This is the author's accepted version of the article. The final version published by IEEE is A.T.-J. Akem, B. Bütün, M. Gucciardo, and M. Fiore, "Showcasing In-Switch Machine Learning Inference," 2023 IEEE Conference on Network Softwarization (IEEE NetSoft 2023), Madrid, Spain, June 2023, doi: TBD.

Showcasing In-Switch Machine Learning Inference

Aristide Tanyi-Jong Akem*†, Beyza Bütün*†, Michele Gucciardo*, Marco Fiore*
*IMDEA Networks Institute, Spain, †Universidad Carlos III de Madrid, Spain
{aristide.akem, beyza.butun, michele.gucciardo, marco.fiore}@imdea.org

***Abstract*—Recent endeavours have enabled the integration of trained machine learning models like Random Forests in resource-constrained programmable switches for line rate inference. In this work, we first show how packet-level information can be used to classify individual packets in production-level hardware with very low latency. We then demonstrate how the newly proposed `Flowrest` framework improves classification performance relative to the packet-level approach by exploiting flow-level statistics to instead classify traffic flows entirely within the switch without considerably increasing latency. We conduct experiments using measurement data in a real-world testbed with an Intel Tofino switch and shed light on how `Flowrest` achieves an F1-score of 99% in a service classification use case, outperforming its packet-level counterpart by 8%.**

I. Context and motivation

In modern networking scenarios, machine learning (ML) models are increasingly playing a crucial role in enabling automation, particularly in areas like network security, traffic classification, routing optimization, and quality of service prediction. In the classic software-defined networking (SDN) paradigm, ML models are trained and executed in the control plane. However, since these models cannot operate at line rate, they fail to meet the low latency requirements typical of intelligent next-generation network applications such as industrial automation and remote surgery.

The current availability of commercial programmable data planes, such as Intel Tofino ASICs [1], and domain-specific languages like P4 [2], has opened up new prospects for achieving low-latency and high-throughput inference in networks. Various proposals have suggested encoding trained ML models directly into data plane components, including switches or Smart Network Interface Cards (SmartNICs).

By deploying ML models directly in the data plane, higher throughput and lower latency inference can be achieved. However, embedding ML models into such devices is a daunting task, notably in the case of switches with strict constraints in terms of memory, support for mathematical operations, and the number of allowed per-packet operations.

Recent works [3]–[7] have shown that trained Decision Tree (DT) and Random Forest (RF) models are more adaptable to in-switch inference owing to their logical structure and the simplicity of the mathematical operations involved. However, on account of the constraints above, current solutions fall short in some ways. Firstly, they have limited scalability when it comes to handling complex ML models and challenging classification tasks. Secondly, some of these solutions are only tested in emulation environments and cannot be deployed on real hardware, and lastly, they do not support flow-level inference which is better adapted for difficult tasks.

II. In-switch inference solution

To address the above shortcomings, during the design phase of RF models offline, we consider the limitations of the programmable switches and customize the features and hyperparameters of the RF models accordingly. The models are then translated into match and action (M/A) table entries for in-switch implementation. Inference can be conducted either on individual packets or on traffic flows in the switch. In packet-level inference [3]–[5], packets arriving at the switch are parsed and header fields such as packet length, transport protocol ports, and flags are extracted and used as features for ML inference. Packet-level solutions rely solely on stateless packet-level features and do not utilize stateful flow-level features, such as inter-arrival times and flag counts, which are crucial for effective inference in challenging use cases. This places a limit on their performance and on the complexity of use cases they can effectively handle.

To overcome the drawbacks of prior packet-level and emulated flow-level solutions, `Flowrest` [8] proposes a first practical framework for running RF models at flow level in real-world programmable switches. It facilitates the embedding of large RFs into production-grade hardware for inference tasks on unique traffic flows and at line rate, with high throughput and low latency. It is implemented as open-source data plane software using the P4 language. It is designed for the Protocol Independent Switch Architecture (PISA) and is thoroughly described in [8]. It enables a harmonious integration of in-switch inference with the legacy forwarding functions of the switch. `Flowrest` effectively uses flow-level statistics that are calculated and stored in registers in the switch as features for difficult inference tasks.

Upon packet arrival at the switch, the header information is parsed and extracted. If the packet is already classified or does not have to be, it is forwarded normally. Otherwise, the packet undergoes flow management which involves a flow management table and occurs in three phases: (i) *flow tracking* that involves CRC32 (flow identifier) and CRC16 (register index) hashes of a 5-tuple of header information (source and destination IP addresses, source and destination ports, and protocol) to identify flows and retrieve their associated data; (ii) *flow-level feature handling* that involves the use of Tofino's `RegisterAction` extern to read and update per-flow features; and (iii) *early forwarding* in which pre-

classified flows are forwarded without inference based on the classification result stored in the switch registers.

Upon completion of the flow management phase, packets are directed to the inference module, where `Flowrest` utilizes a mapping approach inspired by Planter [5] to encode RF models into the switch pipeline. This approach leverages range matches to compare computed features with feature tables and compose unique codewords that identify a specific leaf on a tree. The generated codewords are then compared with ternary matches against codeword tables, each corresponding to a tree. A voting table is employed to produce the final classification as the majority vote of the different tree outputs.

