Received 20 May 2024; accepted 4 June 2024. Date of publication 7 June 2024; date of current version 28 June 2024.

Digital Object Identifier 10.1109/OJCOMS.2024.3411071

In-Line Any-Depth Deep Neural Networks Using P4 Switches

EMILIO PAOLINI[1,2,3] (Student Member, IEEE), LORENZO DE MARINIS[1] (Member, IEEE), DAVIDE SCANO[1], AND FRANCESCO PAOLUCCI[4]

[1]TeCIP Institute, Scuola Superiore Sant'Anna, 56124 Pisa, Italy

[2]CNR-IEIIT, 56122 Pisa, Italy

[3]Sma-RTy Italia SRL, 20061 Carugate, Italy

[4]PNTLab, Consorzio Nazionale Interuniversitario per le Telecomunicazioni, 56124 Pisa, Italy

CORRESPONDING AUTHOR: E. PAOLINI (e-mail: emilio.paolini@santannapisa.it)

This work was supported in part by the European Commission Horizon Europe SNS JU NATWORK Project under Grant 101139285 (NATWORK), and in part by the European Union through the Italian National Recovery and Resilience Plan (NRRP) of the NextGenerationEU partnership on "Telecommunications of the Future" (Program "RESTART") under Grant PE00000001.

ABSTRACT In-network function offloading using programmable data plane languages like P4 offers computational resource savings and efficient operations at the network edge. However, the deployment of ML functions in P4 switches exploiting no additional hardware remains a challenge. In this paper, we tackle the challenges in deploying Deep Neural Networks (DNNs) within programmable network devices, introducing a novel distillation based on Look-Up Tables (LUTs). The proposed method maps quantized DNNs into a cascaded arrangement of LUTs without loss in accuracy and independently of the quantized network depth. The developed approach is demonstrated in two network function use cases: a cyber security use case focused on mitigating Distributed Denial of Service attacks and a malicious activity classification task in IoT Networks. Experimental results show a trade-off between model's accuracy, expressed in terms of F1-Score and the computational demands, influenced by bit size and number of LUTs. In addition, the latency for deploying these models ranged from 54ns to 112ns, showing the method's practical applicability in network functions.

INDEX TERMS Cyber security, deep neural networks, look-up tables, P4, traffic classification.

I. INTRODUCTION

IN THE foreseeable future, the deployment of in-network Machine Learning (ML) is expected to revolutionize network devices, enabling them to process data such as packets, flows, and aggregate traffic at wirespeed, without intermediate processing chains. This advancement will be supported by built-in ML-enabled components [1], [2]. The trend of in-network function offloading is gaining popularity due to the maturity and flexibility of programmable data plane languages such as P4. This approach holds the potential to deliver programmable and versatile networking operations at the data plane, leading to computational resource savings for edge applications with limited energy consumption, as the network infrastructure devices can be reused [3].

Until now, ML-related network accelerations have often been relegated to external and dedicated backends, such as co-located Field-Programmable Gate Arrays (FPGAs) or servers equipped with Graphics Processing Units (GPUs). However, this approach has resulted in suboptimal solutions, introducing additional latency and with low energy efficiency. A more promising solution lies in extracting ML features at the data plane and performing cloud-based inference [4].

Recent endeavors to introduce ML algorithms directly at the data plane of programmable devices have explored reconfigurable pipelines capable of reproducing or approximating ML algorithms through flow tables and stateful operations [5]. For instance, recent works have focused on implementing binary neural networks at the Network

Interface Card (NIC) [6] and specific ML algorithms like Support Vector Machines (SVMs) [7], Decision Trees (DTs) [8], and Random Forests (RFs) [9].

On the other hand, deploying Deep Neural Networks (DNNs) inside programmable network devices presents significant hardware constraints. The DNN workflow requires fast Arithmetic Logic Units (ALU) for efficient neuron computation and high-speed stateful memory updates. However, programmable hardware pipelines are designed to accommodate numerous match-action flow tables, which offer limited arithmetic variable manipulation and low stateful memory update rates. Consequently, only P4 DNN implementations for software switches or smart NICs have been presented so far [10], [11], [12], with significantly limited performance capabilities in terms of introduced latency and resource management.

In this paper, we propose a Look-Up Table (LUT) distillation method that maps the P4 DNN pipeline, which inherently demands ALU and substantial stateful capabilities, into a cascaded architecture of simple flow tables. The method, inspired by knowledge distillation techniques [13] is based on the mapping of all possible outputs of the quantized DNN. These flow tables match input features with DNN outcomes, effectively creating a simple aggregated input/output LUT. Importantly, our distillation technique is lossless, ensuring no information is lost during the mapping from the quantized DNN to LUT. Additionally, the method is agnostic to DNN complexity and is able to reproduce the input-output behavior of a DNN with an arbitrary number of hidden layers, as it requires a fixed amount of resources for the distilled LUT, provided that inputs and outputs adhere to hardware memory constraints. To demonstrate the practical application of our approach, we apply it to a number of different network functions: a cyber security use case focused on mitigating Distributed Denial of Service (DDoS) attacks, and an IoT anomaly activity classification. Experimental results performed on the mapping from the DNN model to the proposed distillation technique, combined with the performance parameters indicators extracted from the deployment on a commercially available programmable switch showcase the feasibility and advantages of our distillation technique within a P4 pipeline, specifically its efficient utilization of stateless features compared to other existing offloading deployments in the data plane, such as entropy-based anomaly detection strategies [14], which typically necessitate large stateful memories. Ultimately, our proposed LUT distillation technique represents a critical first demonstration of fully offloaded DNN-based functions within network programmable devices.

II. BACKGROUND

A. MACHINE LEARNING AND AI FOR NETWORK FUNCTIONS

ML techniques have been applied to various problem domains, from healthcare to robotics [15]. As the amount of data grows and the complexity of network architectures rise, ML algorithms have been recently adopted in networking problems. Indeed, issues regarding human network administrators in overseeing every aspect of networks are becoming evident: a challenge that is expected to intensify with the continuous expansion of network complexity [16].

Hence, with next generation networks expected to be self-driven [16], ML algorithms represent an incredible opportunity to overcome the management and optimization problem. For instance, authors in [17], introduce a method to combat DDoS attacks, which are a significant threat to network security. Utilizing software-defined networking (SDN), the approach integrates a one-dimensional convolutional neural network (1D-CNN) with a Ryu controller and Mininet for efficient detection and mitigation of DDoS attacks. The 1D-CNN model performance significantly surpasses other machine learning models.

Besides cybersecurity, ML techniques can be applied to many other aspects of networking. In the context of congestion control, in [18], the authors present a novel approach that integrates ML and heuristic strategies to enhance network performance in terms of fairness, latency, and packet drop rate. The proposed strategy not only lowers end-to-end latency, but also significantly diminishes packet drop rates and enhances fairness.

B. PROGRAMMABLE DATA PLANE (PDP) AND P4

The SDN paradigm has gained wide consensus, introducing the concept of network programmability. In its early stages, the adoption of network programmability was primarily facilitated by the OpenFlow protocol. To delve deeper, OpenFlow switches possess the ability to process and manipulate a predefined set of headers. Furthermore, they enable the creation of tables with a fixed set of actions designed for traffic management [19]. Subsequently, the potential for complete access to the network operating system (NOS) and data plane pipelines enables the attainment of full network programmability.

