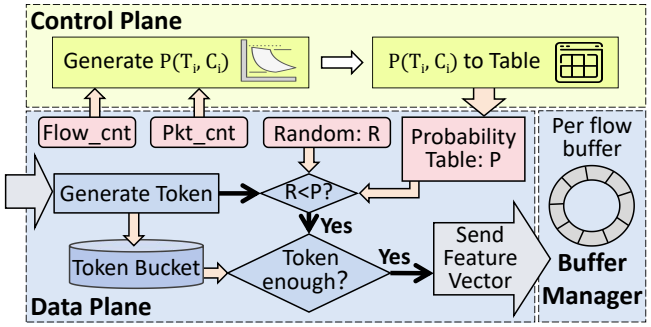



Figure 5: Workflow of Rate Limiter.

hash registers associated with each flow. Upon detecting a new flow, the counter increments $n$ by 1. At the end of each $T_w$ period, both the hash registers and the flow count $n$ are reset by the control plane to begin a new counting cycle, ensuring accurate and up-to-date flow statistics.

**Buffer Index Update.** As depicted in Figure 4 (2), since the data plane cannot perform modulo operations directly, we maintain a separate buffer index (`buff_idx`) for each flow that increments with each packet and resets to 1 when reaching buffer size. The flow's packet counter (`Pkt_cnt`) independently tracks total packets. This design enables a logical ring buffer where the buffer index cycles through available space, ensuring new data replaces old data in a circular manner.

## 4.2 Rate Limiter

**Overview.** Figure 5 illustrates our novel probabilistic rate limiter design, which integrates a dynamic probability model with the token bucket algorithm to intelligently allocate network traffic resources. This hybrid mechanism is specifically engineered to regulate interactions between the data engine and the model engine, ensuring fair opportunities for flow inference while preventing overload of the model engine.

The rate limiter maintains global traffic statistics within each timing window $T_w$, including `Flow_cnt` (the total number of flows) and `Pkt_cnt` (the total number of packets processed). While standard work-conserving algorithms such as WFQ (Weighted Fair Queuing) are mature solutions for ensuring inter-flow fairness, our goal is to achieve precise rate control based on per-flow state. In practice, it is infeasible to allocate physical queues for each flow, and alternative approximation schemes cannot guarantee precise rate matching between FPGA and programmable switches. Our probabilistic token bucket mechanism provides a lightweight, hardware-friendly alternative that can be efficiently implemented on existing switch architectures. Leveraging these statistics, the system achieves several advantages: high-speed flows are more likely to fail when requesting tokens, thereby preserving inference opportunities for lower-rate flows, and by capping the token bucket size to no more than the queue length, the system can effectively absorb traffic bursts without causing excessive queuing or packet drops.

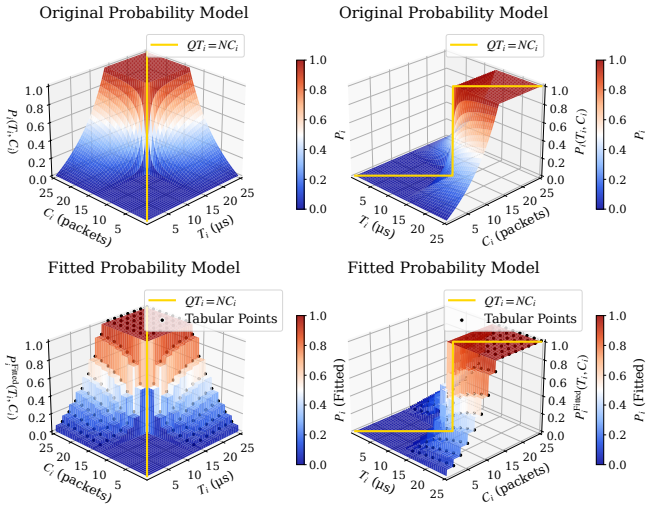



Figure 6: Probability curves of token generation model.

**Probability-Based Token Allocation Model.** From a theoretical perspective, the rate limiter constructs a probability function $P(T_i, C_i)$ based on real-time global traffic statistics—specifically, the global flow count $N$ and the aggregate packet rate $Q$. Here, $T_i$ denotes the elapsed time since flow $i$ last transmitted its features, and $C_i$ represents the number of packets transmitted by flow $i$ during this period. By considering both $T_i$ and $C_i$, the system can approximate the instantaneous rate of each flow. For each flow $i$, the token allocation is guided by two key criteria:

**Criterion 1**: In the idealized scenario where all flows operate at equal rates, each flow should receive tokens at an average interval of $\frac{N}{V}$, where $V$ is the token generation rate.

**Criterion 2**: In practical situations with heterogeneous flow rates, token allocation should be proportional to each flow's speed. Thus, the expected token acquisition interval for flow $i$ should approach $\frac{Q}{Q_i V}$, where $Q_i = \frac{C_i}{T_i}$ represents the average packet rate of flow $i$.

The token bucket generation rate $V$ is set according to the communication bandwidth $B$ between engines, FPGA frequency $F$, and feature vector width $W$:

$$V = \min(F, B/W) \tag{1}$$

Based on these criteria, we propose the following piecewise probability model.

$$P_i(T_i, C_i) = \begin{cases} \frac{C_i(VT_i - N)}{QT_i - NC_i}, & \text{if } T_i \in \left[\frac{N}{V}, \frac{QT_i}{C_i V}\right] \left(\frac{N}{V} < \frac{QT_i}{C_i V}\right) \\ \frac{T_i(VC_i - Q)}{NC_i - QT_i}, & \text{if } T_i \in \left[\frac{QT_i}{C_i V}, \frac{N}{V}\right] \left(\frac{N}{V} > \frac{QT_i}{C_i V}\right) \\ 1, & \text{if } QT_i = NC_i \text{ and } T_i \geq \frac{N}{V} \\ 0, & \text{if } QT_i = NC_i \text{ and } T_i < \frac{N}{V} \end{cases} \tag{2}$$

**Probability Model Deployment.** To intuitively demonstrate our probability model, Figure 6 plots the function curves under representative network settings. The illustrated scenario involves 1000 concurrent flows, with the model engine processing packets at 75 Mpps and the network sustaining a total

**Algorithm 1** Token Bucket Algorithm for Rate Limiter

1: **if** $T_{last} = 0$ **then**
2: $T_{last} \leftarrow T_{now}$, $gap \leftarrow 0$ ▷ Initialize for first packet
3: **else**
4: $gap \leftarrow T_{now} - T_{last}$, $T_{last} \leftarrow T_{now}$ ▷ Calculate time gap
5: **end if**
6: $rand \leftarrow$ Random(), $prob \leftarrow$ LookupProbability()
7: $bucket \leftarrow bucket + gap$ ▷ Refill tokens
8: **if** $rand < prob$ **then** ▷ Selected for sampling
9: **if** $bucket \geq cost$ **then**
10: $bucket \leftarrow bucket - cost$ ▷ Consume token
11: SendFeatureVector()
12: **end if**
13: **end if**

throughput of 1000 Mpps (approximately 800 Gbps, assuming an average packet size of 100 bytes). Given that the data plane cannot directly compute complex probability expressions, the rate limiter discretizes the model in the control plane by constructing a lookup table: the ranges of $T_i$ and $C_i$ are uniformly partitioned, enabling efficient mapping of all possible value pairs to probabilities in the range $[0, 1]$. Figure 6 presents both the original probability function and its table-based approximation, illustrating that our implementation closely preserves the intended behavior of the model.

