# *IN3*: A Framework for In-Network Computation of Neural Networks in the Programmable Data Plane

Xiaoquan Zhang, Lin Cui, *Member, IEEE,* Fung Po Tso, *Senior Member, IEEE,* Wenzhi Li, and Weijia Jia, *Fellow, IEEE*

***Abstract*—Neural networks have been widely used in networking applications due to their high accuracy and generalization. However, the traditional approach of collecting network features from switches and transmitting them to the controller introduces high traffic overhead and extra communication latency. In-network computing (INC) mitigates this issue by running computing tasks directly in the networks on the data paths using programmable data planes (PDP). However, it is challenging to embed more sophisticated computing tasks, such as neural networks, in the networks due to the limitations in the computation and storage resources of PDP. To address this challenge, we propose *IN3*, a framework that enables neural network inference completely in PDP. *IN3* uses model compression techniques to reduce the memory and computational requirements of given neural networks. Additionally, a purposely designed data plane pipeline for per-flow features computation and inference is proposed. We implemented a testbed prototype (based on Intel Tofino ASIC) and experimental results demonstrate that *IN3* effectively reduces memory usage while significantly decreasing the inference time. *IN3* demonstrates the feasibility of implementing neural networks in PDP and we identify potential future research directions for this issue.**

***Index Terms*—P4, In-network computing, Neural networks, Deep compression, NAS**

## I. Introduction

The advancement of PDP has enabled the network to process packets at line rates and to be deeply programmable, allowing many meaningful network functions to be implemented on the data plane. This paves the way for a new computing paradigm, called INC (In-Network Computing). This paradigm empowers distributed network components to partially or entirely execute various computing tasks traditionally delegated to general-purpose servers, enabling their completion along data traversal paths [1]. Most PDPs adopt the Protocol-independent switch architecture (PISA), which is a well-known hardware pipeline architecture used in Intel Tofino ASIC [2], enabling the processing of packets at more than 6.5Tbps. However, to maintain high performance, PISA only provides simple computation operations (e.g., add, subtract, and bit shift) and limited storage resources (e.g., 10Mb SRAM per stage).

Xiaoquan Zhang, Lin Cui, and Wenzhi Li are with the Department of Computer Science, Jinan University, Guangzhou, 510632. China. Email: zhangxiaoquan547@gmail.com, tcuilin@jnu.edu.cn, spaciousli2025@gmail.com.

Fung Po Tso is with the Department of Computer Science, Loughborough University, UK. Email: p.tso@lboro.ac.uk.

Weijia Jia is with BNU-UIC Institute of Artificial Intelligence and Future Networks, Beijing Normal University (BNU Zhuhai) and BNU-HKBU United International College, Zhuhai, China. Email: jiawj@uic.edu.cn.

Corresponding author: Lin Cui

Meanwhile, machine learning (ML) techniques are widely employed in networks to enhance the accuracy of tasks such as flow classification, network anomaly detection, and traffic routing. Among these techniques, neural networks are the most fundamental and widely used algorithm because of their practical properties, including high accuracy, versatility, and scalability. This makes them suitable for modeling complex network problems. In addition, neural networks can scale to handle large and complex input datasets, for which decision trees can become computationally expensive and difficult to manage as the dataset size grows [3].

However, neural networks are usually implemented in servers or controllers due to complicated computation. This inevitably produces additional communication traffic and incurs latency between switches and servers/controllers for collecting network information. The existing literature has discussed the possibility of deploying neural networks directly in the data plane [4], [5]. However, the expensive computational and storage requirements of neural networks pose scalability challenges under the resource constraints in the data plane. As a result, simpler structured binary neural networks are used as alternatives [6].

This paper considers and leverages the INC paradigm to accelerate neural network-based network functions. However, there are two main challenges to deploying neural network inference entirely in the data plane. First, the pipeline does not provide fundamental operations required for neural network inference, such as matrix multiplication and addition. Additionally, implementing floating-point multiplication and addition in the pipeline requires a significant amount of additional resources. Second, the limited resources in the pipeline make it difficult to deploy large neural network models. Specifically, limited memory resources directly affect the storage of model parameters, flow features, and intermediate variables. And the computation complexity of the neural model is correlated with the inference time in the pipeline.

