



Real-Time In-Network Machine Learning on P4-Programmable FPGA SmartNICs with Fixed-Point Arithmetic and Taylor Approximations

Mohammad Firas Sada
University of California, San Diego
La Jolla, California, USA
mfsada@ucsd.edu

John Graham
University of California, San Diego
La Jolla, California, USA
jjgraham@ucsd.edu

Mahidhar Tatineni
San Diego Supercomputer Center
University of California, San Diego
La Jolla, California, USA
mahidhar@sdsc.edu

Dmitry Mishin
University of California, San Diego
La Jolla, California, USA
dmishin@ucsd.edu

Thomas DeFanti
University of California, San Diego
La Jolla, California, USA
tdefanti@ucsd.edu

Frank Würthwein
University of California, San Diego
La Jolla, California, USA
fkw@physics.ucsd.edu

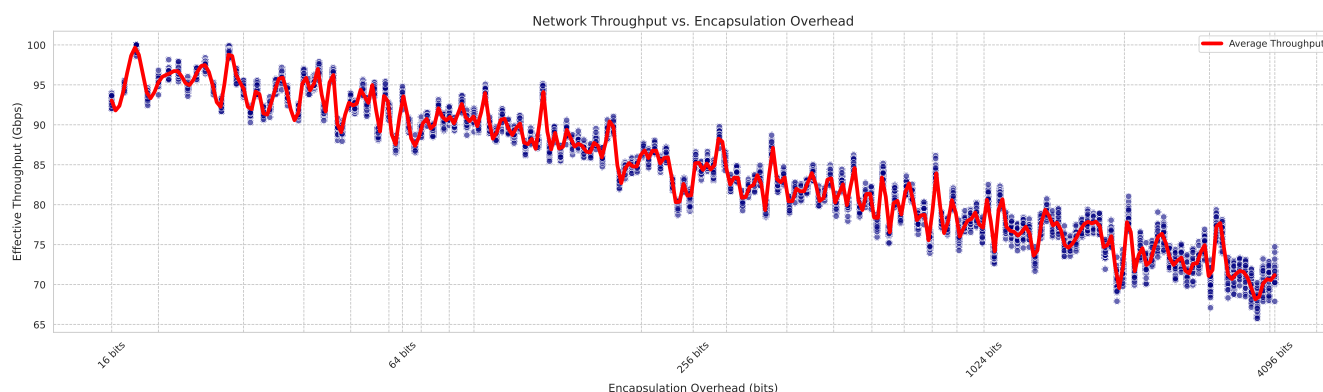


Figure 1: Impact of encapsulation header overhead (input features for regression models) on network throughput (Gbps) in a 100Gbps FPGA SmartNIC. Throughput for ingress and egress traffic is measured as a function of header bit size, which corresponds to the inclusion of input features (data embedded in packets for regression models) and directly affects processing efficiency. Output features (model predictions) are generated during egress and returned to the network. The FPGA model, implemented in P4, uses table lookups for retrainable models via the control plane, sacrificing some throughput for flexibility. To reduce arbitration, we assume input features and weights follow the same fractional and integer bits. The x-axis (logarithmic scale) shows encapsulation overhead (bits), and the y-axis shows measured throughput, with scatter points indicating individual measurements and a red line denoting average performance. Increasing overhead reduces throughput due to added processing demands, highlighting the trade-off between input features and network efficiency.

Abstract

As machine learning (ML) applications become integral to modern network operations, there is an increasing demand for network programmability that enables low-latency ML inference for tasks such as Quality of Service (QoS) prediction and anomaly detection in cybersecurity. ML models provide adaptability through dynamic weight adjustments, making Programming Protocol-independent Packet Processors (P4)-programmable[7] FPGA SmartNICs an ideal

platform for investigating In-Network Machine Learning (INML). These devices offer high-throughput, low-latency packet processing and can be dynamically reconfigured via the control plane, allowing for flexible integration of ML models directly at the network edge. This paper explores the application of the P4 programming paradigm to neural networks and regression models, where weights and biases are stored in control plane table lookups. This approach enables flexible programmability and efficient deployment of retrainable ML models at the network edge, independent of core infrastructure at the switch level.



