Distributed DNN Serving in the Network Data Plane

Kamran Razavi
Technical University of Darmstadt

George Karlos
Vrije Universiteit Amsterdam

Vinod Nigade
Vrije Universiteit Amsterdam

Max Mühlhäuser
Technical University of Darmstadt

Lin Wang
Technical University of Darmstadt
Vrije Universiteit Amsterdam

ABSTRACT

Programmable networks have received tremendous attention recently. Apart from exciting network innovations, in-network computing has been explored as a means to accelerate a variety of distributed systems concerns, by leveraging programmable network devices. In this paper, we extend in-network computing to an important class of applications called deep neural network (DNN) serving. In particular, we propose to run DNN inferences in the network data plane in a distributed fashion and make our programmable network a powerful accelerator for DNN serving. We demonstrate the feasibility of this idea through a case study with a real-world DNN on a typical data center network architecture.

CCS CONCEPTS

• **Networks** → **Programmable networks**; **In-network processing**; **Data center networks**;

KEYWORDS

programmable networks, in-network computing, DNN serving

ACM Reference Format:
Kamran Razavi, George Karlos, Vinod Nigade, Max Mühlhäuser, and Lin Wang. 2022. Distributed DNN Serving in the Network Data Plane. In *P4 Workshop in Europe (EuroP4 '22), December 9, 2022, Roma, Italy*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3565475.3569079

1 INTRODUCTION

The emergence of programmable, high-performance network switches and SmartNICs, has not only enabled exciting innovations in networking but also inspired a new computing paradigm called in-network computing [6, 12]. With in-network computing, programmable network devices are instructed to accelerate application components by leveraging the high-throughput, low-latency processing capabilities, and convenient on-path placement of these devices [2]. Example applications that have been proven to benefit from in-network computing are caching [5], aggregation [10, 13], agreement [4], and database query processing [18].

EuroP4 '22, December 9, 2022, Roma, Italy

ACM ISBN 978-1-4503-9935-7/22/12…$15.00
https://doi.org/10.1145/3565475.3569079

Deep learning has become the de facto approach for various inference tasks such as object detection and speech recognition. With the widespread adoption of deep learning, it becomes critical that we can serve DNNs with high performance in terms of both throughput and latency. DNN serving is usually done preferably with high-end accelerators like GPUs/TPUs over CPUs due to higher efficiency and lower costs [14]. GPUs/TPUs typically employ batching to improve throughput, at the cost of increased inference latency [14].

Considering both the fast development of programmable network devices and the high demand for DNN serving, we ask a bold question: *Can we leverage a programmable network to perform DNN serving?* Given that a modern Tofino2 switch can process packets with nanosecond latency, and at the rate of billions of packets per second [1], DNN serving would achieve an unprecedented level of performance if the DNN can be executed entirely in the data plane of the programmable network. Note that with this design the inference can be performed "on the fly" while transferring DNN input data on the network, eliminating the need of accelerators.

Prior work has explored the intersection of programmable networks and machine learning. For example, in-network aggregation has been used to accelerate the gradient synchronization in data-parallel DNN training [10, 13]. Other work has explored data-plane packet classification by running per-packet inference tasks, like decision trees, SVMs and small (binary) neural networks, on programmable switches and SmartNICs [16, 19]. Confined to a single device, such approaches limit the size of the supported ML models, and work towards addressing this issue only involves new hardware architectures [17]. So far, and to the best of our knowledge, none of these efforts support the serving of large DNNs (models with millions of weights) across a network of programmable network devices targeting user applications.

In this paper, we propose an in-network system for fast, end-to-end DNN serving by distributing a DNN across a network of programmable switches, as depicted in Figure 1. Our inspiration stems from the observation that DNNs are dataflow computations similar to how packets flow through a network. Based on that, we ❶ map the neurons in the DNN to the physical network switches, ❷ craft and route packets carrying the input/intermediate data to go through the switches containing the corresponding neurons, and ❸ instruct each switch to perform the computations specified by the neurons assigned to the switch. In the rest of this paper, we demonstrate the feasibility of this idea through a case study with a real-world DNN and a typical data center network architecture. We also discuss further challenges.