**Token Bucket Algorithm Implementation.** In our data plane design, the rate limiter tracks three essential state variables within a custom timing window $T_w$: the number of active flows, the total packet count, and the last packet arrival time $T_{last}$. The details of our probabilistic token bucket mechanism are presented in Algorithm 1. Upon the arrival of each packet, the algorithm first computes the time interval between the current arrival time $T_{now}$ and $T_{last}$ to determine the appropriate token replenishment (lines 1–4). Next, a random number *rand* is generated and compared against the probability value *prob* retrieved from the precomputed lookup table (line 5). If the packet is probabilistically selected ($rand < prob$) and the token bucket contains enough tokens, the required token *cost* is deducted from the bucket and the feature vector is transmitted (lines 6–8). Otherwise, if the selection fails or tokens are insufficient, the algorithm only updates the token count based on the elapsed *gap* (lines 9–14).

**Discussion.** The Rate Limiter offers several notable advantages. By probabilistically denying token requests from high-speed flows, it helps reserve transmission opportunities for slower flows, promoting fairness across diverse traffic patterns. Additionally, by capping the token bucket capacity to not exceed the queue length, the system can efficiently accommodate bursty transmissions without risking buffer overflow. The probability model is computed in the control plane and distributed to the data plane as a lookup table. Overall, this module enables equitable and adaptable traffic management, while maintaining stability and robustness within the system.

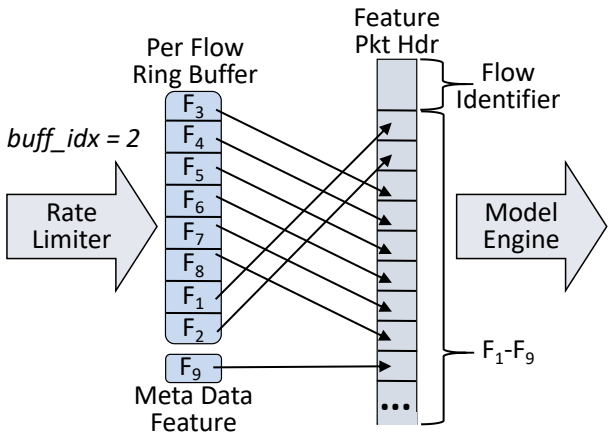



Figure 7: Mechanism of Buffer Manager.

## 4.3 Buffer Manager

**Ring Buffer Design.** Inspired by BoS [64], the Buffer Manager assigns a dedicated ring buffer to each network flow, which temporarily stores feature vectors until they are processed by the Model Engine. For each flow, the ring buffer holds features extracted from preceding packets ($F_1$–$F_8$), including packet intervals and packet lengths, as well as the current packet's feature ($F_9$, stored in metadata). This architecture allows the system to preserve the temporal and sequential relationships between packets within the same flow, offering a comprehensive context for subsequent analysis. When a buffer reaches capacity, it operates in a circular (FIFO) manner, overwriting the oldest features with new entries. This ensures that the buffer consistently reflects the most recent traffic characteristics, efficiently utilizes memory, and maintains the necessary flow context for downstream processing.

**Feature Transfer Process.** The Buffer Manager operates in close coordination with the Rate Limiter to manage feature transmission. As illustrated in Figure 7, when the Rate Limiter determines that feature information should be exported, the Buffer Manager reads the `buff_idx` value from the Flow Tracker (e.g., `buff_idx`=2 as shown in the figure) and extracts the corresponding feature vectors from the flow's ring buffer in SRAM, assembling them into the packet header. The newest feature, stored in metadata, is appended to the end of this header. During the final Deparser Stage, the Buffer Manager inserts the constructed header into a mirrored packet. These mirrored packets encapsulate both the flow's five-tuple identification and the sequence of feature vectors, providing the Model Engine with the necessary context for processing.

# 5 Model Engine

The Model Engine performs low-latency inference on traffic features from the Data Engine and is implemented on an FPGA. As shown in Figure 8, it comprises two components: a Vector I/O Processor which manages flow-info and a DNN Inference Module which executes model inference. Upon receiving packets, the Vector I/O Processor separates each

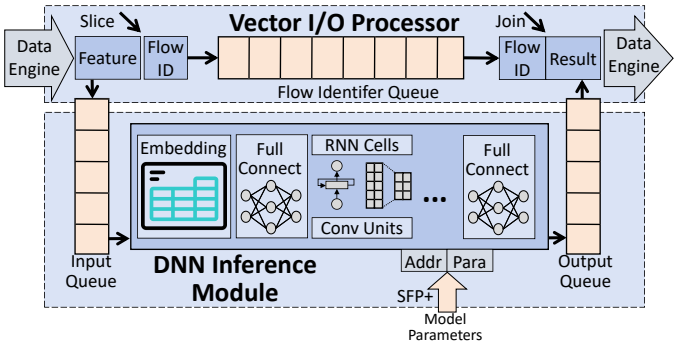



Figure 8: Workflow of Model Engine.

into flow identifier and feature vector. Flow identifiers are stored in the Flow Identifier Queue to preserve ordering, while feature vectors are forwarded to DNN Inference Module.

After inference completion, the results are placed in the Output Queue. Each inference result in the Output Queue is paired with the corresponding flow identifier retrieved from the front of the Flow Identifier Queue, forming a new packet with both fields. This packet is then transmitted back to the Data Engine, enabling the switch chip to execute follow-up operations on the relevant flow based on the inference result. This approach maintains correct mapping between flows and their inference results throughout processing.

## 5.1 Vector I/O Processor

The Vector I/O Processor is responsible for parsing incoming network packets to extract both the flow identifier (such as the five-tuple) and the associated feature vector. To maintain the correspondence between flows and their features throughout the inference process, the processor employs a FIFO queue to store flow identifiers. Meanwhile, the DNN Inference Module utilizes asynchronous FIFO queues for both its input and output, enabling seamless data transfer between modules operating under different clock domains. This asynchronous FIFO design not only decouples the timing dependencies between the Vector I/O Processor and the DNN Inference Module, but also improves system robustness and throughput.

During operation, the Vector I/O Processor continuously monitors the status of both the flow identifier FIFO and the output FIFO of the DNN Inference Module. Whenever both queues contain valid entries, the processor simultaneously dequeues the head elements, assembles the flow identifier and inference result into a new packet, and transmits this packet to the Data Engine. The Data Engine then forwards the result to the programmable switch, ensuring timely and accurate flow-level actions based on the inference outcome. For example, when a flow is identified as malicious based on the inference result, the programmable switch can record this status in its flow register. Subsequently, whenever packets belonging to this flow are encountered, the switch can enforce corresponding actions such as rate limiting or traffic isolation, thereby mitigating potential security threats in real time.

## 5.2 DNN Inference Module

The DNN Inference Module performs neural network inference for traffic analysis directly on the FPGA, as illustrated in Figure 8. The core computation uses a systolic array optimized for INT8 operations, enabling efficient matrix-vector multiplications for several common neural network layers. Our implementation supports embedding lookups mapped to LUTs, fully connected (FC) layers, convolution (Conv) layers, and recurrent layers. These layers are composed sequentially according to the model architecture, with all layer parameters and feature dimensions fixed at synthesis time to match application requirements and FPGA resource constraints. Although the implementation is not fully modular, each layer's computation is mapped onto the same systolic array, and the data path is managed to support the required layer ordering.

Model parameters, including weights and biases for all supported layers, are loaded from the host into on-chip memory via the network interface. During inference, feature vectors are processed in batches: embedding lookups are performed first, followed by the configured sequence of FC, Conv, and recurrent layers, all using offline-quantized INT8 arithmetic. Asynchronous FIFO queues decouple dataflow between layers and enable efficient pipelining. This design provides low-latency inference for network data while making efficient use of FPGA compute and memory resources.

