Cascaded Look Up Table Distillation of P4 Deep Neural Network Switches

Lorenzo De Marinis, Emilio Paolini, Rana Abu Bakar
Scuola Superiore Sant'Anna
Pisa, Italy
lorenzo.demarinis@santannapisa.it

Filippo Cugini, Francesco Paolucci
CNIT
Pisa, Italy
francesco.paolucci@cnit.it

***Abstract*—In-network function offloading represents a key enabler of the SDN-based data plane programmability to enhance network operation and awareness while speeding up applications and reducing the energy footprint. The offload of network functions exploiting machine learning and artificial intelligence has been recently considered with intermediate solutions such as feature extraction acceleration and mixed architectures including AI-specific platforms (e.g., GPU, FPGA).**

Indeed, the P4 language enables the programmability of deep neural networks inside the pipelines of both software and hardware switches and NICs. However, programmable hardware pipeline chipsets suffer from significant computing capability limitations (e.g., missing arithmetic logic units, limited and slow stateful registers) preventing the plain programmability of a deep neural network (DNN) operating at wirespeed.

This paper proposes an innovative knowledge distillation technique that maps a DNN into a cascade of lookup tables (i.e., flow tables) with limited entry size. The proposed mapping avoids stateful elements and maths operators, whose requirement prevented the deployment of DNNs within hardware switches up to now. The evaluation is carried out considering a cyber security use case targeting a DDoS mitigator network function, showing negligible impact due to the lossless mapping reduction and feature quantization.

***Index Terms*—In-network computing, artificial intelligence, knowledge distillation, P4, hardware acceleration, DDoS mitigation.**

I. Introduction

The advent of in-network Artificial Intelligence (AI) will enable future network devices to process data (i.e., packets, flows, aggregate traffic) at wirespeed with the support of builtin AI-enabled components [1], [2]. In-network function offloading is becoming popular thanks to the flexibility and maturity of programmable data plane languages such as P4, and has the potential to bring programmable and versatile networking operation at the data plane thus saving computational resources for applications at the edge and with limited energy consumption, since re-using the same network infrastructure devices [3].

So far, network accelerations encompassing AI have been delegated to external backends, for example at co-located Field Programmable Gate Arrays (FPGAs) or at Graphics Processing Units (GPUs)-equipped servers, that lead to suboptimal solutions providing additional latency and low energy efficiency. A relevant intermediate solution encompasses AI features extraction at the data plane and cloud-based inference [4]. Recent attempts to introduce machine learning (ML) algorithms directly at the data plane of programmable devices have leveraged reconfigurable pipelines able to reproduce or approximate ML algorithms as a series of flow tables and stateful operations [5]. For example, existing works target binary neural networks at the Network Interface Card (NIC) [6] and a restricted family of machine learning algorithms, such as Support Vector Machines (SVMs) [7], Decision Trees (DTs) [8] and Random Forests (RFs) [9].

Conversely, Deep Neural Network (DNN) pipeline deployments inside programmable network devices are facing relevant hardware constraints due to the specific DNN workflow requiring fast Arithmetic Logic Units (ALU) for neuron computation and high speed update of stateful memories, while hardware programmable pipelines are designed to support a large number of match-action flow tables with very limited arithmetic variable manipulation and low update rates for stateful memories. For this reason, so far, only P4 DNN implementations for software switches or smart NICs have been presented [10], [11], with limited performance capabilities.

In this paper, to the best of our knowledge, we propose for the first time a knowledge distillation method that maps the P4 DNN pipeline, in principle requiring ALU and significant stateful capabilities, into a cascaded architecture of flow tables, matching the input features with the DNN outcomes as a simple aggregated input/output Lookup Table (LUT). The developed distillation technique is lossless with respect to the originary pipeline, meaning that no information is lost during the mapping from the DNN to the LUT. Moreover, provided that inputs and outputs match the hardware memory constraints, the method is agnostic to the DNN complexity, requiring a fixed amount of resources for the distilled LUT. We apply the proposed workflow to a cyber security use case targeting Distributed Denial of Service (DDoS) attack mitigation, showing the applicability of the proposed distillation technique inside a P4 pipeline, the feasibility of the approach and its advantages in terms of efficient use of stateless features with respect to other existing offloading deployments in the data plane, e.g., entropy-based anomaly detection strategies [12], typically requiring large stateful memories. The proposed distillation technique represents the first theoretical step of the applicability of fully offloaded AI-

based functions inside network programmable devices.