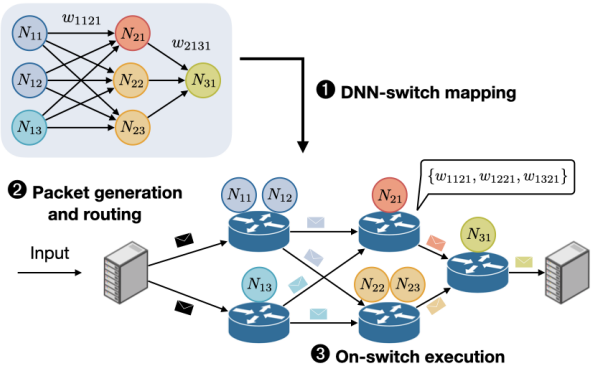

Figure 1: An overview of in-network DNN serving.

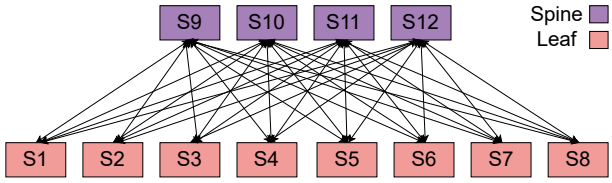

Figure 2: The spine/leaf network architecture.

2 A CASE STUDY WITH MINI-ALEXNET

We use a simplified version of AlexNet [9] (mini-AlexNet) used in [11] and trained on CIFAR-10 [8] with three dimensions (height, width, and channel). We map the mini-AlexNet network to a set of programmable switches using the data center network architecture (spine/leaf) adapted from the Google infrastructure [15].

The mini-AlexNet neural network is described in Table 1. It consists of an input layer and three convolutional layers followed by three fully connected layers. The convolutional layers are composed of convolutional filters, ReLU activation functions, and 2D Max Pooling (2x2) layers. Due to the fact that the state-of-the-art programmable switches (Barefoot Tofino2 [3]) are not equipped with Floating Point Units (FPUs), we store the inputs and weights in 8-bit integers. To avoid multiplication and aggregation overflow, we can increase the input values to 32-bit integers as it does not affect the number of operations and the cost of storing results is either temporary, or minuscule compared to weights. Previous work has shown that fixed-point arithmetic is faster than floating point equivalent accuracy, and 8-bit precision is sufficient for DNN inference [17].

DNN-switch mapping. The data center network architecture we use is depicted in Figure 2. It consists of four programmable switches as spines and eight programmable switches as leaves. Each leaf switch is connected to all spine switches with 400GbE links. The spine/leaf architecture advantage is that any leaf (spine) switch is connected to any other leaf (spine) switch in exactly two hops. We use this property to map the layers of the DNN to just leaf switches (one layer is mapped to one switch) to avoid stragglers in the synchronization.

The Tofino2 switch we consider here has four processing pipelines, each equipped with eight 400GbE ports supporting up to 12.8Tb/s of aggregate throughput. To leverage the processing power of all four pipelines of each switch, we distribute the neurons of the same layer to all the pipelines on the same switch (as the neurons in the same layer do not need to communicate with each other) evenly based on the layer type: For the *convolutional layer* we divide the number of filters by the number of switch pipelines and allocate the convolutional filters to the respective pipelines. For the *dense layer*, we divide the number of dense neurons by the number of switch pipelines and store the dense neurons' weights in the respective pipelines.

In the mini-AlexNet scenario, after we get the input from the client (on S1), we allocate and store 16 convolutional filters to each pipeline, as the first convolutional layer (on S2) has 64 filters. After the on-switch execution (detailed later), the packets for the next layer are generated (detailed later) and are sent to the next convolutional layer (on S3). We follow the same procedure until we reach the first dense layer (on S5) with 4096 neurons. We divide the dense layer into four parts and allocate each part containing 1024 neurons to each of the pipelines and perform the neuron computations until we reach the output layer (on S7), where we obtain the prediction result and send it back to the client.