# 6 Implementation

**Experimental Hardware.** As shown in Figure 9, our experiments are conducted on a high-performance programmable switch platform that integrates both FPGA and Tofino chips on a single board. The hardware is implemented using a 22-layer high-performance printed circuit board (PCB), utilizing back-drilling technology to balance 100 Gbps high-speed interconnection performance and manufacturing costs. The switching system is equipped with a dedicated power control network, supporting a core operating current of up to 100 A and peak control currents up to 300 A, with precise impedance control to ensure reliable power delivery. The overall design process included approximately half a month for requirement analysis, one month for schematic design, two months for PCB design, and one and a half months for PCB manufacturing, soldering, and prototype hardware debugging. The interface design and testing between FPGA and Tofino required an additional month, with software driver development and system-level integration carried out in parallel.

Figure 9 presents a photograph of the hardware prototype, with the primary components highlighted as follows:

❶ **Tofino 2 switch chip** [31]: Featuring 20 MAU stages, 200 Mbits of SRAM, and 10.3 Mbits of TCAM per pipeline. We configured this chip to efficiently implement our Data Engine modules while balancing resource utilization across pipeline stages. Multiple 100 Gbps

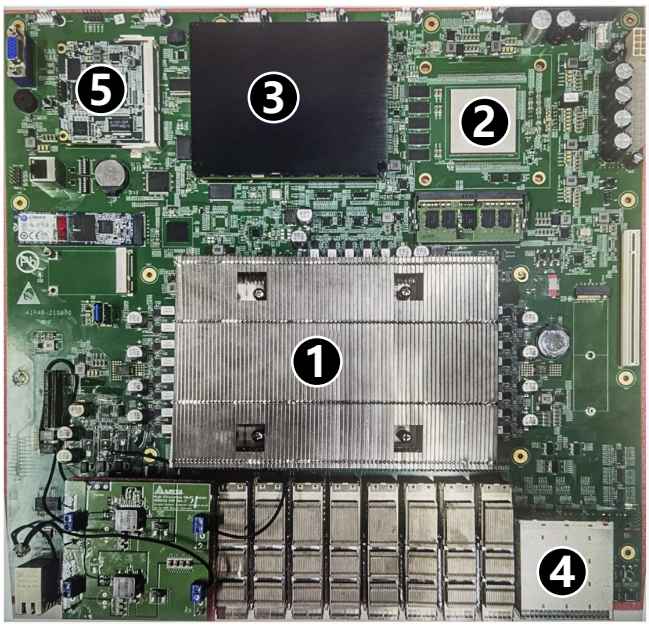



Figure 9: Hardware Platform of FENIX: The FPGA-Enhanced Programmable Switch Implementation.

port channels were designed to connect the Tofino chip to the FPGA, enabling high-bandwidth data exchange.

❷ **Xilinx ZU19EG FPGA chip** [1]: Integrates approximately 80 Mbits of on-chip memory and 1,143,450 logic elements. The FPGA resources were partitioned to accommodate the Model Engine's Vector I/O Processor and Neural Computing Array, while maintaining timing closure at high clock frequencies. The FPGA is directly connected to the Tofino chip through high-speed port channels, enabling low-latency communication.

❸ **Control plane CPU**: Programmed to manage the Tofino chip, responsible for configuration, monitoring, and probability model adjustments.

❹ **Four front-panel FPGA ports**: Designed with configurable transceivers to operate in either 10G or 25G modes for external communication. The design process required careful signal integrity analysis and impedance matching.

❺ **BMC module**: Customized for hardware status monitoring and remote management of the platform, including thermal management for high-performance components.

**Experimental Software.** Our development process encompassed multiple hardware and software platforms, requiring the integration of a diverse set of technologies and toolchains. For the Tofino chip, we used SDE version 9.8.0 for software development and debugging, while FPGA programming was conducted using Vivado and Vitis HLS 2024.1 [8]. The prototype system comprises: (1) over 1,500 lines of P4 code [13] to implement the Data Engine's three core modules (Flow Tracker, Rate Limiter, and Buffer Manager) on the programmable switch ASICs, which involved precise implementation of hash table maintenance, token bucket algorithms, and feature cache rings; (2) over 2,000 lines of HLS code for the Model Engine's two main components (Vector I/O Processor and DNN Inference Module) on the FPGA, responsible for parallel operations and efficient matrix processing; (3) over 1,500 lines of Python code for model training, testing, and quantization, ensuring efficient model execution under the resource constraints of the hardware; and (4) over 100 lines of Python code for calculating probability models in the control plane, enabling intelligent feature transmission decisions and adaptive flow management. Additionally, we developed an over 800 lines Python simulation program to evaluate and validate FENIX's performance and scalability under high-throughput and large-scale network scenarios.

**Model Training and Quantization.** We trained CNNs and RNNs for traffic classification using protocol-agnostic features: packet lengths and inter-packet arrival times. These features capture temporal patterns without requiring protocol-specific information. Our models include normalization layers, convolutional/recurrent units for temporal dependencies, and fully connected layers with ReLU activation. To address class imbalance, we applied oversampling and undersampling techniques during preprocessing.

After training, we used Vitis-AI [7] to quantize models from floating-point to INT-8 format for efficient FPGA deployment. The quantization process assigns different decimal positions to layers based on activation distributions to preserve accuracy. Our evaluation shows that quantization substantially reduces computation and storage requirements while maintaining classification performance with negligible degradation, enabling high-throughput, low-latency inference on resource-constrained FPGA platforms.

# 7 EVALUATION

Our evaluation addresses several key questions: (i) What classification accuracy can FENIX achieve on real traffic datasets, and how does it compare to other methods? (ii) What computational resources are required by FENIX, including both programmable switching ASICs during feature extraction and FPGA resources? (iii) How is FENIX performance affected under different scales of concurrent flows and throughput? (iv) As an asynchronous hybrid system, does FENIX introduce noticeable latency overhead when using FPGA for inference?

To answer these questions, we implemented FENIX and other baselines on an FPGA-enhanced programmable switch using P4 [13] and HLS [8] (Details in Section § 6), and conducted comparative evaluations. We used publicly available datasets for VPN encrypted traffic and malware detection for testing.

## 7.1 Methodology

**Testbed Setup.** We implemented FENIX on an FPGA-enhanced programmable switch (Details in Section § 6) us-

ing P4 and HLS, connected to a high-performance server equipped with dual Intel Xeon 5418Y processors and 512GB of memory. The server is equipped with two NVIDIA MCX75310AAS-NEAT 400G network interface cards, with one dedicated to traffic generation and the other to packet reception. The sending NIC generates synthetic traffic and replays pcap files using DPDK (v23.11.3) pktgen [18], while the receiving NIC collects and analyzes the processed packets using the DPDK environment.

**Schemes compared.** We compared FENIX with the following 9 schemes, including both flow-level and packet-level comparisons:

**(a) FENIX-CNN-FLOW:** We implemented a flow-level CNN with 3 convolutional layers and 2 fully connected layers. Flow accuracy is calculated based on majority voting of packet results within each flow.

**(b) FENIX-RNN-FLOW:** We developed a flow-level RNN with a single custom RNN cell and a dense output layer. The model processes packet length and IPD features through embeddings and classifies flows using the final hidden state.

**(c) FlowLens [12]:** We implemented packet statistics in Tofino using FlowLens' FMA code with control plane inference. Our XGBoost model used default parameters [59], with collection windows adjusted based on throughput.

**(d) FENIX-CNN-PKT:** We created a packet-level CNN with the same network architecture as FENIX-CNN-FLOW. This model processes individual packets independently, with accuracy calculated on per-packet classification results.

**(e) FENIX-RNN-PKT:** We implemented a packet-based RNN with the same network architecture as FENIX-RNN-FLOW. Accuracy is measured on a per-packet basis, evaluating the model's ability to correctly classify each individual packet.

**(f) Netbeacon [71]:** We implemented multi-phase tree models. Each phase uses a Random Forest (3 trees, depth 7) matching their configuration.