III. Demonstration

The objective of the demonstration is twofold. The first one is to exhibit how `Flowrest` performs better than a benchmark solution that relies on packet-level inference in a use case involving service categorization. The second one is to show that the entire inference task is performed in-switch without considerably increasing packet processing latency.

A. Service classification use case

To show that both packet-level and flow-level inference can be carried out in the switch, with our proposed `Flowrest` model outperforming the packet-level model serving as a benchmark that is representative of existing works, we build a use case on the public UNIBS-2009 dataset. It comprises real-world traffic from peer-to-peer applications (BitTorrent, Edonkey), mail (POP3, IMAP4, SMTP), web traffic (HTTP/HTTPS), and other protocols (FTP, SSH). The use case is a service classification task that aims at distinguishing the 8 traffic categories.

We train the packet-level model by using stateless packet-level features like the TCP/UDP source and destination ports, packet length, and TCP flags (*e.g.*, ACK, SYN, PUSH, *etc.*). In `Flowrest`, stateful flow-level features are used for model training. These are statistics computed from the packet length (total, minimum, maximum and mean), inter-arrival time (minimum, maximum and mean), flow duration, and counters of the TCP flags above. A model and feature selection phase that takes into account the constraints of programmable switches is then performed and the selected features and model hyperparameters are used to train the final models. The packet-level benchmark is a RF with a maximum depth of 9 and 2 trees and the `Flowrest` model is a RF with a maximum depth of 8 and 3 trees.

The trained models are then encoded into a P4 program with M/A table entries that represent the model. We implement the models in a real-world testbed with one Edgecore Wedge100BF-QS switch with an Intel Tofino BFN-T10-032Q chipset and 32 100-Gbps QSFP28 ports, and two off-the-shelf servers with Intel 8-core Xeon processors at 2GHz and 48GB of RAM. The hardware is interconnected with QSFP28 interfaces, resulting in a full 100-Gbps platform, as shown at the bottom of Figure 2 and pictured in Figure 1. We evaluate the classification performance of `Flowrest`

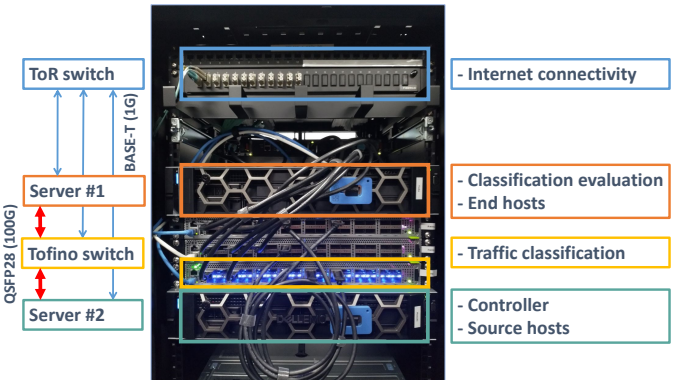

Fig. 1: Picture of the demonstration testbed, annotated with the hardware components (left) and their function (right).

TABLE I: Scores of the 3 considered metrics, for the per-packet benchmark and `Flowrest`, with absolute and relative gains of `Flowrest`.

Metric	Per-Packet	Flowrest		
		Value	Gain	
			Absolute	Relative
Precision	91.718%	99.577%	7.859%	8.569%
Recall	92.387%	99.505%	7.118%	7.705%
F1-score	91.978%	99.530%	7.522%	8.211%

and the benchmark using 3 standard metrics: precision, recall and F1-score. The results summarized in Table I reveal how `Flowrest` improves the classification accuracy relative to the benchmark by 8%.

B. Demonstration setup

The demo consists of five logical components, which include (i) packet source, (ii) programmable switch, (iii) traffic sink, (iv) statistics dashboard, and (v) a controller that interacts with the switch, traffic sink, and the statistics dashboard. These components are mapped onto the testbed hardware, as illustrated in Figure 2 and detailed below.

Traffic source. We implement the traffic source into a dedicated server. This component generates and sends 40 Gbps background traffic to the switch using the MoonGen packet

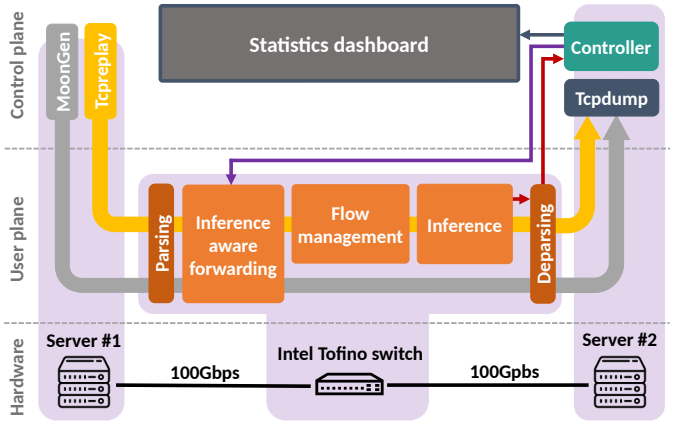

Fig. 2: Demonstration workflow, mapping the logical components of control and user planes into the testbed hardware.