We overcome these two challenges by employing neural model compression techniques to decrease the sizes of neural networks. This reduces the computation and memory resources required for embedding the entire neural network for inference on the data plane, while keeping a high inference accuracy. This forms the foundation of our *IN3* (**I**n-**N**etwork **N**eural **N**etwork) framework. *IN3* uses the Neural Architecture Search (NAS) algorithm to search for optimal structures of neural networks and compression algorithms to compress memory usage and prune unnecessary neurons. Furthermore, *IN3* proposed a pipeline design for feature statistics and inference by transfer-

ring complex computations to multiple basic operations. *IN3* automatically adapts and allocates pipeline resources for the implementation of network feature collections and different components of neural networks in the data plane. To the best of our knowledge, *IN3* is the first study that considers the use of NAS and compression to fit neural networks into PISA hardware switches.

The rest of the paper is organized as follows. Section II gives background and related works. Section III introduces the architecture of *IN3*. Section IV conducts experiments and analyzes their results. Section V envisions future research directions and Section VI concludes this work.

## II. Backgrounds

### A. ML-based method on PISA

PISA evolved from the well-known RMT (Reconfigurable Match Tables) architecture to be the widespread packet processing model for hardware switches (e.g., Intel Tofino ASIC [2]). It allows users to design applications on demand using a high-level language, namely P4 language (Programming Protocol-independent Packet Processors). The pipeline is the key in PISA for customizing network functionality by manipulating match-action tables (MAT). It imposes two constraints on programmability: computation and storage. First, to guarantee line rate speed, complex operations are not offered (e.g., multiplication/division or floating operations). Limited numbers of ALUs (Arithmetic Logic Units) per stage and no loop logic also limit the complexity of network functionality. Second, limited storage capacity (e.g., SRAM) confines the performance of network functions.

Previous works have implemented partial or full offloading of ML algorithms in PISA (e.g., decision trees and SVM) [3], [7]. Due to the aforementioned constraints, the inference procedures of these ML algorithms are generally transferred to basic operations supported in PISA via well-designed encoding strategies. For example, a decision tree can be mapped into multiple MATs by perfectly splitting the feature spaces. In comparison, implementing neural networks in PISA is much more challenging. First, the large number of weights consumes considerable storage resources that can easily exceed the capacity of the pipeline. For example, the VGG-16 model is over 4,000 Mb while the hardware pipeline of Tofino 1 only provides an average of 10Mb within each stage (one pipeline has 12 stages). Second, the design goal of PISA does not fit for such a large number of floating computations in the neural network. For example, to provide accuracy, a well-designed floating computation can almost exhaust the resources of the whole hardware pipeline [4]. Third, to execute matrix manipulation, the recirculation operation is used to achieve loops in matrix multiplication by enforcing the packet entering the pipeline multiple times, which will lead to a dramatic decrease in throughput and increased latency when the number of loops increases.

### B. Neural architecture search and deep compression

Given a constrained resource budget on the target hardware, elaborating the design of the neural network architecture for optimal performance requires a significant amount of expertise and experimentation. NAS automatically generates neural network architectures to optimize performance for a given task [8]. It works by searching a large space of possible network architectures, evaluating each of those performances, and then using this information to guide the search towards more promising architectures. One of the advantages of NAS is that it can save a significant amount of time and effort in the design process, especially when the search space is large and complex.

Deep compression is a technique for reducing the size of neural networks to improve their efficiency and inference speed [9]. Pruning is a well-known compression technique to reduce the number of parameters in a neural network by removing connections with small weights or by removing entire neurons that have little impact on the overall performance of the network. The idea behind pruning is to simplify the network structure and reduce the computational cost of training and inference while maintaining or even improving its performance. Another efficient technique is quantization, which represents the network's parameters and activations with a lower precision data format to reduce storage requirements, making it easier to deploy on devices with limited computational resources. For example, using 8-bit integers to represent 32-bit floating-point numbers has been proven to be efficient [10].