**(g) Leo [33]:** We implemented a decision tree (max depth 22, up to 1024 leaf nodes) on switches using packet length extremes and cumulative flow length.

**(h) BoS [64]:** We implemented the largest variant with a binarized GRU network (9-bit hidden states), 8 GRU units, 6-bit embeddings, and complete feature embedding-GRU-output architecture.

**(i) N3IC [50]:** We implemented a binary MLP on SmartNIC using flow-level and packet-level features with hidden layers [128, 64, 10]. Due to hardware constraints, we simulated switch-side logic and inference in software.

**Tasks.** We use the following tasks to evaluate FENIX, as summarized in Table 1. (i) Encrypted Traffic Classification on VPN: This task aims to classify network traffic that has been encrypted by VPNs. We use the ISCXVPN2016 dataset [25]. (ii) Malware Identification: This task distinguishes between traffic generated by benign applications and various types of malware. We use the USTC-TFC2016 dataset [57]. Note that model training is performed offline, and the FPGA is used exclusively for inference tasks.

Table 1: Experimental settings.

| Dataset (Task) | ISCXVPN2016 [25] | USTC-TFC [57] |
|---|---|---|
| **Training Flows** | 29,295 | 101,789 |
| **Test Flows** | 7,328 | 25,455 |
| **Number of Classes** | 7 | 12 |
| **Class Ratio** | 11:4:13:10: 18:128:1 | 92:10:4:14:17:23: 105:1:16:132:27:1 |
| **Optimizer** | AdamW | AdamW |
| **Learning Rate** | 0.01 | 0.005 |

**Metrics.** We use flow-level macro-F1 as the accuracy metric for FlowLens, while for other schemes we report packet-level macro-F1. We also provide a breakdown of Precision and Recall for each class to enable detailed performance analysis.

## 7.2 Classification Accuracy on Real Datasets

Table 2 presents a comprehensive comparison of accuracy and recall performance of different methods on encrypted traffic classification and malware detection tasks, evaluated at both the flow and packet levels. Across all categories in both tasks, FENIX consistently demonstrates superior accuracy compared to other baselines. At the flow level, the macro-F1 score of FENIX reaches 0.890, which is very close to that of FlowLens (0.870), indicating that FENIX can achieve state-of-the-art detection performance while retaining the advantages of a neural network-based approach.

When evaluated at the packet level, FENIX outperforms all other compared methods across both tasks. On the malware detection benchmark, FENIX achieves a macro-F1 score of 0.907, greatly surpassing tree-based approaches such as NetBeacon (0.670) and showing a clear margin over advanced neural baselines like N3IC (0.858). Compared to other neural models, including Leo (0.741) and BoS (0.814), FENIX also demonstrates consistently higher accuracy.

This performance gap becomes especially pronounced in challenging multiclass scenarios, where traditional methods often suffer from error propagation or limited packet-level representational capacity. While BoS and Leo achieve moderate results, they still lag behind FENIX, highlighting the benefits of high-precision inference enabled by the FENIX architecture. Notably, BoS is constrained by model binarization and the limited size of each component that switching ASICs can accommodate, which reduces its accuracy. Tree-based methods like NetBeacon can only update predictions at discrete points, further limiting performance on fine-grained, packet-level tasks. Although FENIX does not perform continuous inference for every packet, its higher per-inference accuracy ensures better overall packet-level accuracy. This advantage is particularly evident in complex multiclass settings, where the CNN-based FENIX model achieves the highest classification performance among all tested methods.

Table 2: Performance Comparison of Different Methods on Encrypted Traffic Classification and Malware Detection

| Class | FENIX $_{\text{F-CNN}}$ | FENIX $_{\text{F-RNN}}$ | FlowLens [12] | FENIX $_{\text{P-CNN}}$ | FENIX $_{\text{P-RNN}}$ | NetBeacon [71] | Leo [33] | BoS [64] | N3IC [50] |
|---|---|---|---|---|---|---|---|---|---|
| **Encrypted Traffic Classification (ISCXVPN2016 [25])** | | | | | | | | | |
| Chat | 0.883/0.852 | 0.939/0.882 | 0.862/0.922 | 0.917/0.836 | 0.804/0.653 | 0.627/0.169 | 0.489/0.353 | 0.922/0.913 | 0.533/0.527 |
| Email | 0.924/0.834 | 0.944/0.924 | 0.888/0.821 | 0.862/0.882 | 0.893/0.746 | 0.230/0.321 | 0.333/0.019 | 0.932/0.925 | 0.349/0.353 |
| File | 0.879/0.849 | 0.923/0.912 | 0.860/0.889 | 0.886/0.797 | 0.889/0.844 | 0.861/0.701 | 0.815/0.720 | 0.915/0.922 | 0.820/0.848 |
| P2P | 0.977/0.963 | 0.976/0.988 | 0.923/0.913 | 0.911/0.914 | 0.932/0.947 | 0.861/0.908 | 0.853/0.867 | 0.925/0.917 | 0.905/0.892 |
| Stream | 0.902/0.968 | 0.919/0.973 | 0.966/0.959 | 0.877/0.965 | 0.894/0.969 | 0.890/0.976 | 0.850/0.944 | 0.916/0.908 | 0.886/0.938 |
| Voip | 0.989/0.995 | 0.992/0.996 | 0.998/0.995 | 0.999/0.998 | 0.999/0.999 | 0.986/0.993 | 0.994/0.997 | 0.829/0.723 | 0.994/0.993 |
| Web | 0.803/0.662 | 0.793/0.625 | 0.700/0.525 | 0.800/0.861 | 0.869/0.821 | 0.860/0.405 | 0.784/0.046 | 0.729/0.623 | 0.856/0.524 |
| Macro-F1 | 0.890 | 0.912 | 0.870 | 0.892 | 0.873 | 0.658 | 0.578 | 0.863 | 0.738 |
| **Malware Detection (USTC-TFC [57])** | | | | | | | | | |
| Cridex | 0.999/1.000 | 0.999/1.000 | 0.999/0.998 | 0.999/1.000 | 0.996/1.000 | 0.933/0.997 | 0.984/0.995 | 0.983/0.996 | 0.866/0.861 |
| FTP | 0.999/0.998 | 0.999/0.998 | 0.999/0.998 | 0.997/1.000 | 0.993/1.000 | 0.993/0.485 | 0.928/0.986 | 0.986/0.791 | 0.848/0.853 |
| Geodo | 0.979/0.881 | 0.984/0.891 | 0.945/0.905 | 0.932/0.343 | 0.786/0.424 | 0.899/0.524 | 0.369/0.188 | 0.934/0.615 | 0.857/0.851 |
| Htbot | 0.962/0.987 | 0.959/0.989 | 0.964/0.984 | 0.932/0.986 | 0.938/0.982 | 0.830/0.782 | 0.897/0.935 | 0.919/0.937 | 0.871/0.867 |
| Neris | 0.921/0.594 | 0.888/0.732 | 0.902/0.817 | 0.766/0.473 | 0.736/0.575 | 0.665/0.469 | 0.584/0.453 | 0.761/0.610 | 0.852/0.847 |
| Nsis-ay | 0.985/0.970 | 0.971/0.982 | 0.985/0.986 | 0.959/0.968 | 0.962/0.972 | 0.984/0.912 | 0.918/0.960 | 0.963/0.968 | 0.860/0.855 |
| Warcraft | 0.998/0.999 | 0.996/0.994 | 0.995/0.993 | 1.000/1.000 | 0.999/1.000 | 0.890/1.000 | 0.995/0.997 | 0.956/0.998 | 0.855/0.859 |
| Zeus | 0.932/0.945 | 0.955/0.863 | 0.971/0.932 | 0.962/0.978 | 0.978/0.958 | 0.823/0.260 | 0.895/0.791 | 0.976/0.951 | 0.865/0.860 |
| Virut | 0.671/0.941 | 0.760/0.893 | 0.827/0.908 | 0.723/0.863 | 0.763/0.836 | 0.571/0.815 | 0.691/0.692 | 0.744/0.844 | 0.859/0.855 |
| Weibo | 0.684/0.822 | 0.702/0.927 | 0.745/0.804 | 0.655/0.821 | 0.685/0.817 | 0.649/0.789 | 0.653/0.708 | 0.652/0.837 | 0.862/0.859 |
| Shifu | 0.999/0.966 | 0.997/0.995 | 0.982/0.914 | 0.947/0.829 | 0.965/0.767 | 0.194/0.302 | 0.610/0.593 | 0.835/0.501 | 0.862/0.859 |
| SMB | 0.700/0.522 | 0.845/0.504 | 0.726/0.654 | 0.632/0.416 | 0.665/0.492 | 0.607/0.434 | 0.555/0.491 | 0.651/0.406 | 0.862/0.859 |
| Macro-F1 | 0.887 | 0.901 | 0.914 | 0.907 | 0.838 | 0.670 | 0.741 | 0.814 | 0.858 |