This work is licensed under a Creative Commons Attribution 4.0 International License.
PEARC '25, Columbus, OH, USA

© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1398-9/25/07
<https://doi.org/10.1145/3708035.3736086>

CCS Concepts

• **Networks** → **Network protocols**; • **Software and its engineering** → **Real-time systems software**; • **Computing methodologies** → **Supervised learning by regression**; • **Hardware** → **Hardware accelerators**; **Arithmetic and datapath circuits**.

Keywords

P4 Programming, In-Network Computing, Fixed-Point Arithmetic, Network Machine Learning, Real-Time Inference, Packet Encapsulation, SmartNIC Acceleration, FPGA

ACM Reference Format:

Mohammad Firas Sada, John Graham, Mahidhar Tatineni, Dmitry Mishin, Thomas DeFanti, and Frank Würthwein. 2025. Real-Time In-Network Machine Learning on P4-Programmable FPGA SmartNICs with Fixed-Point Arithmetic and Taylor Approximations. In *Practice and Experience in Advanced Research Computing (PEARC '25)*, July 20–24, 2025, Columbus, OH, USA. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3708035.3736086>

1 Introduction

The integration of machine learning (ML) into network infrastructure has emerged as a critical enabler for real-time applications. However, traditional ML deployment models—reliant on centralized GPU/CPU clusters—introduce latency bottlenecks that hinder their effectiveness in time-sensitive, high-throughput environments. P4-Programmable network hardware, such as FPGA P4 SmartNICs, offers a transformative alternative by enabling computation directly within the data plane. Yet, the absence of native floating-point support and limited arithmetic operations in protocol-independent switch and SmartNIC architecture (P4) poses significant barriers to deploying ML models at line rate. This paper bridges this gap by presenting a novel framework for in-network ML inference that uses mathematical methods to convert ML models into programmable data planes and control planes deployed on the National Research Platform’s FPGA SmartNIC infrastructure.

1.1 The National Research Platform (NRP)

The National Research Platform (NRP) is a distributed, multi-tenant, Kubernetes-based cyberinfrastructure designed to facilitate collaborative scientific computing. Spanning over 75 sites internationally, the NRP uniquely integrates diverse computational resources, ranging from single nodes to extensive GPU and CPU clusters, to support various research workloads, including advanced AI and machine learning tasks. It emphasizes flexibility through user-friendly interfaces such as JupyterHub. The NRP includes more than 1,400 GPUs, utilized by over 300 research groups and numerous classrooms. More importantly, its 32 AMD/Xilinx Alveo U55C FPGAs provide immense capability for programmable network applications [11]. Each FPGA, equipped with two fully P4-programmable 100Gbps ports (using the AMD OpenNIC shell) and two PCIe-connected host ports, functions as a SmartNIC.

1.2 Motivation

The growing need for real-time, low-latency ML inference in network environments, such as academic clusters, drives the demand for decentralized, edge-centric intelligence. Traditional approaches

that offload ML computations to CPUs or GPUs via PCIe interfaces introduce significant latency, undermining their suitability for time-sensitive tasks like intrusion detection or QoS optimization. While FPGA SmartNICs are often used as PCIe-based accelerators, their true potential lies in enabling in-network computation through P4 programmability. However, deploying ML models directly on the data plane faces challenges, including P4’s lack of native support for floating-point arithmetic and regression operations. Our work addresses these limitations by approximating floating-point operations through fixed-point arithmetic using Taylor series expansions[9], enabling ML inference within the constraints of P4’s syntax and hardware. By embedding models into the data plane, we reduce PCIe communication from overheads by relying on control plane table lookups solely for the weights/coefficients, achieving microsecond-scale inference times critical for high-performance networks. This methodology not only enhances programmability but also opens avenues for adaptive cybersecurity frameworks and QoS mechanisms that evolve with dynamic network conditions, marking a significant step toward self-optimizing network infrastructures.

2 Methodology

The proposed approach uses the Alveo U55C FPGAs within the NRP for in-network inference. The ESnet SmartNIC tool stack [5] is used to provide an integrated software environment for OpenNIC shell, incorporating features such as DPDK-pktgen [3], probe counters, and automated FPGA flashing, all managed through a Docker Compose configuration [4].