### C. State of the art

To implement neural network-based network functions, traditional frameworks employ a controller that collects and utilizes the feature information of each flow from packets for neural network inference. However, limited bandwidth struggles to cope with the challenge of today's increasing traffic, and frequent feature computation and network inference impose high CPU performance demands. Leveraging the programmability of PDP to offload the inference tasks onto the data plane can preclude the bottleneck problem. To achieve this, research has provisioned reconstruction of the PISA architecture [4], [11]. Taurus [4] has attempted to add a complex computation block, MapReduce, in PISA to support data-parallelism common in modern ML applications such as SVM and neural networks. IOI [11] has provisioned optical matrix multiplication in the PDP hardware switch to provide low-latency machine learning inference. However, they struggle to make a trade-off between the size of neural networks and performance. Moreover, it is inevitable to consider the issue of design cost and production cycle of new chips.

Razavi et al. [5] have discussed using data planes as inference nodes for computer vision inference tasks. However, due to the resource limitations of switches, even for a mini-version of a neural network, they should employ multiple hardware switches to save models and compute, and each individual packet needs to be processed in the pipeline nearly a million times. This results in severe performance degradation and increased latency.

In this paper, we investigate the open problem of implementing neural networks in PISA-based hardware switches

TABLE I
STATE OF THE ART

| Framework | Target architecture | Functionality | Model size | Inference position | Solutions for line rate performance |
|---|---|---|---|---|---|
| Controller/ servers-based | PISA switches & controllers. | Neural network-based network functions. | Any. | Control plane. | Sampling or increasing bandwidth. |
| Taurus [4] | Additional FPGA inference module. | DNN-based network functions. | Tradeoff between inference performance and model size. | Data plane. | A well-designed MapReduce module achieved 10Gbps throughput. |
| IOI [11] | New optical architecture. | DNN-based network functions. | Unknown. | Data plane. | Unknown |
| Razavi et al. [5] | PISA switches. | CV CNN-task inference. | Mini-version of AlexNet. | Data plane. | A large amount of recirculation distributed among multiple switches. |
| *IN3* (Ours) | PISA switches. | CNN/DNN-based network functions. | Compress large input models under pipeline restrictions. | Data plane. | Using compression techniques to reduce recirculation. |

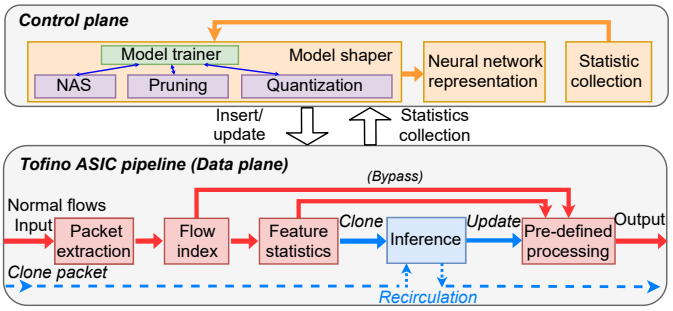


Fig. 1. Architecture of *IN3*.

(Table I). We believe that the utilization of model compression techniques for neural networks implemented in PDP can help to solve issues in the aforementioned works. Given a training model, model compression techniques are to reduce their size to meet the resource constraints of the P4 pipeline. Moreover, the current deployment of P4 applications tends to lean towards multi-application deployments to avoid wasting switch resources, and model compression techniques can effectively reduce the deployment costs of P4 applications. Benefitting from techniques of model compression, *IN3* can receive a large size of the neural network as input to automatically generate the optimal model fitting to implement in the PISA hardware switch.

## III. Design

### A. Overview

As shown in Fig. 1, the architecture of *IN3* is designed based on two major processing units: the control plane and the hardware pipeline in Tofino ASIC (data plane). The control plane is mainly responsible for generating an optimal neural network and inserting/updating it into the pipeline. Its workflow is as follows: the *Model trainer* first gathers historical traffic data and initiates neural network training, and the *Model shaper* shapes the network structure with a minimal size (which can be fit into the hardware pipeline) while keeping a high inference accuracy, then the module of *Neural network representation* transforms the neural network into MAT format. To periodically update the neural network, the control plane consistently collects flow statistics from the pipeline as historical data and retrains it.