**Note:** The four FENIX columns represent models using flow-level statistics with CNN (FENIX $_{\text{F-CNN}}$) and RNN (FENIX $_{\text{F-RNN}}$), as well as packet-level statistics with CNN (FENIX $_{\text{P-CNN}}$) and RNN (FENIX $_{\text{P-RNN}}$).

## 7.3 Hardware Resource Utilization

Table 3 compares the hardware resource overhead of several representative P4 systems on programmable switches. The results show that FENIX achieves relatively low resource usage across multiple dimensions. For instance, its SRAM consumption (12.9%) is much lower than FlowLens (34.2%), indicating that the introduction of FPGA effectively reduces the resource overhead of P4 systems. In terms of TCAM usage, FENIX also stands out, with only 4.4% overhead compared to NetBeacon's 18.8%, highlighting that FENIX avoids the trade-off of reducing one resource at the cost of sharply increasing another. Beyond memory, FENIX maintains efficient use of bus bandwidth and pipeline stages, remaining below or comparable to most other systems. This balanced and predictable resource profile is enabled by the decoupled DataEngine design, which keeps FENIX 's overhead stable across different tasks and workloads.

Table 4 summarizes the FPGA resource utilization, where percentages represent the proportion of total available resources consumed by each module. The results show that LUTs and FFs are the primary resources consumed, particularly in the core computational components of the CNN and RNN modules. For example, the overall CNN module uses 38.4% of LUTs and 33.8% of FFs. While BRAM and DSP usage is higher in the overall modules (up to 7.1% and 8.1% for CNN, and 6.3% and 4.6% for RNN, respectively), it remains minimal within individual submodules, where most values are below 4%. This indicates that our design primarily leverages combinational and sequential logic to efficiently implement neural network inference on the FPGA, while memory and DSP consumption remains moderate. Overall, the resource utilization is low, leaving ample headroom for further optimization and larger deployments.

Table 3: P4 Systems Resource Overhead Comparison

| System | SRAM | TCAM | Bus | Stage |
|---|---|---|---|---|
| FENIX | 12.9% | 4.4% | 3.5% | 9 |
| FlowLens [12] | 34.2% | 0.0% | 2.4% | 9 |
| BoS [64] | 26.3% | 6.3% | 8.6% | 12 |
| Leo [33] | 26.9% | 9.0% | 5.2% | 12 |
| NetBeacon [71] | 11.6% | 18.8% | 6.4% | 12 |

## 7.4 Flow Count and Throughput Scalability

We conduct stress tests on FENIX under high-concurrency and high-throughput scenarios. Since original network trace files were collected in low-bandwidth networks, we generate high-throughput trace files by concurrently packaging flows with unique identifiers and reassigning timestamps. Figure 10 shows results where we progressively increase flow concurrency until the traffic generator's NIC bandwidth is saturated. FENIX easily handles this scale, with macro-F1 scores nearly identical to Table 2.

To evaluate larger scales, we build a simulator emulating FENIX's workflow. We configure maximum concurrent flow

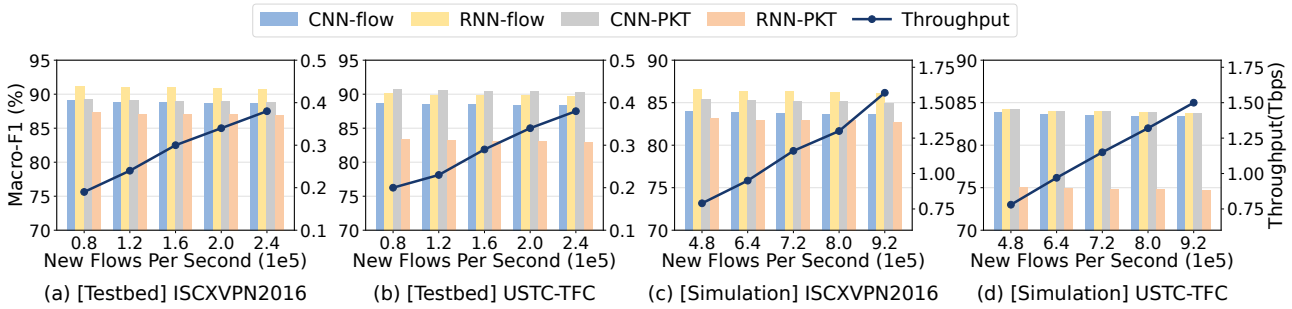



Figure 10: Scaling test of FENIX

Table 4: Neural Network Resource Utilization (%)

| Module | LUT | FF | BRAM | DSP |
|---|---|---|---|---|
| CNN (overall) | 38.4% | 33.8% | 7.1% | 8.1% |
| Embedding | 4.2% | 5.1% | 0.5% | 0.0% |
| Convolutional | 25.6% | 19.7% | 4.0% | 5.7% |
| FC | 8.6% | 9.0% | 2.6% | 2.4% |
| RNN (overall) | 25.6% | 31.2% | 6.3% | 4.6% |
| Embedding | 4.2% | 5.1% | 0.5% | 0.0% |
| Recurrent | 15.8% | 18.7% | 3.6% | 2.4% |
| FC | 8.6% | 9.2% | 2.2% | 2.2% |
| Vector I/O | 6.0% | 4.8% | 0.3% | 0.0% |

count based on switch chip registers and set inference latency according to measured results. After validating simulator accuracy against our testbed, we explore significantly larger scales with tens of thousands of flows per second and Tbps-level throughput. As shown in Figure 10, FENIX's macro-F1 score experiences only a minor decrease, with about a 13.2% reduction at the largest scale.

## 7.5 Latency Microbenchmark

We conduct a comprehensive latency breakdown comparing the control-plane inference approach (FlowLens [12]) with our proposed FENIX, as summarized in Figure 11. To capture microsecond-scale delays, we use an RTT-based method: internal transmission is measured via PCB interconnects, and external transmission via optical modules. Inference latency reflects the total model execution time for each system.