The methodology consists of a software pipeline that transforms trained Python-based regression models into a format suitable for execution within the P4 data plane. After training, model weights and biases are serialized into a structured textual format, which is then parsed to generate P4 control plane table entries. These entries store the model parameters, including weight scaling factors, in a manner compatible with conventional P4 targets such as BMv2 [8]. The entire P4 syntax and control plane configuration are automatically synthesized and deployed onto the FPGA.

First, accuracy validation and computational trade-offs are assessed through software simulations. Initially, the trained model is executed on a CPU using Python to evaluate fixed-point arithmetic approximations and loss characteristics. Following this, BMv2 simulations [2] are conducted, utilizing traffic generated via Scapy [6], to verify correctness and expected packet behavior.

FPGA inference operates within the data plane, processing network packets with an appended header. Input features are extracted, and model weights are retrieved from control plane tables, eliminating registers/externs. After computation, the header is replaced with an output format for interoperability [10].

For low-latency ML inference in academic clusters like NRP[11], the motivation is to execute neural network inference directly in the P4 data plane. We approximate floating-point operations using fixed-point arithmetic and Taylor series expansions, avoiding PCIe offloading.

3 Mathematical Details

This section details the mathematical foundations of the model.

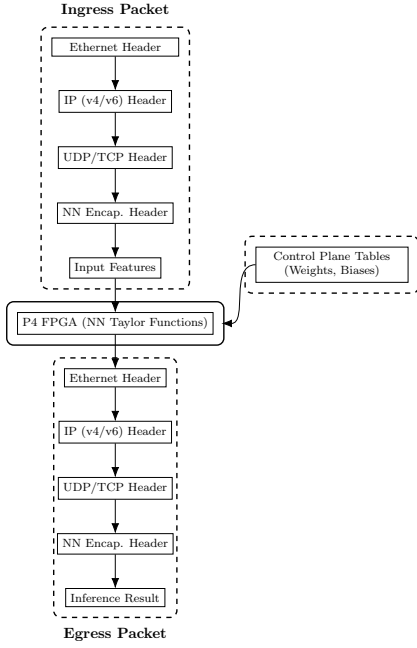


Figure 2: Packet processing pipeline for neural network inference using P4-programmable FPGAs. The control plane loads model parameters; packets carry input features, are processed via Taylor-approximated functions, and exit with embedded inference results.

Table 1: Neural Network Encapsulation Header

Field	Size (bits)	Description
Model ID	16	Model identifier
Feature Cnt	8	# input features
Output Cnt	8	# output features
Scale	16	Fixed-point scaling factor
Flags	8	Control flags (e.g., padding)
Feature 1	32	1st input feature value
Feature 2	32	2nd input feature value
⋮	⋮	⋮
Feature N	32	Nth input feature value

- (1) **Floating-Point to Fixed-Point Conversion:**
 - Floating-point x to fixed-point x_q via scaling 2^s . Equations and parameters are in Table 2.
- (2) **Taylor Series Approximations:**
 - Non-linear functions (e.g., sigmoid) approximated using Taylor series. Table 3 compares orders, and Table 4 lists scaled constants.
- (3) **Control Plane Integration:**
 - Fixed-point parameters and Taylor coefficients stored in FPGA control plane tables (Table 4) for efficient retrieval.

3.1 Fixed-Point Representation

Table 2 summarizes the procedures for encoding a floating-point weight into fixed-point and decoding it back. Here, a weight w is encoded as follows in the table.

Table 2: Fixed-Point Encoding and Decoding

Operation	Equation	Parameters
Encoding	$w_q = \text{round}(w \times 2^s) + b$	w : float weight, s : scale, b : offset
Decoding	$w \approx \frac{w_q - b}{2^s}$	w_q : fixed-point, s : scale, b : offset

3.2 Sigmoid Approximation

The sigmoid activation function is approximated using Taylor series expansions to facilitate its implementation in fixed-point arithmetic. Table 3 presents the first-order (linear), third-order (cubic), and fifth-order (quintic) approximations, where $R_n(x)$ represents the residual error term at each order.