Packet generation and routing. Once the DNN has been partitioned and the neurons deployed on the switches, the next step is to generate packets and route them on the network following the DNN dataflow. For the input layer, the input data is encapsulated into as many packets as are required (based on the input and the packet header size) and is multicast to the switch's pipelines (as the neurons are distributed among all pipelines) hosting neurons that are directly connected to that layer. Upon completing its computations, a layer will emit packets encapsulating the input for the next layer to the spine switches, and these packets will be multicast to the next associated switch hosting the next layer. Each switch maintains a forwarding table applying the above logic to route the packets inside the network. Each packet carries a label through which its target layer can be identified. The switch multicasts the packet based on the label to all the pipelines of the next switch. Upon a packet's arrival, the switch knows the computations to apply to the data carried by that particular packet.

In the mini-AlexNet scenario, the input switch (S1) multicasts the input to all the spine switches (with recirculation), and the switches in the spine layer redirect the packets to the first convolutional layer (on S2). After the processing of the first convolutional layer is done, each pipeline has 1/4 of the input for the next layer. With the spine/leaf design, we transfer each input segment to the spine switches, and there we multicast the input segments to all four pipelines of the next switch (on S3). Each pipeline in S3 receives packets from a switch in the spine layer, shaping the current layer's input. We follow the same procedure until we reach the output layer, where we send back the final prediction.

On-switch execution. Switches will perform computations for the neurons they host upon packet arrivals. More specifically, as each pipeline of the switch gets the entire input, we perform the computations based on the layer type. If the layer is a convolutional layer, we apply the filter to a subset of the input and store the result

	mini-AlexNet (CIFAR-10)	Number of Parameters	Number of Operations	Memory (byte)	Mapped Switch
Layer 1	Input: $32 \times 32 \times 3$	0	0	3,072	S1
Layer 2	Conv1: $3 \times 3 \times 3 \times 64$	1,792	3,110,400	16,192	S2
Layer 3	Conv2: $3 \times 3 \times 64 \times 192$	110,784	37,347,648	117.696	S3
Layer 4	Conv3: $3 \times 3 \times 192 \times 384$	663,939	21,227,520	665,475	S4
Layer 5	FC1: 4096	6,295,552	12,582,911	6,299,648	S5
Layer 6	FC2: 2048	8,390,656	16,777,215	8,392,704	S6
Layer 7	FC3 (Output): 10	20,490	40,959	20,500	S7

Table 1: The mini-AlexNet network. It contains seven layers, with over 15 million parameters and 91 million operations requiring less than 16MB of memory to store.

of the dot product. If it is a dense layer, we multiply all the input values by all the weights and aggregate their results.

We decompose each multiplication into a number of shifts. The input is shifted left by i if the i-th bit of the 8-bit weight is set, otherwise by 0. All shifts are performed in parallel, in a single stage, and the intermediate results are stored in temporaries. This step takes one stage, assuming predicate instructions (to check if the i-th bit is set), otherwise two. Then, a reduction step aggregates the intermediate results. Since we use 8-bit weights, this step takes three stages. Given N free ALUs in the first stage, we can perform $N/8$ multiplications in parallel, each of which has a maximum depth of five stages. This allows us to replicate the process a number of times without recirculation. The (intermediate) result of each multiplication is accumulated to a register, with a cost, in terms of stages, logarithmic to the number of multiplications performed. If the dot product requires more multiplications, the packet is recirculated.