The results demonstrate that FENIX significantly outperforms FlowLens across all latency components. FlowLens, relying on general-purpose CPUs and control-plane communication, experiences transmission and inference delays in the millisecond range (2.1 ms for transmission and 1.5 ms for inference). In contrast, FENIX benefits from a tightly integrated FPGA architecture where the model and data engines communicate directly within the FPGA fabric, eliminating PCIe and memory bus overhead. This design achieves sub-microsecond internal transmission and 1–3$\mu$s external transmission. Most importantly, FENIX reduces inference latency by nearly three orders of magnitude, completing inference in just 1.2$\mu$s on

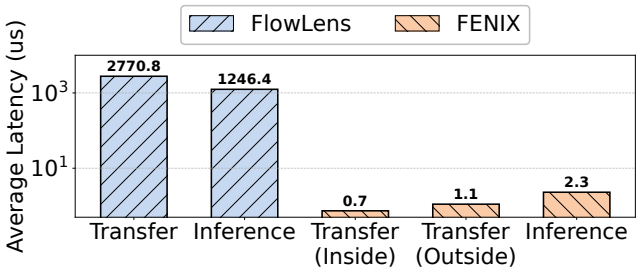



Figure 11: Latency Compare of FENIX and FlowLens.

average compared to FlowLens' $1,000+\mu$s. This improvement stems from in-network FPGA acceleration that offloads model execution from the host CPU and enables efficient, microsecond-scale inference.

## 8 Discussion

**Hardware Adaptability Beyond Tofino.** FENIX is prototyped using a Tofino 2 switch and Xilinx ZU19EG FPGA, but its architecture is fundamentally portable. The discontinuation of Intel's Tofino line [32] does not threaten the broader applicability of FENIX, as the programmable switch ecosystem remains rich, including Juniper Trio [65], Broadcom Trident [29], runtime programmable switch designs [21, 22, 61], and RMT-based research designs such as dRMT [15] and Menshen [56]. For example, Broadcom Trident 5's NetGNT integrates neural traffic analysis directly into switching silicon. However, publicly available information on such commercial products lacks detailed specifications of AI accelerator capabilities and programming interfaces, making it difficult for the research community to leverage these platforms for academic research and innovation. The hardware and software co-design of FENIX anticipates industry trends and provides a platform that can be easily adapted to new generations of programmable switches.

**Prototyping for ASIC Evolution.** While custom ASICs like Taurus [54] have set a precedent for per-packet ML inference in data planes—often validated using novel chips like Plasticine [43]—barriers to ASIC adoption remain high due to inflexible design and long development cycles. FENIX addresses this gap by providing a commercial, reprogrammable prototype that achieves high throughput and low latency at switch nodes, serving as both a practical solution and a refer-

ence for future ASIC designers. By demonstrating how DNN inference can be efficiently distributed across programmable switches and FPGAs, FENIX charts a pragmatic path toward intelligent data planes in production settings.

**Automation and Future Directions.** Recent frameworks such as Homunculus [55] highlight the growing need for automated ML pipelines and model deployment in data plane environments [63]. While FENIX currently requires expert-driven integration when migrating to new tasks or hardware, its modular Model Engine design is well-suited for future automation. However, the current FPGA implementation has inherent limitations regarding flexibility. Layer parameters and feature dimensions are fixed during synthesis, meaning that any DNN architecture updates or changes typically require complete hardware re-synthesis, which limits system adaptability. Our current design primarily supports common CNN and RNN models. Dynamic model updates and partial reconfiguration mechanisms can further enhance system flexibility, and these represent future research directions. Extending FENIX with high-level, task-agnostic abstractions, automated toolchains for FPGA and switch targets, and runtime dynamic model switching capabilities would further lower deployment barriers and accelerate its adoption across diverse operational scenarios.

## 9 Related Work

**ML for Network Traffic Analysis.** Machine learning has become a core tool for network traffic analysis, powering solutions for malicious traffic detection [6, 24, 40, 44], website fingerprinting [17, 46, 48], and fine-grained traffic classification [4, 49]. The field is rapidly advancing toward sophisticated models and tasks, including traffic prediction and encrypted flow analysis [38, 68, 70]. However, deploying models at line rate in data plane remains challenging due to hardware constraints, motivating innovations in model compression, quantization, and hardware-aware co-design.

**Data Plane Device Roles and Limitations.** Modern network infrastructures rely on programmable switches, SmartNICs, and FPGAs—each with distinct strengths. SmartNICs efficiently offload compute from servers for functions such as rate limiting, traffic analysis, and protocol acceleration [11, 45, 50, 51, 58]. Programmable switches provide unparalleled throughput and enable in-network tasks such as DDoS defense and real-time classification [2, 3, 5, 14, 33, 35, 36, 39, 60, 62, 64, 67, 71]. FPGAs serve as both prototyping platforms and as flexible data plane accelerators, often complementing the rigid resource budgets of ASIC-based switches [9, 19, 20, 27, 28, 37, 47, 52, 54, 66].

**Comparison and Positioning.** Several commercial products integrate hardware accelerators (FPGAs, DPUs, or AI accelerators) with programmable switch ASICs, such as APS Networks' APS6120Q [10] (Intel Tofino 3.2 Tbps ASIC with dual Intel Stratix 10 MX FPGAs) for broadband gateways and network security, and Cisco's smart switch series [16] for enterprise networking. However, these products do not publicly disclose accelerator-switch integration details, deployable model scales, or resource allocation strategies for network intelligence tasks. FENIX differs by providing a complete co-design methodology specifically for high-throughput DNN inference in the data plane, addressing the throughput gap challenge between heterogeneous chips and conducting real-world evaluations. Prior work either deploys quantized models in switches (sacrificing accuracy) or executes high-precision models at network edges (introducing latency). FENIX leverages tight FPGA-switch integration at central nodes to achieve practical trade-offs between inference accuracy and speed, demonstrating that advanced neural inference can be effectively integrated into switch-centric architectures. This work provides design insights that could inform both commercial deployments and future ASIC developments for intelligent network data planes. We envision these commercial products could serve as potential deployment platforms for FENIX, further advancing intelligent network technologies.

## 10 CONCLUSION

In this paper, we present FENIX, a hybrid in-network machine learning system designed to enable fast and accurate traffic analysis directly in the data plane. FENIX splits the workload by performing feature extraction on programmable switches and delegating DNN inference to FPGAs, allowing the system to achieve both high throughput and low latency without sacrificing accuracy. To address the bandwidth gap between switches and FPGAs, FENIX introduces a token bucket-based mechanism to efficiently regulate feature transmission. Our implementation and evaluation on real-world traffic show that FENIX delivers microsecond-level inference latency, multi-terabit throughput, and over 90% classification accuracy with minimal hardware overhead. Our results serve as a compelling proof-of-concept for the viability of future intelligent data-plane switches.

## Acknowledgements

We thank the anonymous reviewers and our shepherd, Eric Keller, for their insightful comments and suggestions. This work is supported by the National Science Foundation for Distinguished Young Scholars of China under Grant No. 62425201; the Science Fund for Creative Research Groups of the National Natural Science Foundation of China under Grant No. 62221003; the National Natural Science Foundation of China under Grant Nos. 62472240, 62394322, U22B2031, 62202473, and 62572473; the Taishan Scholar Foundation of Shandong Province under Grant No. tstp20250724; and the Beijing National Research Center for Information Science and Technology under Grant No. BNR2025RC01010. Ke Xu (xuke@tsinghua.edu.cn) and Su Yao (yaosu@tsinghua.edu.cn) are the corresponding authors.

## References


[1] Advanced Micro Devices (AMD). AMD Zynq™ UltraScale+™ MPSoC. https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-ultrascale-plus-mpsoc.html. Accessed: 2025-01.

[2] Aristide Tanyi-Jong Akem, Beyza Bütün, Michele Gucciardo, Marco Fiore, et al. Jewel: Resource-efficient joint packet and flow level inference programmable switches. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2024.

[3] Aristide Tanyi-Jong Akem, Michele Gucciardo, and Marco Fiore. Flowrest: Practical flow-level inference programmable switches with random forests. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–10. IEEE, 2023.

[4] Khaled Al-Naami, Swarup Chandra, Ahmad Mustafa, Latifur Khan, Zhiqiang Lin, and Bhavani Hamlen, Kevand Thuraisingham. Adaptive encrypted traffic fingerprinting with bi-directional dependence. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 177–188, 2016.