This advancement laid the foundation for the emergence of the P4 (Programming Protocol-Independent Packet Processors) language. As described in [20], P4 is a programming language for creating custom data plane pipelines. P4 enables the definition and manipulation of customized packet headers by creating dedicated packet parsers. It supports the utilization of counters, registers, facilitating the creation of stateful functions. Furthermore, thanks to P4 high-level language properties of logic and arithmetic manipulation, it becomes feasible to perform various simple arithmetic operations, such as addition, subtraction, multiplication, and division. Additionally, P4 enhances network visibility, particularly through methods like In-band Network Telemetry (INT). P4 finds support across various target platforms, including bare metal or software switches, smart-NICs, and NetFPGAs. As presented in [21], P4 finds application in a diverse range of use cases, encompassing network telemetry, network features, middlebox functions, accelerated computation, cybersecurity, etc.

In [22], a novel pipeline is implemented to deliver stateful traffic engineering and cybersecurity functionalities on an edge node tailored for a multi-layer IP over optical network. Additionally, the concept of augmented firewalling capabilities is envisaged to counteract DDoS cyberattacks. Further P4 use cases within multi-layer networks are detailed in [11], encompassing end-to-end optical performance indicator telemetry exchanged among packet-optical nodes, as well as P4-defined neural networks designed to enhance online cybersecurity measures. P4 can be also used to implement active queue management (AQM) algorithms, in [23] the CoDel (Controlled Delay) algorithm is successfully implemented and showcased on a P4 hardware switch. Packets processed through the ingress pipeline are placed in a queue within the traffic manager block. Subsequently, in the egress pipeline, the CoDel algorithm is employed to regulate queueing delay by selectively discarding packets.

C. EXISTING IN-NETWORK ML FUNCTIONS

While there is a current gap in strategies to seamlessly incorporate DNNs at wire-speed within programmable switches, other machine learning techniques have been applied in online functions within network devices. Some recent research works in the literature have explored the potential of in-network applications that leverage programmable data planes to address this challenge. The main considered machine learning algorithms embedded in the programmable data plane pipelines are the DT, the RF, and the NN [24]. Concerning the latter, however, modified ASICs are mainly used [25].

In [26], the authors propose a machine learning-based method, based on DTs, for predicting heavy flows directly within the switch. Given the limited memory and computation capabilities, the solution employs a packet processing pipeline that utilizes pre-trained DT models for in-network prediction, evaluated in BMv2 and in the Tofino ASIC.

Extending this approach to utilize a broader ensemble of machine learning techniques, [27] introduces an approach that incorporates the RF algorithm into a programmable switch. Such integration makes the RF both configurable and re-configurable in real-time, demonstrating the estimation of flow-level stateful features, such as round-trip time and bitrate for each network flow.

Another recent work [28] proposes a novel mapping method of ML classification models to off-the-shelf switches. However, the discussed approach only supports tree-based classification models and classical model, such as SVM and K-Means. Additionally, the proposed method adopts a hybrid approach for ensemble models, running a small model on a switch and a large model on the backend. Our approach, on the other hand, enables the deployment of DNN with no additional hardware.

A relevant work in the field in-network ML proposes and deploys a comprehensive tool called Planter [5]. Planter automatically maps different types of ML models, including related datasets and training steps for the specific network function, into a P4 code deployable on different software/hardware backends, utilizing the state of the art of the literature. Concerning DNN, Planter supports the P4 deployment of binary neural networks (BNNs).

Still in the context of binarized NNs, in [29] the authors propose show how BNNs can be implemented as switch functions at the network edge to classify packets at the line speed of the switches. However, it is important to note that this method is limited to the use of BNNs. In [30] an NN layer splitting framework is discussed, allowing to overcome small memory constraints when implementing NN models in one switch. However, the solution is still limited to the context of BNNs and involves multiple switches, e.g., requiring more hardware and potentially more sophisticated network management tool.

Finally, a new hardware architecture for per-packet ML in the data plane of network devices is introduced in [31], namely Taurus. The proposed solution leverages custom hardware based on a flexible, parallel-patterns (MapReduce) abstraction to enable per-packet MapReduce operations, which includes inference tasks executed at line rate. Taurus adds new hardware components to existing network devices, specifically implementing SIMD (Single Instruction, Multiple Data) parallelism to support its per-packet ML operations, e.g., multiplications, resulting in additional hardware.

III. NETWORK-INLINE PROGRAMMABLE DNN

A. ARCHITECTURAL STRATEGIES TO INCLUDE DNNS IN NETWORK FUNCTIONS

While solutions have been proposed to enable ML-driven functions within the networks, resorting to DNN-based algorithms is still challenging. This is because neural networks require distributed and non-linear computations, while algorithms such as decision trees or support vector machines do not and are more easily implemented within the available network functionalities. We have divided the solutions to include DNNs in network functions in four architectural approaches, which are depicted in figure 1.

In architecture a), the switch or NIC, receive and match selected packets/flows. These packets/flows are directed towards an external device, such as an FPGA, where the DNN is deployed. The result produced after neural network inference is transmitted back to the switch/NIC through internal interfaces (e.g., dedicated or SDN control plane interfaces). In this design, packets my be stored within dedicated buffers while awaiting prediction or classification, or they can be immediately directed towards the external device. In either scenario, the process of chaining devices introduces several delays and lacks power efficiency as it demands the operation of two devices, ideally at wire speed.

In architecture b) the switch/NIC is enhanced to manage both packet and metadata processing with the aim of extracting DNN features during runtime. This relieves part of

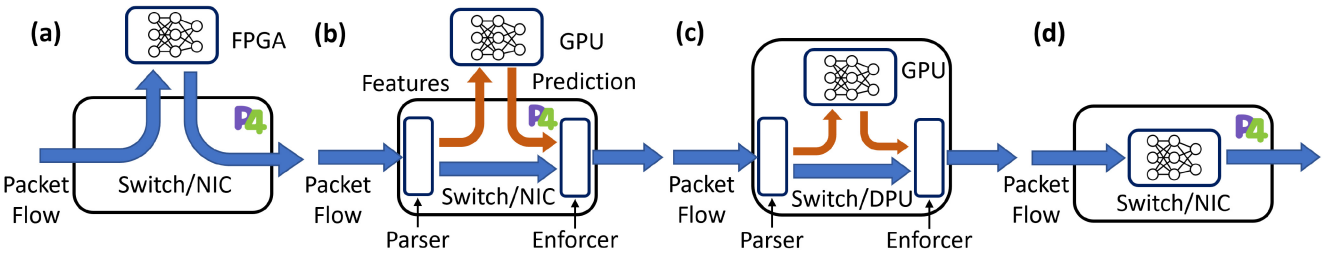


FIGURE 1. Strategies for the deployment of ML models at the Switch/NIC side. (a) the packet flow is directly sent to a co-located FPGA, which parses the data and runs a DNN. (b) Hybrid strategy where packet features are extracted within the switch/NIC and sent to the DNN run on a GPU. (c) Switch embedding a GPU, actually non programmable within the P4 environment (d) The ML model is deployed in the pipeline within the switch/NIC.

the computational burden from the external device, as feature extraction stands out as a time and resource consuming process, particularly when conducted through software-based methods. This method is faster and more effective at the data plane level where programmable parsers and a limited amount of memory registers and counters are used [4]. The DNN processing is still delegated to a specialized and external device, such as a GPU, while the switch/NIC handles packet enforcement and deparser tasks. Once more, the packets must be buffered as they await the outcome of the DNN processing.