Fig. 3: Outline of the demonstration dashboard, displaying the performance of different models of ML-based inference in programmable switches.

generator. The packets that are targeted by the inference, highlighted in yellow in Figure 2, are simultaneously injected into the switch by replaying historical pcap traces of the UNIBS-2009 dataset using Tcpreplay.

Programmable switch. We use an Intel Tofino switch to forward traffic from the source to the sink while performing inference on target packets or flows. After parsing, the traffic is categorized as generic or target traffic in the inference-aware forwarding block, as shown in Figure 2. While generic traffic is forwarded normally, the target packets are either classified directly in the case of the packet-level solution or first identified as belonging to specific flows in the case of `Flowrest`, and then classified, and finally forwarded according to their class [8]. After each classification, a digest is sent to the controller with the class of the flow. This enables the controller to generate a table entry that is injected into the inference-aware forwarding block to ensure that subsequent packets of that flow are no longer classified but forwarded directly.

Traffic sink. The traffic forwarded by the switch is collected by a Tcpdump instance running on a second server. Once all the target packets have been received, we can analyze the resulting pcap trace in order to find the time of arrival of the packets and the classification result stored in a header field. These enable us to analyze the packet processing latencies and the classification performances of the models.

Controller. We run a controller instance in the second server of the testbed. It is a very important element since it plays multiple roles. First of all, it configures the switch at each iteration by injecting the P4 code of the relevant evaluation to perform via the Barefoot Runtime Interface of the switch. Second, it analyzes the traces collected by the traffic sink to calculate the end-to-end packet processing latency. Third, it calculates the inference performance in terms of precision, recall, and F1-score by comparing the classification results with the ground truth. Finally, the controller interacts with the statistics dashboard and feeds data to it for display.

Statistics dashboard. We display the performance of `Flowrest` in a dashboard connected to the Controller. The dashboard is organized into two main areas, as shown in Figure 2. In the first one, we show live results while each evaluation is running. In the second one, we summarize and compare the results of the evaluations. At the top, we have three buttons which enable us to select which experiment we want to run at any time and a reset button to reinitialize the dashboard.

C. Experiments

To study how the classification accuracy and packet processing latency change over different scenarios, we perform three different runs of experiments for evaluation. The first one aims at estimating the end-to-end latency when no inference is performed in the switch, with only normal forwarding running. In the second and third experiments, we estimate the end-to-end latency and the classification performance of `Flowrest` and the per-packet benchmark solution respectively, in the service classification task.

The evaluations are performed by running the three experiments using the control buttons on the dashboard to select the experiment under consideration. Throughout each experiment, we compute and display the pertinent metrics in real time on the dashboard. At the end of each experiment, the results are kept on the dashboard to allow for comparison between the other evaluations.

From the controller, the latency involved with each of the three experiments is computed and displayed on the dashboard. The results reveal that while normal packet forwarding entails an average packet processing latency of 284 ns, per-packet inference brings just an additional 34 ns latency, taking the end-to-end latency up to 318 ns. In the case of `Flowrest`, the latency becomes 391 ns meaning that it can improve classification performance and enable for the first time, flow-level ML inference in production-grade hardware while incurring just an additional 73 ns of latency. Overall, this demonstrates that in-switch inference solutions like `Flowrest` can bring artificial intelligence to assist in network automation at very low latency.

ACKNOWLEDGMENTS

This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement no. 101017109 "DAEMON", the Marie Skłodowska-Curie grant agreement no. 860239 "BANYAN", and the CHIST-ERA grant no. CHIST-ERA-20-SICT-001 "ECOMOME", via grant PCI2022-133013 of Agencia Estatal de Investigación. We thank the Intel Connectivity Research Program for their support.

REFERENCES

[1] Intel, "Tofino Switch." [Online]. Available: https://tinyurl.com/ycx79k4z
[2] P. Bosshart et al., "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, 2014.
[3] Z. Xiong et al., "Do switches dream of machine learning? toward in-network classification," in *HotNets 2019*. NY, USA: ACM, 2019.
[4] A.T-J. Akem et al., "Henna: Hierarchical machine learning inference in programmable switches," in *NativeNi '22*. NY, USA: ACM, 2022.
[5] C. Zheng et al., "Planter: Seeding trees within switches," in *SIGCOMM '21*. NY, USA: ACM, 2021.
[6] C. Busse-Grawitz et al., "pForest: In-network inference with random forests," *CoRR*, vol. abs/1909.05680, 2019.
[7] H. Siddique et al., "Towards network-accelerated ML-based distributed computer vision systems," in *IEEE ICPADS*, 2021.
[8] A.T-J. Akem et al., "Flowrest: Practical flow-level inference in programmable switches with random forests," in *IEEE INFOCOM 2023*.