Table 3: Taylor Series Approximations for Sigmoid with Error Terms

Order	Approximation	Use Case
1st (Linear)	$\sigma(x) \approx 0.5 + \frac{x}{4} + R_1(x)$	Low-precision for small $ x $; residual error $R_1(x)$.
3rd (Cubic)	$\sigma(x) \approx 0.5 + \frac{x}{4} - \frac{x^3}{48} + R_3(x)$	Higher precision over broader range.
5th (Quintic)	$\sigma(x) \approx 0.5 + \frac{x}{4} - \frac{x^3}{48} + \frac{x^5}{1440} + R_5(x)$	Very high precision over wider inputs.

Table 4 lists the scaled constants used in the fixed-point arithmetic when a typical scaling factor of $s = 16$ is employed.

Table 4: Scaled Constants for Fixed-Point Arithmetic with Higher Precision

Constant	Float Value	Fixed-Point ($s = 16$)
Bias	0.5	32768
Linear Term ($\frac{1}{4}$)	0.25	16384
Cubic Term ($-\frac{1}{48}$)	-0.0208333	-1365
Quintic Term ($\frac{1}{1440}$)	0.0006944	45

3.3 ReLU and Piecewise Linear Approximations

Activation functions such as ReLU and its variants (Leaky ReLU, Parametric ReLU) play an essential role in deep learning. The ReLU function is defined as:

$$\text{ReLU}(x) = \max(0, x),$$

and can be implemented in fixed-point arithmetic using a simple conditional statement. Leaky ReLU is defined as:

$$\text{Leaky ReLU}(x) = \begin{cases} x, & \text{if } x > 0 \\ ax, & \text{otherwise,} \end{cases}$$

where α is a small constant. Parametric ReLU extends this concept by making α a learnable parameter. In our system, piecewise linear approximations are used to balance computational efficiency with the precision required over a wide range of input values.

3.4 Loss Functions and Their Taylor Series Approximations

Loss functions guide the training process by quantifying the error between predicted and true values. Common functions include the Mean Squared Error (MSE) and Cross-Entropy losses. Their Taylor series approximations allow for implementation in fixed-point arithmetic by replacing logarithmic operations with polynomial approximations. Table 5 summarizes these approximations.

Table 5: Taylor Series Approximations of Common Loss Functions

Loss Function	Mathematical Definition	Taylor Series Approximation (around 0)
Mean Squared Error (MSE)	$L(y, \hat{y}) = (y - \hat{y})^2$	$(y - \hat{y})^2$
Binary Cross-Entropy	$L(y, \hat{y}) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$	$-y \left(\hat{y} - \frac{\hat{y}^2}{2} + \frac{\hat{y}^3}{3} \right) - (1 - y) \left(1 - \hat{y} + \frac{\hat{y}^2}{2} - \frac{\hat{y}^3}{3} \right)$
Categorical Cross-Entropy	$L(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i)$	$-\sum_i y_i \left(\hat{y}_i - \frac{\hat{y}_i^2}{2} + \frac{\hat{y}_i^3}{3} \right)$

4 Results and Discussion

Experiments demonstrate that in-network processing reduces inference latency to microsecond scale by eliminating PCIe round-trips. While fixed-point arithmetic and Taylor approximations introduce quantization errors, the normalized MSE remains below 0.15 for 8-bit fractional precision—a tolerable trade-off for latency-sensitive regression tasks like QoS prediction (Fig. 3). Similarly, third-order Taylor polynomials balance accuracy and overhead (Fig. 4), limiting MSE to below 0.2 while requiring only two additional P4 table lookups per approximation. These results validate that lightweight ML models can operate at line rate without compromising network throughput. Future work will extend this methodology to support sampling for CPU training feedback loops to the control plane for continuous training on inference data, further bridging the gap between programmable data planes and edge-deployable models.

5 Conclusion

FPGA SmartNICs enable practical in-network machine learning without heavy compromise in throughput or burden in compute resources. The proposed framework supports diverse applications through a combination of fractional arithmetic and careful pipeline design. Future directions will focus on examining the performance of models with varying weight lengths, scales, degrees, and bit precision through an in-depth exploration of hyperparameter configurations.