II. In-Network programmable AI: strategies and current limitations

The programmability of the Software Defined Networking data plane has reached a mature level of abstraction, language capabilities (e.g., P4) and data plane acceleration tools (e.g., eBPF, DPDK). This enables network engineers to concentrate on legacy network functions development while abstracting from the complexity of the hardware platforms. However, when novel data plane offloading strategies are conceived bringing more complex requirements (e.g., a machine learning algorithm), relevant limitations are still in place, being the main programmable hardware platforms designed for specific network functions.

The P4 language enables in-network functions programming as pipeline structures and logic workflows. Thanks to stateful objects (e.g., registers) any packet content may be processed in combination with flow/session statistics. In addition, thanks to the high-level language property logic and arithmetic manipulation supported by P4, it is possible to instruct the arithmetic manipulation of metadata variables in order to derive neuron compute functions, including the non-linear stage at the end of the linear combination of neuron states and weights. Such plain approach has two main drawbacks: 1) parallelized operation is either prevented or strongly limited; 2) Hardware backends do not support all the P4 language capabilities.

The former drawback results in suboptimal latency performance. As shown in the P4 DNN implementation in software switches exploiting all 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]. The latter drawback is due to the hardware vendors chipset design policies, aiming at accelerating typical network functions such as routing, forwarding, and load balancing.

In Fig. 1 we show the different flavours in the design of a programmable network function exploiting a DNN. In flavour a) the switch/NIC is designed to receive and match selected packets/flows to be sent to an external device (e.g., FPGA) implementing the DNN. The outcome of the DNN is sent back to the switch/NIC using internal interfaces (e.g., dedicated or SDN control plane interfaces). In this design, the packets may be either stored in specific buffers waiting for prediction/classification or directly steered to the external device. In both cases, extra delays are introduced due to the device chaining. Moreover, the solution is not power efficient, since requiring two devices operating at wirespeed. In flavour b) the switch/NIC design is enriched to host packet and metadata processing in order to extract the DNN features at runtime. Feature extraction represents one of the most time and resource consuming steps if performed using software-based functions, whereas it has been demonstrated to be fast and effective at the data plane level exploiting programmable parsers and a limited amount of memory registers and counters [4]. The DNN engine is outsourced again to a dedicated device, e.g., a GPU, while packet enforcement and deparser stages run on the switch/NIC. Again, the packets need to be buffered while waiting for the DNN outcome. Finally, flavour c) represents the proposed solution offloading a complete pipeline including the DNN function inside the programmable switch/NIC, without resorting to any additional device. The design includes parsers, feature extraction, and the DNN macro-function at wirespeed, thus packet buffering (and latency) is minimized and just one device is utilized for the entire operation, leading to significant power savings, since the impact of a programmable chipset in respect to fixed-function ASIC is negligible [13]. Moreover, the modular P4 design allows for multiple network functions acceleration inside the same pipeline/device, thus additional functions (e.g., forwarding/steering, load balancer) may co-exist with the DNN-based in-network function.

While the last solution is the only one that allows achieving fully in-network AI functionalities, it is not possible to deploy neural networks within the dataplane pipeline with common methods. DNNs are distributed architectures of interconnected primitives (artificial neurons), whose operation relies on the repetition of simple mathematical tasks, mainly constituted by multiply-accumulate operations. These AI models learn from examples to build good approximate solutions to tasks, and are particularly resilient to noisy inputs and low-precision representation formats if trained accordingly [14]. Hence, the high resolution of floating-point is not mandatory and the related computational burden can be avoided with integers used instead. However, the programmable ASIC pipelines of commercially available P4 switches do not support, up to date, floats or fast integer arithmetic [11]. The requirement of wirespeed processing allows only for specific tasks such as flow table matching, with variables stored in an integer format. The lack of multiply-accumulate operations prevents the deployment of DNNs on current hardware P4 switches, even if the parameters are quantized in an integer format.

An implementation example of DNN inside a P4 software switch is shown in [15], comprising two main P4 stages: the feature extraction and the DNN function. Indeed, the P4 language allows the definition and declaration of functions implementing a DNN, using integer-based neuron computation updates and tree-based structures, with limited parallelized computations leading to low performance capabilities. The porting of such P4 codes inside a programmable ASIC backend is successful only for the feature extraction stage, while the DNN function requires a complete remapping design, mainly avoiding ALU-based operations.