Then we apply the ReLU activation function to the result of the dot product by checking whether the most significant bit (MSB) is 1 (sign bit) and replacing the value with 0. For the max pooling part, we check whether the current value is the last piece of the pooling window. A major challenge here is to find the pooling elements due to the fact that there is no mod operation in the switches available. To avoid this issue, we process the inputs in the order of the max pooling window. For a simple 2×2 pooling window, after we processed all four window's values, we get the maximum of them in four cycles (three comparisons in total; two cycles for the first two comparisons in parallel and storing the result and two cycles for the second stage comparison). To meet the promised 12.8Tb/s throughput, Tofino2 allows a limited number of operations per packet traversal (few 10's of multiplications like the one described above). Therefore, we need to recirculate to process all the inputs for all the filters in the same pipeline. For each filter/neuron on a switch, the switch accumulates the multiplication results and maintains a counter to ensure that packets from all weights have been processed before emitting the result as a packet to the downstream layers.

In Table 1 we calculate the number of operations and memory requirements for each layer of mini-AlexNet. Even the most memory hungry of the layers (Layer 6) requires less than 10% of the available memory on the Tofino2 (a couple of hundred of MBs). However, the number of operations greatly exceeds the 10's of operations we can perform in one traversal. To solve this issue, we recirculate the input in the switch with a new set of operations until all the required operations are done. In the most computation-intensive layer (Layer 3), each pipeline needs to compute roughly $10,000,000$ operations, we need to recirculate the same input less than $1,000,000$ times. The packet recirculating comes with a latency cost similar to parsing another packet. Tofino2 can potentially process six billion packets (1.5 billion packets per pipeline), resulting in less than 1ms to process even the most computation-intensive layer in our scenario. In total, a set of available switches in the data centers require less than $2ms$ to perform inference in the mini-AlexNet scenario. Compared to the evaluations reported in [11], our in-network DNN serving system not only reduces the inference latency by over 2× and 2.5× compared to CPU and GPU, respectively but also eliminates the necessity of having inference servers.

3 CHALLENGES AND DISCUSSION

Accuracy improvement. Floating point addition on programmable switches has been carefully explored in [20] while other more complex arithmetic operations are still not feasible with the current switch design. Currently, we use 8-bit fixed point weights as we are still not able to implement the floating point multiplication operations in the data plane respecting the memory access limitation of Tofino2 switches. We plan to explore how packet re-circulation can help work around this limitation.

Support for more complex layer types. So far, we have only discussed how to handle DNNs with convolutional and dense layers. However, popular DNNs typically involve a variety of layer types with more complex structures and activation functions, calling for a careful design of the data structure. There are also layers with nonlinear activation functions like `tanh` and `sigmoid`, which are currently hard to perform on programmable switches.

Fault tolerance. Switches and links can fail, and ensuring that the final prediction is generated without being affected by such failures is essential. Also, a packet loss between switches could render a DNN execution stagnation due to the use of the per-neuron counter. We acknowledge that achieving reliability for stateful in-network computing like DNN serving is a big challenge, which has not been extensively studied yet [7]. We leave this for future work.

ACKNOWLEDGEMENTS

This work has been partially funded by the German Research Foundation (DFG) within the Collaborative Research Center (CRC) 1053 MAKI, the Dutch Research Council (NWO) Open Competition Domain Science XS Grant 12611, and Google Research.

REFERENCES

[1] Anurag Agrawal and Changhoon Kim. 2020. Intel tofino2–a 12.9 tbps p4-programmable ethernet switch. In *2020 IEEE Hot Chips 32 Symposium (HCS)*. IEEE Computer Society, 1–32.

[2] Marcel Blöcher, Lin Wang, Patrick Eugster, and Max Schmidt. 2021. Switches for HIRE: resource scheduling for data center in-network computing. In *ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021*, Tim Sherwood, Emery D. Berger, and Christos Kozyrakis (Eds.). ACM, 268–285. https://doi.org/10.1145/3445814.3446760

[3] Intel Corporation. [n. d.]. Intel Tofino 2. ([n. d.]). https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.html Accessed on 21.9.2022.

[4] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica. 2018. NetChain: Scale-Free Sub-RTT Coordination. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*, Sujata Banerjee and Srinivasan Seshan (Eds.). USENIX Association, 35–49.