The recent efforts in embedding DNN processing in network functionalities have resulted in architecture c), where a GPU is co-located within the hardware switch. In fact, novel switch and smart NIC platforms are encompassing the embedding of a GPU dedicated to ML algorithms in support of programmable ASIC devoted to network functions. The main advantage is that the use of internal GPU drastically speeds up the processing and detection workflow, since the communication between the two modules is embedded and optimized, relying on internal bus architecture. On the other hand, relying on a GPU results in low energy efficiency.

Finally, architecture d) illustrates the proposed solution, where the entire pipeline, including the DNN function, is offloaded into the programmable switch/NIC. This approach eliminates the necessity for any additional devices. The design incorporates parsers, feature extraction, and the DNN macro-function operating at wire speed exploiting only match-actions in LUTs. This configuration minimizes the need for packet buffering and reduces latency, as the entire process is executed using just one device. Moreover, the influence of a programmable chipset in comparison to a fixed-function ASIC is negligible [32]. The modular P4 design enables the acceleration of multiple network functions within the same pipeline/device. As a result, supplementary functions such as forwarding/steering and load balancing can coexist seamlessly with the DNN-based in-network function.

Additionally, the proposed architecture integrates the ML model directly into the pipeline, allowing for a more energy-efficient operation, as the network hardware can be optimized to execute these operations with lower power consumption compared to a GPU setup.

B. CURRENT LIMITATIONS IN PROGRAMMABLE SWITCHES

Nowadays the limitations in programmable switches are: 1) parallelizing the operation is either prevented or significantly restricted 2) hardware backends do not encompass all the capabilities offered by the P4 language. This limitation leads to latency performance that falls below the optimal level. For instance, when examining the P4 DNN implementation in software switches that maximizes the potential of the P4 language, the intra-switch latency is around one order of magnitude higher for standard pipelines (e.g., forwarding and steering) [11]. This limitation arises from the hardware vendors' chipset design strategies, which prioritize meeting network requirements, such as providing fast memories for lookup tables, over adding computational resources like ALUs. While the last architecture discussed in the previous section is the only approach that enables the full realization of in-network ML functionalities, deploying neural networks within the dataplane pipeline using conventional methods remains unfeasible. DNNs are distributed architectures composed of interconnected primitives (artificial neurons), which operate by iteratively performing elementary mathematical tasks primarily involving multiply-accumulate operations and nonlinear functions. As a positive feature, DNNs exhibit a specific resilience to handle noisy inputs and low-precision representation formats effectively when subjected to appropriate training methods [33]. Therefore, it's not necessary to use high-resolution floating-point numbers, and the associated computational overhead can be circumvented by employing integers. Nonetheless, the programmable ASIC pipelines in commercial P4 switches do not currently offer support for floating-point operations nor integer arithmetic [11]. The necessity for wirespeed processing permits only specific operations, like flow table matching, where variables are maintained in an integer format. In current P4 switches the absence of multiply-accumulate operations prevents the deployment of DNNs even if the parameters are in an integer format. In [34] is presented an implementation example of DNN, consisting of two primary P4 stages: feature extraction and the DNN function. The P4 language can be used to implement DNNs by employing integer-based updates for neuron computations and utilizing tree-based structures. However, due to the

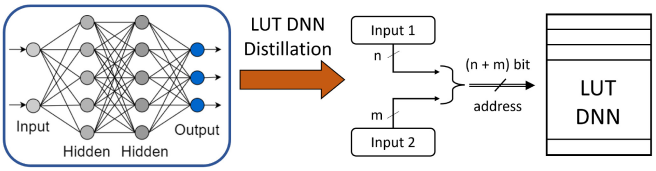


FIGURE 2. Deep neural network to lookup table distillation method. Integer-encoded inputs are treated as a compound address for the LUT, whose entries encode the corresponding outputs of the distilled DNN.

constraints imposed by limited parallelized computation resources, the resulting performance capabilities are low. The P4 codes can be successfully ported to a programmable ASIC backend, specifically benefiting the feature extraction stage. However, adapting the DNN function necessitates a comprehensive remapping design, primarily focusing on avoiding ALU-based operations.

C. CASCADED LOOK UP TABLE DISTILLATION

To deploy DNN models in hardware pipelines without arithmetic capabilities, we have conceived a strategy to distill the knowledge of a trained and integer-quantized DNN into a LUT, leveraging the constraint of integer-encoded variables as an advantage. Figure 2 represents the core idea of our distillation technique. Consider the case of a 2-input network with integer-valued parameters, whose inputs are encoded in n and m bits, respectively. To distill the network to a LUT, the inputs are paired and treated as a compound address with a bitwidth of $n + m$ bits. The table is generated by simply collecting the DNN outputs for all the possible input configurations, which are $2^{(n+m)}$. In the more general case of a network with more than two inputs, all inputs can be concatenated in a compound address in the same way. With this method, the inference of an integer-quantized DNN is reduced to a match-action on a flow table. The LUT distillation method is lossless as the procedure considers exhaustively all possible DNN outcomes, hence prediction accuracy is not reduced after model transformation.

With respect to other table-based quantization strategies [35], [36], [37], we rely on LUTs not to perform multiplications at higher speed nor to accelerate the quantization procedure, but to embed the whole DNN in it. However, the LUT distillation method has some limitations: (i) memory consumption grows exponentially with the number of bits composing the input features, (ii) input features must be encoded as integer and must avoid the floating-point type, while hidden units can use the latter, and third (iii) the number of output variables linearly affect the LUT size, i.e., if the DNN produces k outputs then k entries are added in the LUT per input feature. Concerning the first point, the keys (representing the quantized input features) for each table share a fixed amount of memory (e.g., hundreds of bits). This limitation can impact the size of the keys and the maximum number of keys that can be defined in a table. Moreover, as the space occupied by the keys increases, the memory occupancy of the table also increases.

Still, this distillation method has stark advantages, first of all being making DNNs deployable in hardware P4 switches. Second, no constraint is set to the DNN complexity nor type: this method can be applied to huge DNNs and even to other Machine-Learning algorithms, given that the requirements on inputs and outputs are met. Hence, if the constraints related to the quantization of all possible inputs and outputs are met, any ML method could theoretically be implemented. Particularly, in networking scenarios, the quantization challenges may not be as pronounced as working with more complex data types, e.g., images, allowing for greater flexibility in the implementation of ML methods. Finally, to increase the prediction accuracy [33] more complex DNN can be trained without affecting the final inference time (one match-action), with the drawback of increasing the training and distillation times. Moreover, if a use case requires periodical retraining of the neural model, this can be done on a co-processor and applied by simply updating the LUT entries.

D. DESIGN STRATEGIES

In practical cases, the need of a quantized DNN model is not a stringent limitation: quantization methods produce models with low or even negligible prediction accuracy loss [38]. On the other hand, the exponential grow in memory requirements with respect to the input features could prevent the use of this method as most DNNs make use of more than a couple of inputs. Consider a DNN with one 8-bit output and four 8-bit input features, the correspondent LUT entries would be 2^{32} corresponding to 4 Gbytes.

To circumvent this issue, and exploit the LUT distillation method, a hierarchical structure composed of simple and cascaded DNNs can be designed instead of a single DNN. The workflow of this strategy is represented in Fig. 3. Input features are paired and sent to a first batch of DNNs, whose outputs are in turn paired and sent to a second layer of DNNs. This is done recursively until the final output is computed, forming a hierarchical architecture of simple 2-input neural networks. The compound structure can be trained for the target task, and the LUT distillation method is applied to each neural model forming a corresponding structure of cascaded tables. Note that, the first step involving a single and bigger DNN can be skipped and the hierarchical structure of simpler and interconnected DNNs can be directly designed instead. This is a shift in the design strategy, so the hierarchical structure do not need to come from a bigger and single DNN. Following the initial example with four 8-bit inputs, with this technique the final model will have three tables with 2^{16} entries each, and a total memory usage of 192 kbytes, instead of 4 Gbytes. The cascaded method opens the possibility to distill DNNs in LUTs even when several inputs are present. More features will result in a deeper hierarchy of tables, with a depth of $\lceil \log_x y \rceil$, where x is the number of inputs per small DNN and y is the total number of inputs.