III. Cascaded LUT as a Deep Neural Network

To deploy ML models within hardware pipelines with strictly limited arithmetic capabilities, we have conceived a strategy to distill the knowledge of a trained DNN into a LUT, with no loss of information, leveraging the constraint of integer-encoded variables as an advantage. Fig. 2 represents the core idea of our knowledge distillation technique. Consider

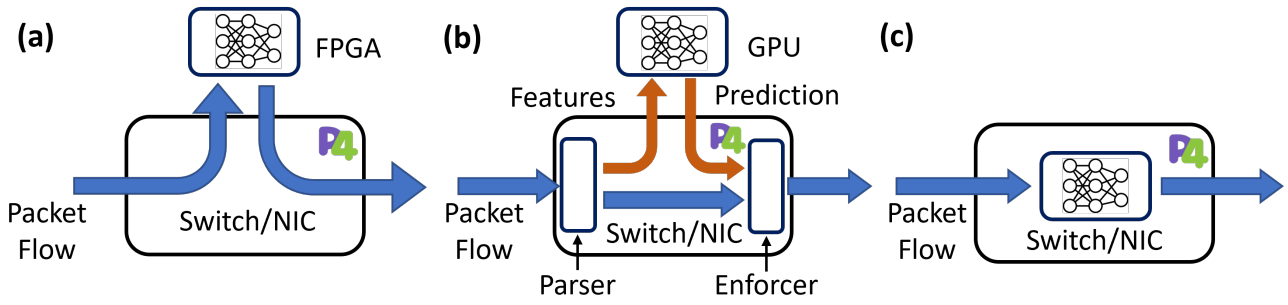

Fig. 1: Strategies for the deployment of AI models at the Switch/NIC side. (a) the packet flow is directly sent to a co-located FPGA, wich parse the data and run a DNN. (b) Hybrid strategy where packet features are extracted within the switch/NIC and sent to the DNN run on a GPU. (c) The ML model is deployed in the pipeline within the switch/NIC.

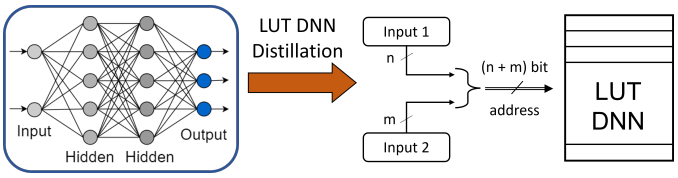

Fig. 2: Deep neural network distillation in a lookup table for inference acceleration. Integer-encoded inputs are treated as a compound address of the LUT, whose entries encode the corresponding output of the distilled DNN.

the case of a 2-input network, whose inputs are integers with n and m bits, respectively. Once the DNN is trained, the inputs are paired and treated as a compound address of the final LUT, whose bitwidth is $n + m$. The table is generated by simply collecting the DNN outputs for all the possible input-pair combinations, which are $2^{(n+m)}$. With this method, the inference of a DNN is reduced to a match-action on a flow table. It is worth noting that this is a lossless procedure as it considers exhaustively all the possible DNN outcomes.

With respect to other table-based quantization strategies [16]–[18], we rely on LUTs not to accelerate the quantization process nor to perform multiplications at higher speed, but to distill the knowledge of the whole DNN. Logically, the LUT distillation method has some limitations: first (i) memory consumption grows exponentially with the number of bits composing the input features, second (ii) features should not use the floating-point format but the integer one, and lastly (iii) also the number of output variables linearly affect the LUT size, i.e., if the DNN produces three output then three entries are added in the LUT per input combination. Nonetheless, the LUT distillation method presents stark advantages, with the most important being making DNNs deployable in hardware P4 switches. No constraint is set to the DNN complexity nor type: if the requirements on inputs and outputs are met, this method applies to huge DNNs and even to other ML algorithms. Training more complex DNNs results, up to some extent, in a more accurate model [14], with the drawback of increasing the training and LUT 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 replacing the LUT.

In practical cases, several input features are used in ML tasks, resulting in huge memory requirements that would prevent the use of this method. Consider a DNN with one 8-bit output and four 8-bit input features, the correspondent LUT entries would be $2^{32} = 0.537$ Gbytes, a value well beyond the memory capabilities of current P4 switches. 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 layer 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. 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 24.6 kbytes. 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.