[5] Albert Gran Alcoz, Vincent Strohmeier, Martand Lenders, and Laurent Vanbever. Aggregate-based congestion control for pulse-wave ddos defense. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 693–706, 2022.

[6] João Romeiras Amado, Francisco Pereira, David Pissarra, Salvatore Signorello, Miguel Correia, and Fernando M. V. Ramos. Peregrine: Ml-based malicious traffic detection for terabit networks. *arXiv preprint arXiv:2403.18788*, 2024.

[7] AMD. *Vitis AI 3.5 Documentation*. Accessed: 2025-01.

[8] AMD. *Vitis High-Level Synthesis User Guide (UG1399)*. Accessed: 2025-01.

[9] Muhammad Bilal Anwer, Murtaza Motiwala, Muhammad Mukarram BTariq, and Nick Feamster. Switchblade: a platform for rapid deployment of network protocols. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 183–194, 2010.

[10] APS Networks GmbH. APS6120Q Advanced Programmable Switch. https://www.aps-networks.com/wp-content/uploads/2021/06/210616_APS6120Q_preliminary.pdf, 2021. Product Datasheet, Accessed: 2025-10.

[11] Mina Tahmasbi Arashloo, Alexey Lavrov, Manya Ghobadi, Jennifer Rexford, David Walker, and David Wentzlaff. Enabling programmable transport protocols high-speed nics. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 93–109, 2020.

[12] Diogo Barradas, Nuno Santos, Luís Rodrigues, Salvatore Signorello, Fernando M. V. Ramos, and André Madeira. Flowlens: Enabling efficient flow classification for ml-based network security applications. In *Network and Distributed System Security Symposium (NDSS)*, 2021.

[13] Pat Bosshart, Dan Daly, Glen Gibb, Nick Izzard, Martand McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, George Vahdat, Amand Varghese, and others. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[14] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. pforest: In-network inference with random forests. *arXiv preprint arXiv:1909.05680*, 2019.

[15] Sharad Chole, Andy Fingerhut, Sha Ma, Anirudh Sivaraman, Shay Vargaftik, Alon Berger, Gal Mendelson, Mohammad Alizadeh, Shang-Tse Chuang, Isaac Keslassy, et al. drmt: Disaggregated programmable switching. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 1–14, 2017.

[16] Cisco Systems. Cisco N9300 Series Smart Switches. https://www.cisco.com/site/us/en/products/networking/cloud-networking-switches/9300-series-smart-switches/index.html. Accessed: 2025-10.

[17] Xinhao Deng, Qilei Yin, Zhuotao Liu, Xiyuan Zhao, Qi Li, Mingwei Xu, Ke Xu, and Jianping Wu. Robust multi-tab website fingerprinting attacks the wild. In *IEEE Symposium on Security and Privacy (S&P)*, pages 1005–1022, 2023.

[18] DPDK Project. DPDK - The open source data plane development kit accelerating network performance. https://www.dpdk.org/. Accessed: 2025-01.

[19] Xinle Du, Tong Li, Guangmeng Zhou, Zhuotao Liu, Hanlin Huang, Xiangyu Gao, Mowei Wang, kun Tan, and ke Xu. Pred: Performance-oriented random early detection for consistently stable performance in datacenters. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 1–20, 2025.

[20] Weiqi Feng, Jiaqi Gao, Xiaoqi Chen, Gianni Antichi, Ran Ben Basat, Michael Mingchao Shao, Ying Zhang, and Minlan Yu. F3: Fast and flexible network telemetry with an fpga coprocessor. *Proceedings of the ACM on Networking*, 2(CoNEXT4):1–22, 2024.

[21] Yong Feng, Zhikang Chen, Haoyu Song, Wenquan Xu, Jiahao Li, Zijian Zhang, Tong Yun, Ying Wan, and Bin Liu. Enabling in-situ programmability in network data plane: From architecture to language. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 635–649, 2022.

[22] Yong Feng, Zhikang Chen, Haoyu Song, Yinchao Zhang, Hanyi Zhou, Ruoyu Sun, Wenkuo Dong, Peng Lu, Shuxin Liu, Chuwen Zhang, et al. Empower programmable pipeline for advanced stateful packet processing. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 491–508, 2024.

[23] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. Azure accelerated networking:smartnics the public cloud. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 51–66, 2018.

[24] Chuanpu Fu, Qi Li, Meng Shen, and Ke Xu. Realtime robust malicious traffic detection via frequency domaanalysis. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 3431–3446, 2021.

[25] Gerard Drapper Gil, Arash Habibi Lashkari, Mohammad Mamun, and Ali A Ghorbani. Characterization of encrypted and vpn traffic using time-related features. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, pages 407–414, 2016.

[26] Shreyan Gupta, Jiacheng He, Vamsi Olagappan, Arvind Raghunathan, Ausav Foong, Konstantina Papagiannaki, and Jitendra Padhye. Loom: Flexible and efficient NIC packet scheduling. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 73–88, 2019.

[27] Hanlin Huang, Xinle Du, Tong Li, Haiyang Wang, Ke Xu, Mowei Wang, and Huichen Dai. Re-architecting buffer management in lossless ethernet. *IEEE/ACM Transactions on Networking*, 32(6):4749–4764, 2024.

[28] Hanlin Huang, Ke Xu, Tong Li, Zhuotao Liu, Xinle Du, and Xiangyu Gao. Diffecn: Differential ecn marking for datacenter networks. *IEEE/ACM Transactions on Networking*, 33(1):210–225, 2025.

[29] Broadcom Inc. Bcm78800: Strataxgs trident 5 ethernet switch series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm78800. Accessed: 2025-01.

[30] Intel. Intel® intelligent fabric processors. https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors.html. Accessed: 2025-01.

[31] Intel. Intel® tofino™ 2. https://www.intel.com/content/www/us/en/products/sku/218648/intel-tofino-2-12-8-tbps-20-stage-4-pipelines/specifications.html. Accessed: 2025-01.

[32] Intel. Product change notification: Tofino 2. https://cdrdv2-public.intel.com/827577/PCN827577-00.pdf. Accessed: 2025-01.

[33] Syed Usman Jafri, Sanjay Rao, Vishal Shrivastav, and Mohit Tawarmalani. Leo: Online ML-based Traffic Classification at Multi-Terabit Line Rate. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 1573–1591, 2024.

[34] Sian Kim, Seyed Mohammad Mehdi Mirnajafizadeh, Bara Kim, Rhongho Jang, and D Nyang. Sketchfeature: High-quality per-flow feature extractor towards security-aware data plane. In *Network and Distributed System Security Symposium (NDSS)*, 2025.

[35] Jong-Hyouk Lee and Kamal Singh. Switchtree: in-network computing and traffic analyses with random forests. *Neural Computing and Applications*, pages 1–12, 2020.

[36] Guanyu Li, Menghao Zhang, Shicheng Wang, Chang Liu, Mingwei Xu, Ang Chen, Guofei Hu, Hongxand Gu, Qi Li, and Jianping Wu. Enabling performant, flexible and cost-efficient ddos defense with programmable switches. *IEEE/ACM Transactions on Networking*, 29(4):1509–1526, 2021.

[37] Tong Li, Kai Zheng, Ke Xu, Rahul Arvind Jadhav, Tao Xiong, Keith Winstein, and Kun Tan. TACK: improving wireless transport performance by taming acknowledgments. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 15–30, 2020.

[38] Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification. In *Proceedings of the ACM Web Conference (WWW)*, 2022.

[39] Zaoxing Liu, Hun Namkung, Georgios Nikolaidis, Jeongkeun Lee, Changhoon Kim, XJand Vladimir Braverman, Minlan Yu, and Vyas Sekar. Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric ddos attacks with programmable switches. In *USENIX Security Symposium (USENIX Security)*, pages 3829–3846, 2021.