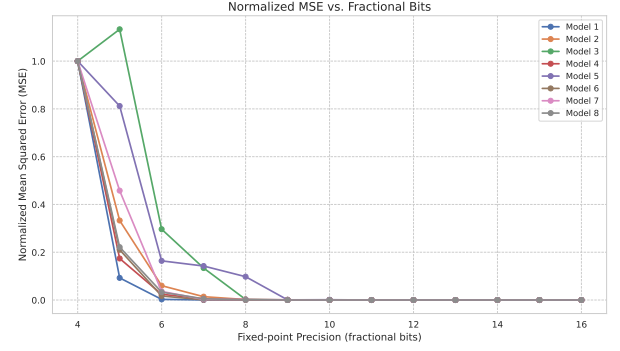


Figure 3: Normalized Mean Squared Error (MSE) versus fractional bit precision: Increasing fractional bits reduces quantization error but incurs overhead due to larger packet sizes, directly impacting network throughput. This highlights the trade-off between numerical precision and line-rate processing efficiency in P4-programmable FPGA SmartNICs.

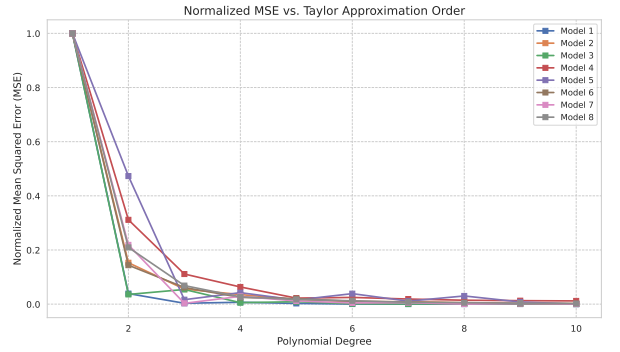


Figure 4: Normalized MSE versus Taylor polynomial order: Higher-order polynomials improve approximation accuracy for floating-point operations (e.g., sigmoid functions in neural networks) but require additional P4 table lookups and arithmetic stages, increasing pipeline latency and resource utilization.

Acknowledgments

This work was supported in part by National Science Foundation (NSF) awards CNS-1730158, ACI-1540112, ACI-1541349, OAC-1826967, OAC-2112167, CNS-2100237, and CNS-2120019. This paper was edited using LLMs hosted on the NRP [1].

References

- [1] [n. d.]. *NRP-Managed LLMs*. <https://nrp.ai/documentation/userdocs/ai/llm-managed/>
- [2] 2023. BMv2: Behavioral Model v2. <https://github.com/p4lang/behavioral-model> Online; accessed 2025-03-28.
- [3] 2023. Data Plane Development Kit (DPDK). <https://www.dpdk.org> Online; accessed 2025-03-28.
- [4] 2023. Docker. <https://www.docker.com/> Online; accessed 2025-03-28.
- [5] 2023. ESnet SmartNIC. <https://www.github.com/esnet/esnet-smartnic-hw> Online; accessed 2025-03-28.
- [6] 2023. Scapy. <https://scapy.net/> Online; accessed 2025-03-28.

- [7] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-Independent Packet Processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95. doi:10.1145/2656877.2656890
- [8] Emilio Paolini, Lorenzo De Marinis, Davide Scano, and Francesco Paolucci. 2021. Programmable Switches for In-Networking Classification. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*. IEEE.
- [9] Shivam Patel, Rigden Atsatsang, Kenneth M. Tichauer, Michael H. L. S. Wang, James B. Kowalkowski, and Nik Sultana. 2022. In-network fractional calculations using P4 for scientific computing workloads. In *Proceedings of the 5th International Workshop on P4 in Europe, EuroP4 2022, Rome, Italy, 9 December 2022*, Marco Chiesa and Shir Landau Feibish (Eds.). ACM, 33–38. doi:10.1145/3565475.3569083
- [10] Antonio Sapio, Ibrahim Abdelaziz, Abdulla Aldilajan, Marco Canini, and Panos Kalnis. 2023. Janus: An Experimental Reconfigurable SmartNIC with P4 Programmability and SDN Isolation. In *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. ACM.
- [11] Larry Smarr, Camille Crittenden, Thomas DeFanti, John Graham, Dmitry Mishin, Richard Moore, Philip Papadopoulos, and Frank Würthwein. 2018. The Pacific Research Platform: Making High-Speed Networking a Reality for the Scientist. In *Proceedings of the Practice and Experience on Advanced Research Computing: Seamless Creativity* (New York, NY, USA, 2018-07-22) (PEARC '18). Association for Computing Machinery, 1–8. doi:10.1145/3219104.3219108