IV. Demonstration with DDoS Mitigation

To demonstrate the capabilities and potentiality of the LUT distillation method, we carried out experiments considering a DDoS mitigation problem. We have chosen the UNSW-NB15 Network Intrusion Detection data set [19], which contains both normal and malicious traffic packets. For this task, the ML model is developed to act as a firewall, categorizing benign and malicious packets. The neural model can be embedded in a P4 pipeline that extracts features and enforces actions on packets like the one described in [11]. Among the 49 features of the UNSW-NB15 dataset, we chose 2 stateless and 6 stateful features as possible inputs to the DNN model.

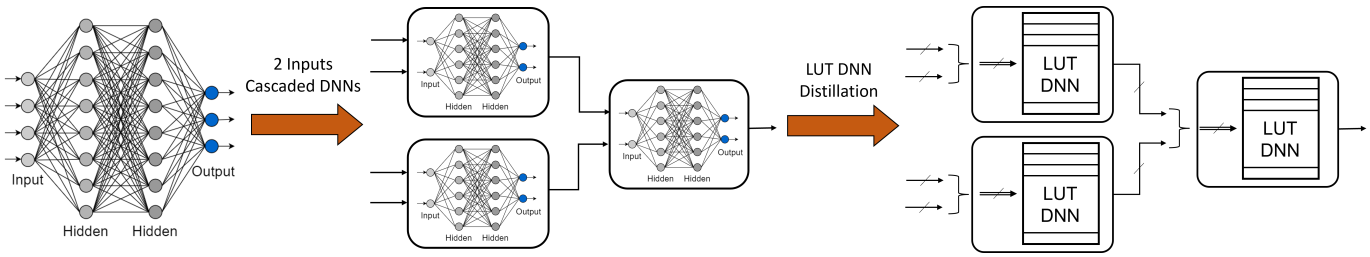

Fig. 3: Cascaded LUT DNN design strategy. Instead of training a single DNN, input features are grouped in pairs and sent to separate DNNs, which are in turn paired recursively until the final output. After training the models are distilled into LUTs.

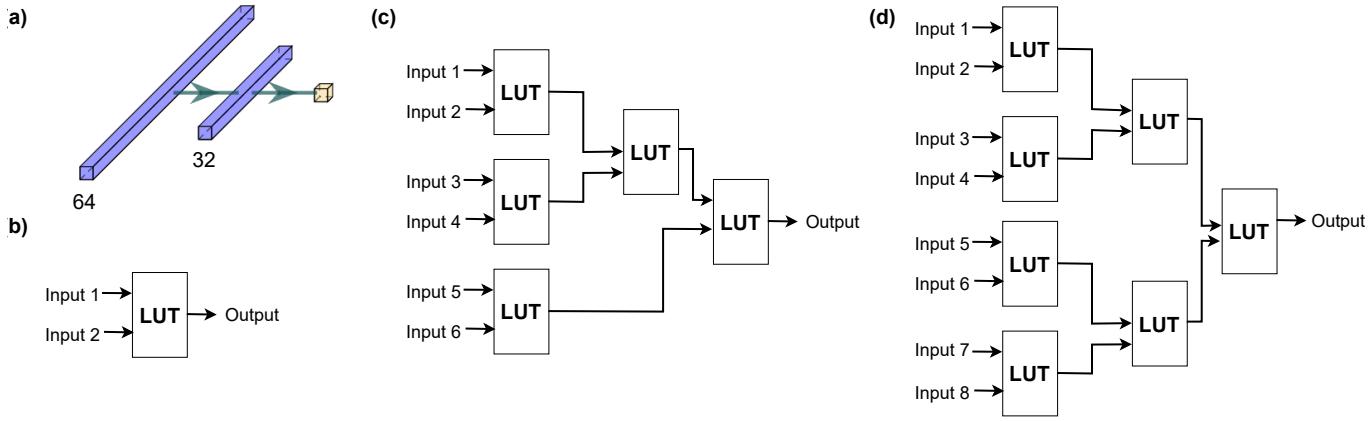

Fig. 4: (a) The 2-input base neural network module, this corresponds to a distilled LUT; (b) model architecture for stateful features; (c) model architecture for the stateless features; (d) model architecture for the combined features.

Depending on the type of features used, we developed three different architectures that use (i) stateful, (ii) stateless, and (iii) both types of features. Specifically, we chose as stateful features the number of connections having the same source (*ct_src_ltm*) and destination (*ct_dst_ltm*) IP address within the last processed 100 packets. As stateless features, we chose the IP protocol (*IP_proto*), IP Time To Live (TTL) (*s_TTL*), the source and destination TCP window advertisement (*swin* and *dwin*), source-to-destination (*s_bytes*) and destination-to-source transaction bytes (*d_bytes*).