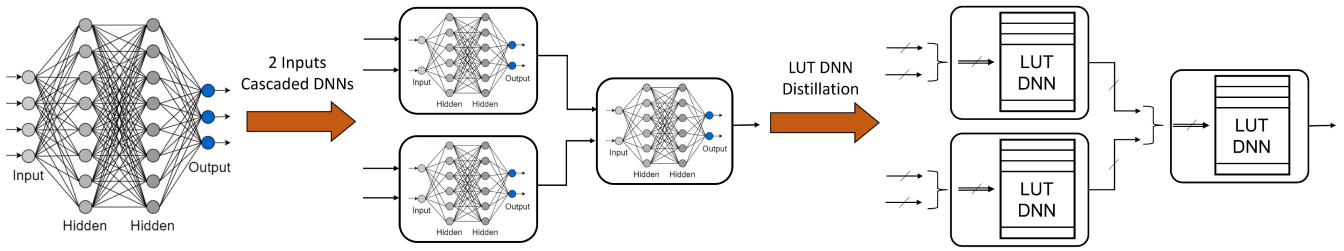


FIGURE 3. Cascaded LUT DNN design strategy. Instead of using a single and big DNN, input features are grouped in pairs and sent to separate DNNs, whose outputs are in turn paired recursively until the final output. After training the single models are distilled into small 2-input LUTs.

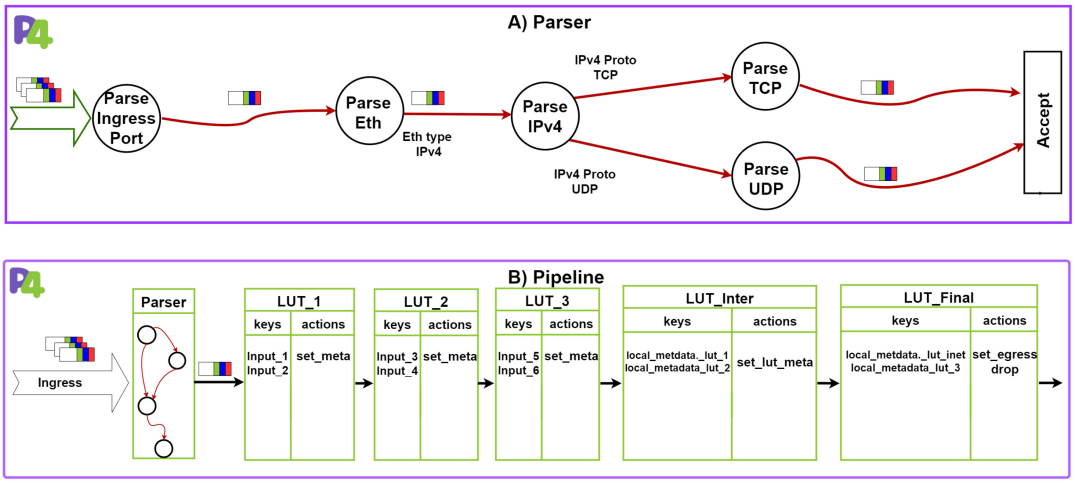


FIGURE 4. P4 Parser. The table is generated by simply collecting the DNN outputs for all the input-pair combinations.

IV. P4 IMPLEMENTATION

To prove the effectiveness of our method in a real case scenario, we have developed the LUT distillation method with the $P4_{16}$ language, targeting a physical realization within the `tofino` switch [39]. We have considered two use-cases, i.e., Intrusion Detection System and IoT anomaly classification, for in-network DNN processing, which are described in Section V.

The developed P4 Program implements a parser and a flow-table pipeline where, the former extracts the input features for the machine learning algorithm and the latter implements the LUT-distilled DNN. This same structure can be easily extended to extract features from other headers that might of interest in different ML tasks. For example, the parser can be extended to extract GTP tunneling by adding a specific check during the parsing of the UDP header, i.e., to recognize the UDP port 2152. Thus, a parser that supports multiple headers could be deployed on the same network infrastructure.

The parser, depicted in Figure 4, serves as the initial module in the pipeline. As packets progress through its stages, the input features required for the cascaded LUT DNN are extracted. The number of input features retrieved during the parsing phase and the number of tables in the pipeline are related to the number of DNNs and inputs per elementary DNN, that you want to deploy in your architecture. The features are sent to the pipeline, whose tables resemble the architecture of interest. The LUTs apply an exact match policy to the `inputs` and initialize the custom metadata (i.e., `local_metadata.*`), which encodes the output of intermediate stages to be used to the following ones. The last table in the pipeline, gives the prediction as an action, e.g., forwarding or dropping a packet.

The first stage of the parser is *Parse Eth* that extracts the Ethernet header. Then, in case of IP packets, the *Parse IPv4* stage parses the IPv4 header and fills the corresponding metadata fields with IP protocol (IP proto), IP Time To Live (TTL) (s TTL). Subsequently, the packet is sent to one of the *Parse TCP/UDP* stages where the metadata fields are filled with the source and destination TCP window advertisement (swin and dwin), source-to-destination (s bytes) and destination-to-source transaction bytes (d bytes).

A. PIPELINE DESIGN

We take as an example the case of a cascaded DNN with 6 input features, one output, and composed of 2-input DNNs. The distilled architecture is composed of 5 tables divided in three layers, as depicted in Figure 5(A). This architecture is implemented in the P4 pipeline illustrated in Fig. 5(B). After parsing, the packets are forwarded to the pipeline and processed by `LUT_1`, `LUT_2` and `LUT_3` which support the actions `set_meta`. The flow rules contained in these tables use the `inputs` as the key parameter in the match policy. When a packet matches

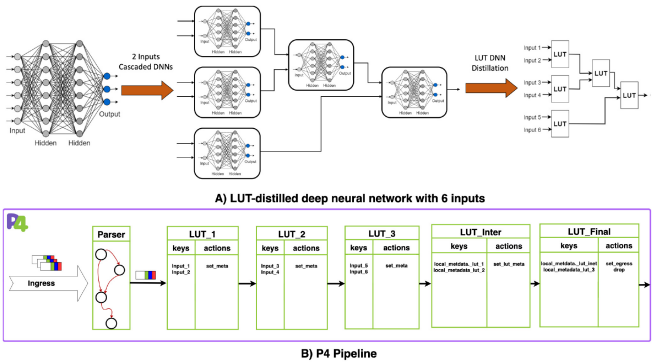


FIGURE 5. A) LUT distilled deep neural network B) P4 Pipeline.

a flow rule, the `set_meta` action initializes the associated `local_metadata_lut_*`. Then the packet passes through the table `LUT_Inter`, which supports the actions `set_lut_meta`. The flow rules in this table use the `local_metadata_lut_*` as the key parameter in the match policy and the action applied is `set_lut_meta` action initializes `local_metadata_lut_inter` associated with it. Finally, the packet reaches the table `LUT_Final`, which supports the actions `set_egress` and `drop`. The `set_egress` action is typically applied to packets that match a forwarding flow rule, assigning the output port through which the packet will be transmitted. Meanwhile, the default `drop` action is applied to packets that do not match any flow rules, resulting in the packet being dropped.