In the data plane, there are two different workflows, i.e., feature statistics for network flows (red arrow) and neural network inference (blue arrow). In the first workflow, the pipeline extracts headers from the network packet to index its corresponding flow. Based on the header information, the pipeline determines whether the packet bypasses (e.g., flows having been predicted or in the white list) or updates its flow features. After the feature update, if the packet triggers inference, this packet is cloned for processing in the second workflow, and the original packet will be output from the pipeline normally. In the second workflow, the computation components of inference are achieved by the basic operations offered by the pipeline (e.g., different layers and the activation function). Moreover, *IN3* uses *Recirculation* operation to force the cloned packet to continuously loop in the pipeline for implementing multiple operations of multiply and addition. After the completion of the inference, its result will be saved to direct how to process the corresponding flow.

### B. Design goal

According to the paradigm of INC, the design goal of *IN3* has two aspects:

*First, the whole neural network inference is offloaded to the data plane*. To avoid unnecessary traffic overhead between switches and controllers (or servers), *IN3* enables neural network inference entirely in the data plane without the involvement of the controller or servers. To cope with the constraints in the pipeline, *IN3* only uses basic operations to achieve different types of neural network layers and the techniques of NAS and deep compression to minimize the size of the model while maintaining accuracy.

*Second, the impact of recirculation on the line-rate forwarding performance is minimized. IN3* uses compression techniques to significantly reduce the amount of necessary recirculation. It only performs recirculation on cloned packets for inference, while packets of normal traffic can bypass the inference and be processed at line-rate, thereby avoiding performance degradation on throughput and latency.

### C. Training in model shaper

A neural network is primarily determined by its network structure and parameters, given that we can evaluate the total computation and storage of the neural network. Specifically, in

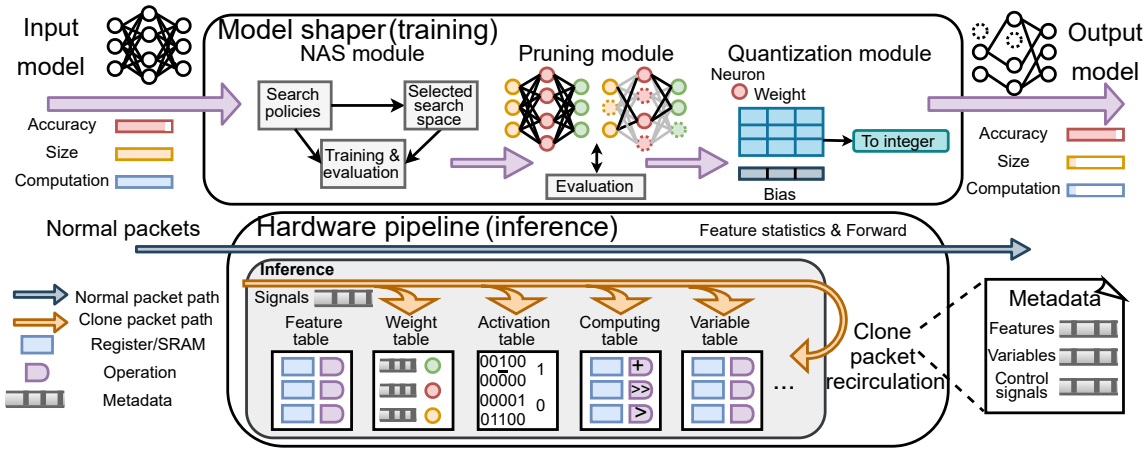


Fig. 2. Design of *IN3*, including the training phrase in *Model shaper* and the inference phrase in the hardware pipeline.

the procedure of inference, trained parameters need to be saved in storage, and the network structure informs which neurons need to pass through so that the amount of required computation can be calculated. The module of *Model shaper* (as shown in Fig. 2), including NAS, pruning, and quantization, analyzes and reduces the amount of requisite storage and computation of the neural network, thus fitting resource constraints in the hardware pipeline.

The search space in NAS is used to define the set of feasible network structures to explore the optimal search results, which includes the number of layers and nodes in each layer, convolution kernel size, pooling operations, activation functions, and so on. With the objective of searching for optimal structures of neural networks, *IN3* automatically generates the parameters of neural network layers and dropout in each layer as the search space. Next, a search strategy is appointed to explore the search space to find the optimal neural network structure by adjusting its policies for parameter selection in the next iteration.