The dataset is highly unbalanced with respect to the normal traffic samples, hence a random oversampling technique is applied to balance the training set, obtaining a $50-50$ % distribution among the two classes. The test set has been kept unbalanced since its distribution must represent as much as possible real data, resulting in a balance of $90-10$ %. Concerning the DNN models, we used a 2-input base module, which is depicted in Fig. 4(a). The module is composed of 3 layers, with 64, 32, and 1 neuron, respectively. The ReLU is adopted as activation function in the first two layers, while the output layers exploit the sigmoid function to perform binary classification of the packets in the benign or attack class. The size and shape of this basic model have been chosen to guarantee a rather rapid training and LUT distillation. Still, the DNN complexity can be increased without impacting memory consumption to have higher prediction accuracies.

Once trained, the DNN model is converted to a single LUT with the method described in Sec. III, and is directly exploited in the experiment with stateful-only features, as only 2 features are used. Fig. 4(b) represents the distilled table for this case. In the other two architectures, which exploit stateless-only or both type of features, the number of inputs is 6 and 8, respectively. Hence, several instances of the Neural Network (NN) module are stacked on top of each other, trained, and consequently compressed to LUTs, as sketched in Fig. 4(c) and Fig. 4(d).

Considered Features	**F1-score 8-bit inputs**	**F1-score with reduced bitwidh**
Stateful	42.96 %	
Stateless	93.75 %	92.19 %
Combined	93.26 %	93.91 %

TABLE I: F1-score on the test set for the models, considering features on 8 bits and a with reduced bitwidth.

In a first set of experiments, we used input features represented on 8 bits. For this case, the *s_bytes* and *d_bytes* features are quantized on 8 bits relying on a quantile-based discretization, while the others are already compatible with the chosen format. The trained model accuracy has been validated on a test set with the F1-score, which is the performance metric used with unbalanced datasets [14]. The results of these experiments are reported in the second

column of Table I. The model with stateful-only features provides an F1-score lower than 50%, a non-acceptable performance for this DDoS detection task. On the other hand, the DNN models with stateless-only and combined features provide a consistent accuracy of 93.75% and 93.26%, respectively. When comparing these two approaches, the stateless features scenario slightly outperforms the combined features. Moreover, stateless features are more amenable to a P4 implementation since they can be directly extracted from packets, without further computations. On the contrary, stateful features need co-located memory stages and additional operations to be computed, making them less suitable for a future P4 application. Hence, the use-case that considers only stateless features is the favorite candidate, for the easily implementable features and slightly less complex architecture.

Although these results already validate the proposed method, there may be the need to further shrink the table size to make them more compatible with P4 switches, as in most devices flow tables entries should be kept limited for memory management. In compliance with this constraint, we have carried out another set of experiments with the same architectures as the first one, but pairing input features so that their combined bitwidth results less than 14 (i.e., LUT size below 10000 entries). Table II reports how the features were combined in this case with the resulting total number of bits. The output of each DNN is encoded on 6 bits. The second column of Table II reports the results for this scenario in the case of stateless-only and combined features. The F1-scores are above 92 %, with a slight decrease in accuracy for the stateless-only case, while a small increase is experienced in the model with combined features.

Input	Number of bits
{*swin, quantized_s_TTL*}	11
{*dwin, quantized_IP_proto*}	11
{*ct_src_ltm, quantized_dbytes*}	13
{*ct_dst_ltm, quantized_sbytes*}	13

TABLE II: Input features and corresponding biwidths for the combined scenario in the reduced bitwidth experiments.

V. Conclusions

The development of AI models able to run at wirespeed and low energy consumption in compliance with the physical hardware is the key missing piece to bring advanced in-network functions in the data plane. While DNNs have been implemented in P4 software switches and smart NICs, the limited computing capability of hardware switches prevents their deployment. To circumvent this issue, in this paper we proposed and demonstrated a knowledge distillation technique that allows to convert DNNs into a cascaded architecture based on LUTs, with no information loss and with a resource usage independent from the DNN complexity. Moreover, in case the DNN needs to be retrained as for tackling a different task, the system can be reconfigured on the fly by simply changing the LUT flow entries, thus avoiding to recompile the whole P4 pipeline. To demonstrate the working principle, we developed a ML model to act as a DDoS attack mitigator task with the UNSW-NB15 dataset. We chose 2 stateful and 6 stateless features among the available ones, and developed three DNN models for different kind of input features: stateful-only, stateless-only and both type of features. We carried a first set of experiments encoding inputs on 8-bits, and a second set with reduced bitwidths to keep the single LUT size below ten thousand entries, for better hardware compatibility. In all cases the models for the stateless-only and aggregated features achieved a F1-score above 92 %.