V. RESULTS AND USE CASES DEMONSTRATIONS

In this section we describe the experimental demonstrations that we run on a P4 switch APS Networks BF2556X-1T, which has 48 interfaces ranging from 1 to 100Gbits. The switch's software configuration is based on the Open Network Install Environment (ONIE). On top of ONIE, it runs Linux Ubuntu 22.04, and P4 Studio SDE 9.7 is installed. The performance of the P4 code was evaluated using the P4 Studio application on a dedicated workstation with Intel Xeon W-2223 8-core 3.60 GHz clock, 16 GB RAM.

A. DISTILLATION TECHNIQUE PERFORMANCE

The use of typical 32-bit floating point numbers in neural networks presents a significant challenge when attempting to transform these networks into LUTs. This challenge arises because the precision and range of 32-bit floating point numbers lead to an extremely large number of possible input-output combinations, making the creation of a direct LUT impractical due to memory constraints and computational inefficiency. To address this, quantization methods become essential. Quantization effectively reduces the precision of the inputs and outputs, mapping them to a smaller set of values. This reduction in precision, while potentially introducing some level of approximation error, significantly decreases the size of the required LUT, making it a feasible approach for certain applications. By quantizing the data, we can represent neural network computations in a more compact, memory-efficient manner, thus enabling the transformation of certain operations within the network into a LUT format, which can be advantageous for speeding up computations, particularly in resource-constrained environments.

Specifically, we adopted a Quantization-Aware Training strategy, in which loss due to quantization is counteracted through an ad hoc training strategy [40]. Specifically, an input quantizer q_{input} is introduced to define the way of quantizing the incoming inputs. Hence, in our approach, quantized layer computes the activation y as:

$$y = \sigma\left(f(w, q_{\text{input}}(x)) + b\right)$$

with full precision weights w and input x, layer operation f, activation function σ, and bias b. We relied on the DoReFa quantizers [41], since it provides a convenient way to flexibly define the bitwidths for inputs.

VI. DEMONSTRATIONS

In this section, we present the results related to the DDoS and traffic classification experiments. We implement 4 different scenarios, considering different number of LUTs and input features: 1 LUT with 2 features, 5 LUTs with 6 features, and 7 LUTs with 8 features, as depicted in Fig. 6. Only 4,6, and 8 bits per input feature are considered to keep the LUT size compliant with the Tofino, i.e., keeping the maximum number of equivalent address bits for each LUT ≤ 16 in all experiments. A feature selection strategy has been implemented in both use-cases, leveraging the ANOVA F-value between labels and features.

A. USE CASE 1: CYBER SECURITY DDOS MITIGATOR

Concerning the DDoS mitigator experiment, the CSE-CIC-IDS2018 dataset has been used [42]. This dataset, developed by the Canadian Institute for Cybersecurity, is a comprehensive dataset designed for the evaluation of intrusion detection systems. It encompasses a wide range of modern attack types, including DDoS, Web attacks, and infiltration of the network from simulating real-world data. Features that are not compliant with a P4 implementation have been removed from the dataset, e.g., mean and standard deviation extracted from traffic flows.

For our experiments, we considered 2 sub-use-cases: (i) a binary classification problem, with the goal of distinguishing between normal and malicious flows and (ii) a multiclass classification problem, considering 4 classes: normal traffic, DoS attacks-Hulk, DoS attacks-GoldenEye, and DDOS attack-HOIC. The NN block, corresponding to a single LUT, is composed of 4 fully-connected layers, with 256, 128, 32, ad 1 or 4 neurons for the last layer depending on the classification task respectively. ReLU has been chosen as activation function, and the architecture has been trained for 50 epochs using Adam as optimizer. Results for varying bitwidths are sketched in Fig. 7.

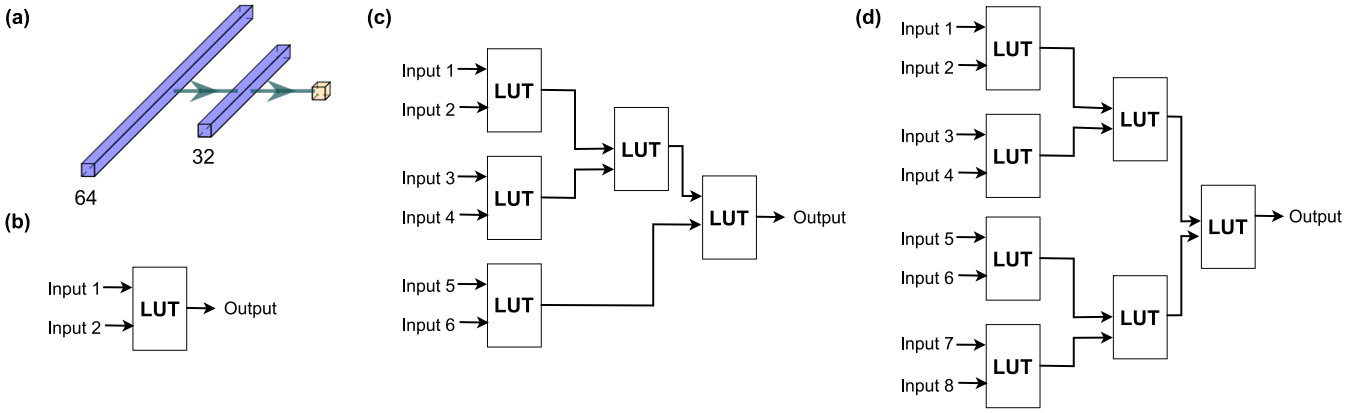
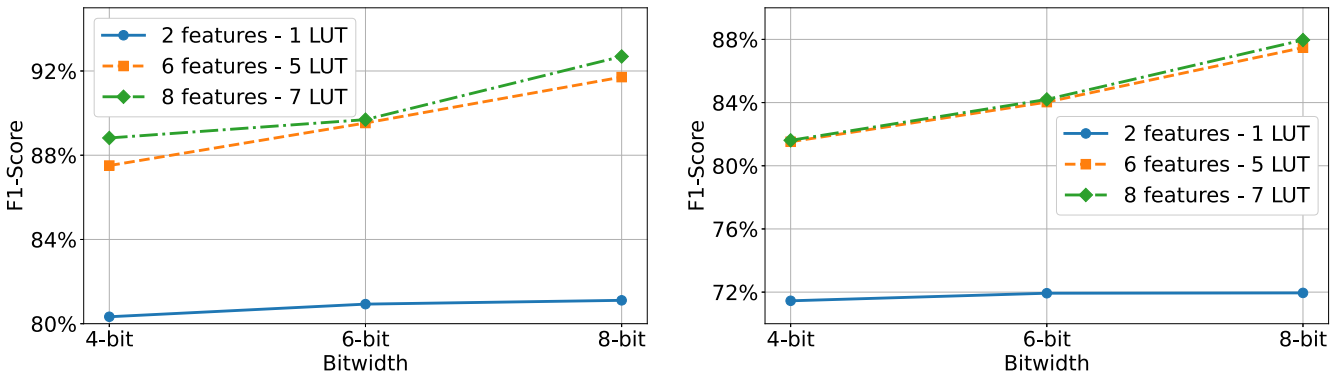


FIGURE 6. (a) The 2-input base neural network module, corresponding to a distilled LUT; (b) model architecture for stateful features; (c) model architecture for stateless features; (d) model architecture for the combined features.