[5] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. 2017. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. ACM, 121–136. https://doi.org/10.1145/3132747.3132764

[6] George Karlos, Henri E. Bal, and Lin Wang. 2021. Don't You Worry 'Bout a Packet: Unified Programming for In-Network Computing. In *HotNets '21: The 20th ACM Workshop on Hot Topics in Networks, Virtual Event, United Kingdom, November 10-12, 2021*. ACM, 99–107. https://doi.org/10.1145/3484266.3487395

[7] Daehyeok Kim, Jacob Nelson, Dan R. K. Ports, Vyas Sekar, and Srinivasan Seshan. 2021. RedPlane: enabling fault-tolerant stateful in-switch applications. In *ACM SIGCOMM 2021 Conference, Virtual Event, USA, August 23-27, 2021*, Fernando A. Kuipers and Matthew C. Caesar (Eds.). ACM, 223–244. https://doi.org/10.1145/3452296.3472905

[8] Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images. (2009), 32–33. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2017. Imagenet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.

[10] ChonLam Lao, Yanfang Le, Kshiteej Mahajan, Yixi Chen, Wenfei Wu, Aditya Akella, and Michael M. Swift. 2021. ATP: In-network Aggregation for Multi-tenant Learning. In *18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021*, James Mickens and Renata Teixeira (Eds.). USENIX Association, 741–761.

[11] Seulki Lee and Shahriar Nirjon. 2020. SubFlow: A Dynamic Induced-Subgraph Strategy Toward Real-Time DNN Inference and Training. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 15–29. https://doi.org/10.1109/RTAS48715.2020.00-20

[12] Dan R. K. Ports and Jacob Nelson. 2019. When Should The Network Be The Computer?. In *Proceedings of the Workshop on Hot Topics in Operating Systems, HotOS 2019, Bertinoro, Italy, May 13-15, 2019*. ACM, 209–215. https://doi.org/10.1145/3317550.3321439

[13] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan R. K. Ports, and Peter Richtárik. 2021. Scaling Distributed Machine Learning with In-Network Aggregation. In *18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021*, James Mickens and Renata Teixeira (Eds.). USENIX Association, 785–808.

[14] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. 2019. Nexus: a GPU cluster engine for accelerating DNN-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019*, Tim Brecht and Carey Williamson (Eds.). ACM, 322–337. https://doi.org/10.1145/3341301.3359658

[15] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, et al. 2015. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. *ACM SIGCOMM computer communication review* 45, 4 (2015), 183–197.

[16] Giuseppe Siracusano, Salvator Galea, Davide Sanvito, Mohammad Malekzadeh, Gianni Antichi, Paolo Costa, Hamed Haddadi, and Roberto Bifulco. 2022. Re-architecting Traffic Analysis with Neural Network Interface Cards. In *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, April 4-6, 2022*. USENIX Association, 513–533.

[17] Tushar Swamy, Alexander Rucker, Muhammad Shahbaz, Ishan Gaur, and Kunle Olukotun. 2022. Taurus: a data plane architecture for per-packet ML. In *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022*, Babak Falsafi, Michael Ferdman, Shan Lu, and Thomas F. Wenisch (Eds.). ACM, 1099–1114. https://doi.org/10.1145/3503222.3507726

[18] Muhammad Tirmazi, Ran Ben Basat, Jiaqi Gao, and Minlan Yu. 2020. Cheetah: Accelerating Database Queries with Switch Pruning. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 2407–2422. https://doi.org/10.1145/3318464.3389698

[19] Zhaoqi Xiong and Noa Zilberman. 2019. Do Switches Dream of Machine Learning?: Toward In-Network Classification. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks, HotNets 2019, Princeton, NJ, USA, November 13-15, 2019*. ACM, 25–33. https://doi.org/10.1145/3365609.3365864

[20] Yifan Yuan, Omar Alama, Amedeo Sapio, Jiawei Fei, Jacob Nelson, Dan R. K. Ports, Marco Canini, and Nam Sung Kim. 2022. Unlocking the Power of Inline Floating-Point Operations on Programmable Switches. In *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, April 4-6, 2022*. USENIX Association, 683–700.