[40] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. In *Network and Distributed System Security Symposium (NDSS)*, 2018.

[41] NetFPGA. NetFPGA. https://netfpga.org/. Accessed: 2025-01.

[42] NVIDIA. NVIDIA ConnectX-7 SmartNICs. https://www.nvidia.cn/networking/ethernet-adapters/. Accessed: 2025-01.

[43] Raghu Prabhakar, Yaqi Zhang, David Koeplinger, Matt Feldman, Tian Zhao, Stefan Hadjis, Ardavan Pedram, Christos Kozyrakis, and Kunle Olukotun. Plasticine: A reconfigurable architecture for parallel paterns. *ACM SIGARCH Computer Architecture News*, 45(2):389–402, 2017.

[44] Yuqi Qing, Qilei Yin, Xinhao Deng, Yihao Chen, Zhuotao Liu, Kun Sun, Ke Xu, Jia Zhang, and Qi Li. Low-quality training data only? a robust framework for detecting encrypted malicious network traffic. In *Network and Distributed System Security Symposium (NDSS)*, 2024.

[45] Sivasankar Radhakrishnan, Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, George Porter, and Amin Vahdat. Senic: Scalable nic for end-host rate limiting. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 475–488, 2014.

[46] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. In *Network and Distributed System Security Symposium (NDSS)*, 2018.

[47] Mariano Scazzariello, Tommaso Caiazzi, Hamid Ghasemirahni, Tom Barbette, Dejan Kostić, and Marco Chiesa. A high-speed stateful packet processing approach for tbps programmable switches. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 1237–1255, 2023.

[48] Meng Shen, Yiting Liu, Liehuang Zhu, Xiaojiang Du, and Jiankun Hu. Fine-grained webpage fingerprinting using only packet length information of encrypted traffic. *IEEE Transactions on Information Forensics and Security (TIFS)*, 16:2046–2059, 2020.

[49] Meng Shen, Jinpeng Zhang, Liehuang Zhu, Ke Xu, and Xiaojiang Du. Accurate decentralized application identification via encrypted traffic analysis using graph neural networks. *IEEE Transactions on Information Forensics and Security (TIFS)*, 16:2367–2380, 2021.

[50] Giuseppe Siracusano, Salvator Galea, Davide Sanvito, Mohammad Malekzadeh, Gianni Antichi, Paolo Costa, Hamed Haddadi, and Roberto Bifulco. Re-architecting traffic analysis with neural network interface cards. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 513–533, 2022.

[51] Giuseppe Siracusano, Salvator Galea, Davide Sanvito, Mohammad Malekzadeh, Hamed Haddadi, Gianni Antichi, and Roberto Bifulco. Running neural networks on the NIC. *arXiv preprint arXiv:2009.02353*, 2020.

[52] Anirudh Sivaraman, Suvinay Subramanian, Mohammad Alizadeh, Sharad Chole, Shang-Tse Chuang, Anurag Agrawal, Hari Balakrishnan, Tom Edsall, and Nick Katti, Sachand McKeown. Programmable packet scheduling at line rate. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 44–57, 2016.

[53] Bruce Spang and Nick McKeown. On estimating the number of flows. In *Stanford Workshop on Buffer Sizing*, 2019.

[54] Tushar Swamy, Alexander Rucker, Muhammad Shahbaz, Ishan Gaur, and Kunle 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)*, 2022.

[55] Tushar Swamy, Annus Zulfiqar, Luigi Nardi, Muhammad Shahbaz, and Kunle Olukotun. Homunculus: Auto-generating efficient data-plane ml pipelines for datacenter networks. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 329–342, 2023.

[56] Tao Wang, Xiangrui Yang, Gianni Antichi, Anirudh Sivaraman, and Aurojit Panda. Isolation mechanisms for high-speedpacket-processing pipelines. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 1289–1305, 2022.

[57] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. Malware traffic classification using convolutional neural network for representation learning. In *International conference on information networking (ICOIN)*, pages 712–717, 2017.

[58] Ziqiang Wang, Zhuotao Liu, Xiaoliang Wang, Songtao Fu, and Ke Xu. Dip: unifying network layer innovations using shared l3 core functions. In *Proceedings of the 21st ACM Workshop on Hot Topics Networks (HotNets)*, pages 60–67, 2022.

[59] XGBoost Developers. Introduction to XGBoost Python Package. https://xgboost.readthedocs.io/en/stable/python/python_intro.html. Accessed: 2025-01.

[60] Guorui Xie, Qing Li, Yutao Dong, Yong Duan, Guangland Jiang, and Jingpu Duan. Mousika: Enable General In-Network Intelligence Programmable Switches by Knowledge Distillation. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 1938–1947, 2022.

[61] Jiarong Xing, Kuo-Feng Hsu, Matty Kadosh, Alan Lo, Yonatan Piasetzky, Arvind Krishnamurthy, and Ang Chen. Runtime programmable switches. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 651–665, 2022.

[62] Jiarong Xing, Qiao Kang, and Ang Chen. Netwarden: Mitigating network covert channels while preserving performance. In *USENIX Security Symposium (USENIX Security)*, pages 2039–2056, 2020.

[63] Wenquan Xu, Zijian Zhang, Yong Feng, Haoyu Song, Zhikang Chen, Wenfei Wu, Guyue Liu, Yinchao Zhang, Shuxin Liu, Zerui Tian, and Bin Liu. Clickinc: In-network computing as a service in heterogeneous programmable data-center networks. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, page 798–815, 2023.

[64] Jinzhu Yan, Haotian Xu, Zhuotao Liu, Qi Li, Ke Xu, Mingwei Xu, and Jianping Wu. Brain-on-switch: Towards advanced intelligent network data plane via nn-driven traffic analysis at line-speed. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 419–440, 2024.

[65] Mingran Yang, Alex Baban, Valery Kugel, Jeff Libby, Scott Mackie, Swamy Sadashivaiah Renu Kananda, Chang-Hong Wu, and Manya Ghobadi. Using trio: juniper networks' programmable chipset-for emerging in-network applications. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 633–648, 2022.

[66] Chaoliang Zeng, Layong Luo, Teng Zhang, Zilong Wang, Luyang Li, Wenchen Han, Nan Chen, Lebing Wan, Lichao Liu, Zhipeng Ding, et al. Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 1345–1358, 2022.

[67] Yinchao Zhang, Su Yao, Yong Feng, Kang Chen, Tong Li, Zhuotao Liu, Yi Zhao, Lexuan Zhang, Xiangyu Gao, Feng Xiong, Qi Li, and Ke Xu. Pegasus: A universal framework for scalable deep learning inference on the dataplane. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, page 692–706, 2025.

[68] Ruijie Zhao, Mingwei Zhan, Xianwen Deng, Yanhao Wang, Yijun Wang, Guan Gui, and Zhi Xue. Yet another traffic classifier: A masked autoencoder based traffic transformer with multi-level flow representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 5420–5427, 2023.

[69] Changgang Zheng, Zhaoqi Xiong, Thanh T Bui, Siim Kaupmees, Riyad Bensoussane, Antoine Bernabeu, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. IIsy: Practical in-network classification. *arXiv preprint arXiv:2205.08243*, 2022.

[70] Guangmeng Zhou, Xiongwen Guo, Zhuotao Liu, Tong Li, Qi Li, and Ke Xu. Trafficformer: An efficient pre-trained model for traffic data. In *2025 IEEE Symposium on Security and Privacy (S&P)*, pages 102–102, 2024.

[71] Guangmeng Zhou, Zhuotao Liu, Chuanpu Fu, Qi Li, and Ke Xu. An efficient design of intelligent network data plane. In *USENIX Security Symposium (USENIX Security)*, pages 6203–6220, 2023.