(a) Binary.

(b) Multiclass.

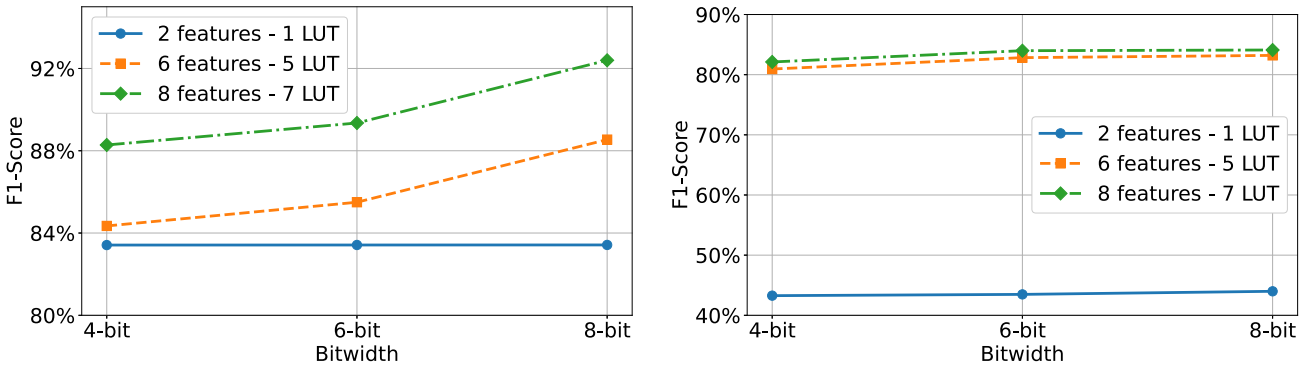
FIGURE 7. F1-scores on CSE-CIC-IDS2018 per varying bitwidths.

In the results for the binary problem, reported in Fig. 7-(a), we observe distinct trends for different configurations of feature sets and LUTs as they relate to the F1-Score at varying bit sizes. The blue line, representing the 2 features with 1 LUT, maintains a consistent F1-Score across all bitwidths (4-bit, 6-bit, and 8-bit), starting and ending at ≈80%. This consistency suggests that for tasks utilizing only two features, the complexity of the information is low enough that the benefits of increased data representation precision (offered by a higher bitwidth) do not significantly impact the model F1-Score.

In the configuration with 5 and 7 LUTs instead, there is a noticeable upward trend as the bitwidth increases from 4-bit to 8-bit. Starting ≈87.5% (≈88.8%) for the 6 features (8 features) scenario and ending at ≈91.7% (≈92.6%) for the 6 features (8 features), it suggests that having a larger bitwidth has a positive impact on the F1-Score when using more features and LUTs. Notably, the performance gain between the 6-bit and 8-bit representations for the more complex configurations is more pronounced than the gain between 4-bit and 6-bit.

In the results related to the multiclass problem, reported in Fig. 7-(b), the blue line (2 features - 1 LUT) indicates a stable F1-Score at approximately 72% across all bitwidths tested (4-bit, 6-bit, and 8-bit). Again, the flat trend line implies that in a multi-class context, increasing the bitwidth does not enhance the model's F1-Score. When the number of features and LUTs increases, we observe an upward trend in F1-Score as the bitwidth expands from 4-bit to 8-bit. Both 5-LUTs and 7-LUTs configurations start at an F1-Score of approximately 82% for 4-bit and reach an F1-Score of 88% for 8-bit. This suggests that in a multi-class problem with more features, the system likely benefits from the increased bitwidth allowing for more nuanced distinctions among the multiple classes, leading to better classification performance.

To determine the best configuration from the results, one must consider the trade-offs between the precision of the model (as indicated by the F1-Score) and the memory consumption (related to the bit size). For instance, if computational resources are limited or if the application necessitates rapid processing times, a lower bitwidth may be more advantageous, particularly when the decrease in



(a) Binary.

(b) Multiclass.

FIGURE 8. F1-scores on IoTID20 per varying bitwidths.

F1-Score is marginal. Conversely, for applications where the utmost accuracy is critical, and there is a capacity for greater computational load, opting for a higher bitwidth and a more complex configuration, like the 8 features - 7 LUTs setup, could be preferable despite the associated increase in resource demands.

B. USE CASE 2: MALICIOUS ACTIVITY CLASSIFICATION TASK IN IOT NETWORKS

To demonstrate the potential applications of the developed approach, we have considered another experiment, identifying anomalous activity across an IoT network. Specifically, we have used the IoTID20 publicly available dataset [43]. Again, features incompatible with a P4 implementation have been excluded from the dataset. As in the previous use-case, we have considered 2 sub-use-cases: (i) a binary classification problem, with the goal of distinguishing between normal and malicious flows and (ii) a multiclass classification problem, considering 4 classes: normal, DoS, Scan, Man-In-The-Middle ARP Spoofing.

The NN block, corresponding to a single LUT, is composed of 6 fully-connected layers, with 512, 256, 128, 64, 32 and 1 or 4 neurons for the last layer depending on the sub-use-case considered, respectively. ReLU has been chosen as the activation function. The architecture has been trained for 50 epochs using Adam as optimizer. Results are depicted in Fig. 7.

In the binary results, reported in Fig. 8-(a), the model utilizing only two features and a single Look-Up Table reaches an F1-Score around 84% across all bitwidths from 4-bit to 8-bit. The flatness of the line suggests that for this simple model, increasing the bitwidth does not significantly influence the predictive accuracy as measured by the F1-Score. The orange dashed line represents a medium-complexity system that processes six features with five Look-Up Tables. Here, there is a slight upward trend from approximately 84% at 4-bit to about 88% at 8-bit. This increase implies that the additional precision afforded by a higher bitwidth contributes to a better classification performance in this scenario. The green dotted line indicates the most complex system presented, with eight features managed by seven LUTs. Starting with an F1-Score just below 88% at 4-bit, it ends close to 92% at 8-bit, showing the benefit of utilizing higher bitwidths.

Finally, results concerning the multiclass problem are reported in Fig. 8-(b). Remarkably, the F1-Score for this setup starts at approximately 43% for all bitwidths in the 2 features - 1 LUT scenario, which is significantly lower than the scores in other configurations. However, when considering 6 (8) features we can observe an F1-Score starting at around 80% (82%) and finishing at ≈83% (≈84%). The F1-Score is noticeable smaller than the one obtained in the binary classification, due to the more complex nature of the multiclass problem.

As in the previous use-case, choosing the most suitable configuration requires balancing the model's accuracy (reflected by the F1-Score) with its resource demands (influenced by bit size). For example, when computational power is a bottleneck or when swift processing is essential, selecting a lower bitwidth might be beneficial, especially if it leads to only a slight reduction in F1-Score. On the other hand, in situations where precision is paramount and computational resources are not a limiting factor, it would be advantageous to select a higher bitwidth and a configuration with more complexity, such as the 8 features - 7 LUTs arrangement, even though it might call for more computational investment.

C. SWITCH: MEMORY RESOURCES AND LATENCY KPI

In order to demonstrate the advantages of the proposed LUT distillation method on programmable devices, extensive evaluations have been carried out on the Tofino 1 switch, a commercially available ASIC-programmable P4 switch equipped with 1, 10, 25, and 100 Gigabit Ethernet interfaces. The P4 source code detailed in Section IV has been compiled

TABLE 1. Programmable ASIC performance: latency and stages used as a function of the model and number of flow rules per LUT.