The provided results represent the first relevant step of the applicability of DNN offloading inside P4 backends. Further works will deploy the considered solution in different P4-enabled hardware platforms (e.g., Tofino switches, smart NICs, Data Processing Units) evaluating the performance in terms of required hardware resources, throughput, latency and power savings.

Acknowledgment

This work has been funded by the European Commission Horizon Europe KDT JU CLEVER (GA 101097560) Project. The work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of the NextGenerationEU partnership on "Telecommunications of the Future" (PE00000001, program "RESTART").

References

[1] W. Quan, Z. Xu, M. Liu, N. Cheng, G. Liu, D. Gao, H. Zhang, X. Shen, and W. Zhuang, "AI-driven Packet Forwarding with Programmable Data Plane: A Survey," *IEEE Communications Surveys & Tutorials*, pp. 1–1, 2022.

[2] A. Sacco, F. Esposito, and G. Marchetto, "On Control and Data Plane Programmability for Data-Driven Networking," in *2021 IEEE 22nd International Conference on High Performance Switching and 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 Network*, vol. 35, no. 3, pp. 154–160, 2021.

[4] F. Musumeci, A. C. Fidanci, F. Paolucci, F. Cugini, and M. Tornatore, "Machine-learning-enabled DDoS attacks detection in P4 programmable networks," *Journal of Network and Systems Management*, vol. 30, pp. 1–27, 2022.

[5] C. Zheng, M. Zang, X. Hong, R. Bensoussane, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "Automating in-network machine learning," in *arXiv*, 2022.

[6] G. Siracusano, S. Galea, D. Sanvito, M. Malekzadeh, G. Antichi, P. Costa, H. Haddadi, and R. Bifulco, "Re-architecting traffic analysis with neural network interface cards," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 513–533.

[7] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," in *Proceedings of the 18th ACM workshop on hot topics in networks*, 2019, pp. 25–33.

[8] B. M. Xavier, R. S. Guimarães, G. Comarela, and M. Martinello, "Programmable Switches for in-Networking Classification," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[9] B. Coelho and A. Schaeffer-Filho, "BACKORDERS: using random forests to detect DDoS attacks in programmable data planes," in *Proceedings of the 5th International Workshop on P4 in Europe*, 2022, pp. 1–7.

[10] Y.-S. Lu and K. C.-J. Lin, "Enabling Inference Inside Software Switches," in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2019, pp. 1–4.

[11] F. Cugini, D. Scano, A. Giorgetti, A. Sgambelluri, L. De Marinis, P. Castoldi, and F. Paolucci, "Telemetry and AI-based security P4 applications for optical networks [Invited]," *Journal of Optical Communications and Networking*, vol. 15, no. 1, pp. A1–A10, 2023.

[12] A. C. Lapolli, J. Adilson Marques, and L. P. Gaspary, "Offloading real-time ddos attack detection to programmable data planes," in *2019 IFIP IEEE Symposium on Integrated Network and Service Management*, 2019, pp. 19–27.

[13] Y. Tokusashi, H. T. Dang, F. Pedone, R. Soulé, and N. Zilberman, "The case for in-network computing on demand," in *Proceedings of the Fourteenth EuroSys Conference 2019*, ser. EuroSys '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3302424.3303979

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[15] F. Paolucci, L. De Marinis, P. Castoldi, and F. Cugini, "Demonstration of P4 neural network switch," in *2021 Optical Fiber Communications Conference and Exhibition (OFC)*, 2021, pp. 1–3.

[16] L. Wang, X. Dong, Y. Wang, L. Liu, W. An, and Y. Guo, "Learnable lookup table for neural network quantization," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 12 423–12 433.

[17] M. S. Razlighi, M. Imani, F. Koushanfar, and T. Rosing, "Looknn: Neural network with no multiplication," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1775–1780.

[18] K. Onishi, M. Hashimoto *et al.*, "Memory efficient training using lookup-table-based quantization for neural network," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2020, pp. 251–255.

[19] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *2015 military communications and information systems conference (MilCIS)*. IEEE, 2015, pp. 1–6.