The pruning module then further removes unnecessary neurons and connections. It has two-folds benefits. First, it can directly reduce the amount of resource requirements for storage and computation in the pipeline, making *IN3* avoid the cooperation of multiple switches and performance degradation [5]. Second, it maintains the accuracy of the original neural network model. *IN3* sets a compression rate as the parameter in pruning, which refers to the percentage of neurons removed.

Lastly, since the float-point operations are not supported in the pipeline, *IN3* applies the quantization module that enables the process of reducing the precision of the weights and activations of the neural network from floating-point values (e.g., FP32) to lower-precision fixed-point values (e.g. INT8). Feature statistics and inference can be directly achieved through basic operations the pipeline provides. As a result, the complexity of neural network inference reduces drastically while keeping a high inference accuracy.

After completing model compression, *IN3* selects an appropriate neural network model based on the current deployment situation. Specifically, it estimates the maximum model size based on the remaining available resources in the pipeline and chooses the model with the highest accuracy from those models that meet this condition to compile into the pipeline.

### D. Inference in the hardware pipeline

Inference in the hardware pipeline is comprised of two processes, i.e., network statistic of flow features and neural network inference (shown in Fig. 2). In the first process, the hash function is used as the flow identifier to track individual flows via packet headers (e.g., IP src/dst address, TCP src/dst port, and protocol). Utilizing simple operations the pipeline offers operations such as bit shifts, add, and condition statements, *IN3* can catch and calculate flow features (e.g., max/min/total packet length and port number) [6]. However, payload is not considered as part of feature statistics currently.

In the second process, the neural network is mapped in the pipeline via MATs for computation and metadata for logic control. Specifically, *IN3* uses weight tables to save weight parameters and computing tables to execute multiplication and addiction. Since the hardware pipeline does not support multiplication, *IN3* deduces fixed-point multiplication to multiple operations of bit shifts within multiple stages [5]. Besides, the computing tables offer comparison operations to achieve the max pool layer. The activation table checks the most significant bit for achieving the Relu function. A well-designed table-stacking approach for user-customized neural network structures can reduce stage usage, thereby decreasing the utilization of recirculation.

The workflow of the packet process in the pipeline is shown in Fig. 2. A packet of a flow first enters the pipeline and the pipeline updates its flow features based on its headers. If the packet triggers neural network inference (e.g., the $i$-th packet of a flow), the pipeline clones a packet (dashed arrow in Fig. 2) while ending the process of the packet and forwards it (solid arrow) without any effect on the normal traffic; otherwise, it is forwarded normally (solid arrow). In the inference process, the cloned packet is responsible for executing the inference computation. It attaches three types of metadata, including input features that are computed in the first process, variables that save the operand of multiplication and addition, and

control signals. *IN3* uses the *Recirculation* operation to feed the cloned packet back to the pipeline recurrently. The benefit of this approach is to achieve complex computation of dense or conventional layers by decomposing the matrix operations that cannot be completed in a one-time process of the pipeline into multiplication and addition operations across the pipeline process multiple times.

*IN3* also designs a *Recirculation ID* in control signals metadata to identify the corresponding neural network layer computations that the cloned packet should undergo when it enters the pipeline for the $i_{th}$ time. Specifically, the packet reads a parameter of a dense layer from registers and conducts multiplication with an operand in the variables metadata, whose result will be saved in the variable registers for the use in the next iterations. After the completion of the inference, the inference result of the flow is updated and the cloned packet is dropped. Subsequent packets of the flow will execute actions based on the inference result.

## IV. Proof of Concept and Performance Evaluation

We implemented a hardware testbed equipped with a P4 hardware switch (using an Intel Tofino ASIC [2]) where each stage comprises 10 Mbits of SRAM and 8 stateful ALUs for register read and write operations. We employed a random search strategy for the NAS algorithm and the autoML toolkit *NNI* to train neural networks on an NVIDIA GeForce RTX 2080 Ti.

### A. Use case

We utilized two datasets: CIC-IDS2017 (dataset #1) [12] which labels benign and various common attacks on network traffic, and a traffic classification dataset (dataset #2) [13] encompassing diverse network flow types. For inference tasks, we designed an original neural network (baseline model) that assembles three convolutional layers (Conv1d), pool layers (MaxPooling1D), three full connection layers (Dense), and ReLU activation functions. No more than 10 flow features, which can be collected and computed in the data plane, are selected for inference. In the *Model shaper*, we set adjustable parameters range of the size of layers to [16, 32, 64] for NAS. The rate of pruning (e.g., scale down the size of the network proportionally) is set to [0.2, 0.5, 0.7, 0.8]. In quantization, we used INT8 to transform neural models. For comparison, we trained the original neural network without compression as the baseline (red dashed line in Fig. 3).