Model	Flow rules per LUT	Latency (ns)	Number of stages used
1 LUT	2^{16}	54	1
	2^{18}	54	1
	2^{19}	55	2
	2^{20}	57	4
	2^{21}	60	8
	2^{22}	63	11
5 LUT	2^{16}	92	3
	2^{18}	95	6
	2^{19}	81	11
7 LUT	2^{16}	111	4
	2^{18}	150	8
	2^{19}	136	12

and deployed on the switch. The Tofino switch has the capability to provide up to 3.2 Tb/s forwarding capacity and supports up to 12 programmable pipeline stages. All the results hereafter shown are extracted from the P4 compiler framework providing information on the compiled P4 code resource placement and utilization within the programmable ASIC hardware platform.

Table 1 reports the Tofino performance as a function of the deployed model (1, 5, or 7-LUT model) and number of flow-rules configured per LUT. For all employed configurations, the results show the performance of the models in terms of occupied number of stages and the latency expressed in nanoseconds, measured by the Intel P4 Insight [44] which provides a detailed evaluation of memory occupancy and latency of P4 program. For these measurements, the number of configured flow rules per table has been doubled until the switch memory capacity was exhausted.

The first part of the table reports the results for the 1-LUT model, this is the one that achieves the lowest latency in respect to all the other models. Even when the flow rules used are 2^{22}, the number of occupied stages is 11 out of 12, the latency of the 1-LUT model is just 63 ns. This is because, aside from the parser, just one match-action is required to perform the inference of the whole DNN. The minimum number of flow-rules configured for this model is 2^{16}, corresponding to the case of 2 features with 8 bits each. In this case the latency is 54 ns, and one just stage is used, leaving a significant amount of resources free for other functions.

The second part of Table 1 reports the results for the other two models, i.e., the 5 LUT and 7 LUT ones. For these models, at least two additional pipeline stages are needed leading to increased latency between 81 and 136 ns. While these models have consistently outperformed the prediction accuracy of the 1-LUT model, as discussed in Sections VI-A and VI-B, they require much more resources as testified by the rapidly-growing number of stages as a function of the number of flow rules per LUT.

VII. CONCLUSION

The integration of ML in network devices has attracted attention over the past years, due to the potential benefits it can bring. P4 programmable switches offer a way to deliver networking operations at the data plane, saving computational resources. However, the deployment of DNNs into P4 pipelines has challenges that hamper their feasibility. For this reason, in this paper, we have proposed a LUT distillation method that transforms quantized DNN structures into a simpler, cascaded architecture of flow tables, which effectively function as LUTs. Our method ensures no loss of information during this transformation and is adaptable to DNNs of varying complexities.

The proposed approach was validated in two functional use cases: cyber security DDoS mitigator and traffic classification. Results indicate that performance varies with feature set and bit size configurations. In both cases, configurations with more features and a single LUT showed improved performance with larger bit sizes.

In conclusion, this paper represents a significant step towards the integration of advanced DNN capabilities into programmable network devices. The obtained results pave the way for further exploration and development of the proposed approach. For future work, we plan to extend the discussed method to other network functions, beyond cybersecurity and traffic classification, to fully leverage the potential of DNN in network management and security. A study related to smartNIC is also planned. Additionally, a refinement of the LUT distillation process to further optimize the balance between computational efficiency and model accuracy will be investigated. Finally, the scalability and adaptability of this approach in larger, real-world network environments will be a key area of focus.

REFERENCES

[1] W. Quan et al., "AI-driven packet forwarding with programmable data plane: A survey," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 1, pp. 762–790, 1st Quart., 2023.
[2] A. Sacco, F. Esposito, and G. Marchetto, "On control and data plane programmability for data-driven networking," in *Proc. IEEE 22nd Int. Conf. High Perform. Switch. Routing (HPSR)*, 2021, pp. 1–6.
[3] F. Paolucci, F. Cugini, P. Castoldi, and T. Osiński, "Enhancing 5G SDN/NFV edge with P4 data plane programmability," *IEEE Netw.*, vol. 35, no. 3, pp. 154–160, May/Jun. 2021.
[4] F. Musumeci, A. C. Fidanci, F. Paolucci, F. Cugini, and M. Tornatore, "Machine-learning-enabled DDoS attacks detection in P4 programmable networks," *J. Netw. Syst. Manag.*, vol. 30, pp. 1–27, 2022.
[5] C. Zheng et al., "Automating in-network machine learning," 2022, *arXiv:2205.08824*.
[6] G. Siracusano et al., "Re-architecting traffic analysis with neural network interface cards," in *Proc. 19th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2022, pp. 513–533.
[7] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? Toward in-network classification," in *Proc. 18th ACM Workshop Hot Topics Netw.*, 2019, pp. 25–33.
[8] B. M. Xavier, R. S. Guimarães, G. Comarela, and M. Martinello, "Programmable switches for in-networking classification," in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.
[9] B. Coelho and A. Schaeffer-Filho, "BACKORDERS: Using random forests to detect DDoS attacks in programmable data planes," in *Proc. 5th Int. Workshop P4 Eur.*, 2022, pp. 1–7.

[10] Y.-S. Lu and K. C.-J. Lin, "Enabling inference inside software switches," in *Proc. 20th Asia-Pacific Netw. Oper. Manag. Symp. (APNOMS)*, 2019, pp. 1–4.

[11] F. Cugini et al., "Telemetry and AI-based security P4 applications for optical networks [invited]," *J. Opt. Commun. Netw.*, vol. 15, no. 1, pp. A1–A10, Jan. 2023.

[12] L. De Marinis, E. Paolini, R. A. Bakar, F. Cugini, and F. Paolucci, "Cascaded look up table distillation of P4 deep neural network switches," in *Proc. IEEE Global Commun. Conf.*, 2023, pp. 2111–2116.

[13] G. Xie, Q. Li, Y. Dong, G. Duan, Y. Jiang, and J. Duan, "Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation," in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 1938–1947.

[14] A. C. Lapolli, J. Adilson Marques, and L. P. Gaspary, "Offloading real-time DDoS attack detection to programmable data planes," in *Proc. IFIP IEEE Symp. Integr. Netw. Service Manage.*, 2019, pp. 19–27.

[15] M. A. Ridwan, N. A. M. Radzi, F. Abdullah, and Y. E. Jalil, "Applications of machine learning in networking: A survey of current issues and future challenges," *IEEE Access*, vol. 9, pp. 52523–52556, 2021.

[16] M. Usama et al., "Unsupervised machine learning for networking: Techniques, applications and research challenges," *IEEE Access*, vol. 7, pp. 65579–65615, 2019.

[17] Y. Al-Dunainawi, B. R. Al-Kaseem, and H. S. Al-Raweshidy, "Optimized artificial intelligence model for DDoS detection in SDN environment," *IEEE Access*, vol. 11, pp. 106733–106748, 2023.

[18] A. Elbery, Y. Lian, and G. Li, "Toward fair and efficient congestion control: Machine learning aided congestion control (MLACC)," in *Proc. 7th Asia-Pacific Workshop Netw.*, 2023, pp. 88–94.

[19] N. Zilberman, P. M. Watts, C. Rotsos, and A. W. Moore, "Reconfigurable network systems and software-defined networking," *Proc. IEEE*, vol. 103, no. 7, pp. 1102–1124, Jul. 2015.

[20] P. Bosshart et al., "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[21] E. F. Kfoury, J. Crichigno, and E. Bou-Harb, "An exhaustive survey on P4 programmable data plane switches: Taxonomy, applications, challenges, and future trends," *IEEE Access*, vol. 9, pp. 87094–87155, 2021.

[22] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi, "P4 edge node enabling stateful traffic engineering and cyber security," *J. Opt. Commun. Netw.*, vol. 11, no. 1, pp. A84–A95, Jan. 2019.

[23] R. Kundel, J. Blendin, T. Viernickel, B. Koldehofe, and R. Steinmetz, "P4-CoDel: Active queue management in programmable data planes," in *Proc. IEEE Conf. Netw. Funct. Virtualiz. Softw. Defined Netw. (NFV-SDN)*, 2018, pp. 1–4.

[24] C. Zheng, X. Hong, D. Ding, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "In-network machine learning using programmable network devices: A survey," *IEEE Commun. Surveys Tuts.*, vol. 26, no. 2, pp. 1171–1200, 2nd Quart., 2024.

[25] Z. Zhong et al., "IOI: In-network optical inference," in *Proc. ACM SIGCOMM Workshop Opt. Syst.*, 2021, pp. 18–22.

[26] X. Zhang, L. Cui, F. P. Tso, and W. Jia, "*pHeavy*: Predicting heavy flows in the programmable data plane," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4353–4364, Dec. 2021.

[27] J.-H. Lee and K. Singh, "Switchtree: In-network computing and traffic analyses with random forests," *Neural Comput. Appl.*, pp. 1–12, Nov. 2020.

[28] C. Zheng et al., "IIsy: Hybrid in-network classification using programmable switches," *IEEE/ACM Trans. Netw.*, early access, Feb. 16, 2024, doi: 10.1109/TNET.2024.3364757.

[29] Q. Qin, K. Poularakis, K. K. Leung, and L. Tassiulas, "Line-speed and scalable intrusion detection at the network edge via federated learning," in *Proc. IFIP Netw. Conf. (Netw.)*, 2020, pp. 352–360.

[30] D. C. Li, M. R. Maulana, and L.-D. Chou, "NNSplit-SØREN: Supporting the model implementation of large neural networks in a programmable data plane," *Comput. Netw.*, vol. 222, Feb. 2023, Art. no. 109537.

[31] T. Swamy, A. Rucker, M. Shahbaz, I. Gaur, and K. Olukotun, "Taurus: A data plane architecture for per-packet ML," in *Proc. 27th ACM Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2022, pp. 1099–1114.

[32] Y. Tokusashi, H. T. Dang, F. Pedone, R. Soulé, and N. Zilberman, "The case for in-network computing on demand," in *Proc. 14th EuroSys Conf.*, 2019, pp. 1–16. [Online]. Available: https://doi.org/10.1145/3302424.3303979

[33] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT, 2016.

[34] F. Paolucci, L. De Marinis, P. Castoldi, and F. Cugini, "Demonstration of P4 neural network switch," in *Proc. Opt. Fiber Commun. Conf. Exhibit. (OFC)*, 2021, pp. 1–3.

[35] K. Onishi, J. Yu, and M. Hashimoto, "Memory efficient training using lookup-table-based quantization for neural network," in *Proc. 2nd IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, 2020, pp. 251–255.

[36] L. Wang, X. Dong, Y. Wang, L. Liu, W. An, and Y. Guo, "Learnable lookup table for neural network quantization," in *Proc. IEEE/CVF Conf. Comput. Vis. pattern Recognit.*, 2022, pp. 12423–12433.

[37] M. S. Razlighi, M. Imani, F. Koushanfar, and T. Rosing, "LookNN: Neural network with no multiplication," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2017, pp. 1775–1780.

[38] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*. Boca Raton, FL, USA: Chapman Hall, 2022, pp. 291–326.

[39] "tofinomodel." Accessed: Apr. 13, 2024. [Online]. Available: https://github.com/barefootnetworks/Open-Tofino/tree/master/share/p4c/p4include

[40] K. Helwegen, J. Widdicombe, L. Geiger, Z. Liu, K.-T. Cheng, and R. Nusselder, "Latent weights do not exist: Rethinking binarized neural network optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–12.

[41] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016, *arXiv:1606.06160*.

[42] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Security Privacy*, vol. 1, 2018, pp. 108–116.

[43] I. Ullah and Q. H. Mahmoud, "A scheme for generating a dataset for anomalous activity detection in IoT networks," in *Proc. Can. Conf. Artif. Intell.*, 2020, pp. 508–520.

[44] "P4Insigth." [Online]. Available: https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/p4-insight.html

EMILIO PAOLINI (Student Member, IEEE) received the B.S. degree in computer engineering and the M.S. degrees in artificial intelligence and data engineering from the University of Pisa, Italy, in 2019 and 2021, respectively. He is currently pursuing the Ph.D. degree with Scuola Superiore Sant'Anna, with a scholarship co-funded by the National Research Council (CNR) and Sma-RTy Italia SRL.

From 2023 to 2024, he was a Visiting Scholar with the Department of Computer Science, Saint Louis University, St. Louis, MO, USA. His research focuses on the optimization and deployment of artificial intelligence algorithms in constrained environments.

LORENZO DE MARINIS (Member, IEEE) received the B.S. and M.S. degrees in electronic engineering from the University of Pisa in 2017 and 2019, respectively, and the Ph.D. degree from Scuola Superiore Sant'Anna (SSSA), Pisa, Italy, in 2022. From November 2021 to April 2022, he was a Visiting Scholar with the WinPhos Laboratory, Aristotle University of Thessaloniki, Thessaloniki, Greece. From September 2022 to February 2023, he was a Research Fellow of Quantum and Neuromorphic Photonics with SSSA, where he is currently an Assistant Professor. His main research concerns the conceptualization and design of photonic integrated circuits for quantum and neuromorphic applications, analog computing, photonic–electronic codesign, and machine learning for networking scenarios.

DAVIDE SCANO received the B.S. degree in telecommunication engineering from the University of Pisa in 2017, and the M.S. degree in computer science and networking from the University of Pisa and Scuola Superiore Sant'Anna, Pisa, in 2019, with a research thesis on SDN for guaranteeing QoS in network slicing. He is currently pursuing the Ph.D. degree in emerging digital technologies with Scuola Superiore Sant'Anna. His research interests are software-defined networking, next-generation software-defined networking, optical networks, and disaggregated networks. In 2020, he got a Research Scholarship at Scuola Superiore Sant'Anna.

FRANCESCO PAOLUCCI is the Head of Research with the National Inter-University Consortium for Telecommunications (CNIT), Pisa, Italy. His main research interests are in the field of networking control plane, edge/cloud networking platforms, traffic engineering, network disaggregation, advanced monitoring/telemetry, and SDN data plane programmability for edge platform and beyond 5G architectures. He has served as a Coordinator of the AI-RIDE Project (H2020 VEDLIoT open call) and the National PNRM DRONET Project. He has been involved in many national, industrial, and European research projects on next-generation networking control and monitoring (METROHAUL, B5G-OPEN, BRAINE, DESIRE6G, SmartEdge, CLEVER, SEASON, and NATWORK). He is the coauthor of three IETF Internet Drafts, more than 200 publications in international journals, conference proceedings, and book chapters, and filed four international patents. He is an Associate Editor of the IEEE/OSA JOURNAL OF OPTICAL COMMUNICATIONS AND NETWORKING and an Executive Editor of the *Transactions on Emerging Telecommunications Technologies*.

Open Access funding provided by 'Scuola Superiore "S.Anna" di Studi Universitari e di Perfezionamento' within the CRUI CARE Agreement