### B. Experimental results

**Performance of model compression.** Fig. 3 is conducted to show the balance between accuracy and compression rate. On dataset #1, the accuracy of some models remains almost the same as the original (86.4%) even when their size is reduced to 70%. Additionally, in some models, when the size is reduced by more than 95%, the accuracy only drops by 1%. This phenomenon has a crucial impact on driving the deployment of larger neural networks. Additionally, despite the differing tasks of the two datasets, they both exhibit similar results.

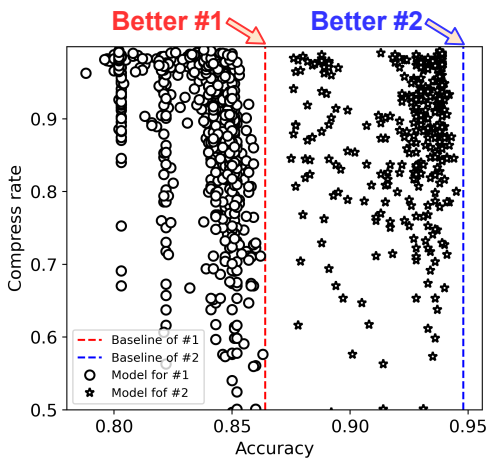


Fig. 3. *Model shaper* balances two metrics, accuracy and compression rate. The baseline (dashed line) represents the accuracy of the original neural network, and the circle represents compressed the neural network model. A better model means achieving high compress rates and accuracy.

This demonstrates the generalizability of the compression techniques across diverse network inference tasks.

**Evaluation in the P4 hardware switch.** We evaluated memory consumption for the three critical tables: feature table, weight table, and variable table. In Fig. 4, memory consumption effectively decreases after the process of *Model shaper*. Apart from the reduction in neural network size and intermediate variable requirements due to the reduction of the number of weight parameters, the storage of quantized features also decreases from 32 bits to 8 bits, leading to an overall decrease in memory consumption. Since SRAM allocation in the pipeline is in blocks, the model has a similar memory consumption after compression.

The greatest advantage of model compression is reflected in dramatically reducing the computational requirements during neural network inference. We use MAC (Multiply Addition Computation) as the metric to measure the total computation, which represents the total number of multiplications and additions required during inference. When matrix operations require multiple multiplications and additions, the inference packets need to be recirculated to the pipeline multiple times. Therefore, reducing the MAC count can reduce the inference time of neural networks in the data plane effectively and directly. From Fig. 4, it can be observed that as the pruning rate increases, the computational workload of the neural network decreases. When the pruning rate reaches 0.8 (nearly the compression rate of over 90%), the MAC count of the original model over the convolutional and dense layer is decreased by 98% and 96%. This provides a promising prospect for reducing the inference time of neural networks in the data plane.

We also evaluated the inference time of neural networks within the hardware pipeline. As shown in Figure 5, the inference time decreases alongside the pruning rate. This occurs because a model with a higher pruning rate requires fewer MACs, consequently reducing the number of necessary recirculations for the cloned packet. When the pruning rate equals 0.8 (around 1000 of recirculations), the inference

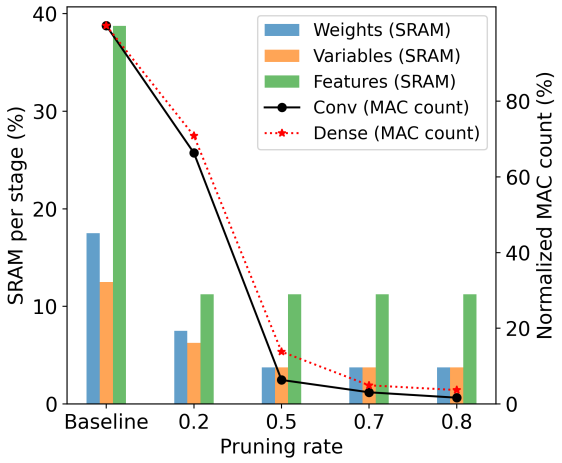


Fig. 4. Comparison of memory usage and requisite MAC count.

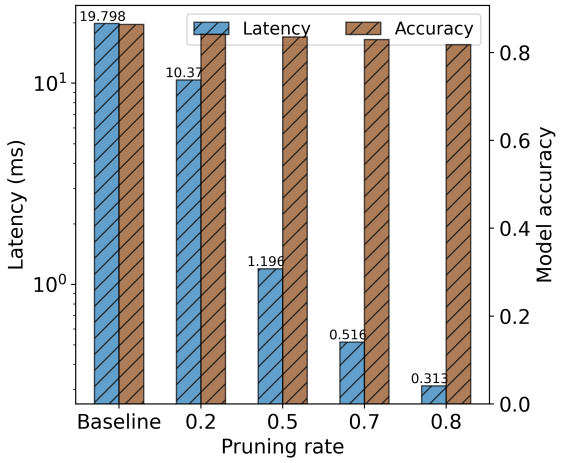


Fig. 5. Comparison of inference latency and accuracy.

time is only 313*us*, which is faster than the uncompressed model (i.e., rate=0, around 40000 of recirculations) by 63x. Meanwhile, from models on dataset #1, we observed that the average accuracy experiences only a slight decrease with the increase in pruning rate, due to the similar reasons above. In comparison, *IN3* provides a faster inference time than an SDN controller in data center (e.g., 10*ms* of average service response time [14]), and obtains more inference time but larger and richer models than other simple models like decision trees and binary neural network [6], [7].

## V. Future Research Directions

This work is our initial step in utilizing model compress techniques to accelerate inference entirely in the data plane. Several interesting research directions are worth pursuing.

**Neural network update.** The neural network's weight parameters are stored in a MAT, allowing the controller to dynamically reassign values during operation and facilitate periodic network updates. However, this method may fail when the network structure or layers change, requiring the redesign of a new table-stacking for optimized inference and resource usage. Despite programmable switches offering fast reconfiguration for deploying new P4 programs within a short timeframe (e.g., 20ms), registers' states will be reset during reconfiguration. Addressing this issue is practical, as network model changes are often essential.

**Pipeline optimization.** Our preliminary work has already demonstrated the effectiveness of compression techniques in reducing recirculation count. Further, optimizing neural network inference in the pipeline can further decrease recirculation demands. Existing research has proposed various methods to mitigate recirculation's negative impact on line rate performance [15]. Therefore, further pipeline optimization for a given neural network structure remains highly worthwhile. Moreover, implementing more complex layers and activation functions is valuable as they can capture intricate relationships and enhance generalizability. In future work, we will explore transferring the Softmax function into match-action operations.

**Scalability.** A feasible solution for the scalability of neural networks is to aggregate resources within multiple pipelines (or switches) in case that a single pipeline (or switch) becomes the bottleneck of resources or throughput. Thus, storage and computation burdens can be distracted in multiple computing components. Separating different logic functions into distinct computing entities allows users to optimize resource utilization. For example, decoupling feature statistics and neural network inference mitigate dependency conflicts arising from shared resource usage, thereby simplifying deployment. Similarly, partitioning large neural networks across multiple entities accommodates resource constraints. However, this approach necessitates a mechanism to coordinate computation across these decentralized components.

## VI. Conclusion

This paper proposed a framework for supporting the in-network computation of neural network inference in the data plane. The NAS algorithm and model compression techniques are first employed to reduce the model size of the neural network while ensuring the model's accuracy is not compromised. On the hardware pipeline, we then design feature computations and support inference calculations for different layers of the neural network. Experimental results demonstrate that this work effectively reduces the memory and inference time requirements.

## Acknowledgments

This work is partially supported by National Natural Science Foundation of China (NSFC) under Grant 62172189; the Outstanding Innovative Talents Cultivation Funded Programs for Doctoral Students of Jinan University.

## References

[1] N. Hu, Z. Tian, X. Du, and M. Guizani, "An energy-efficient in-network computing paradigm for 6G," *IEEE Transactions on Green Communications and Networking*, vol. 5, no. 4, pp. 1722–1733, 2021.

[2] "Intel Tofino switch ASIC," https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html, accessed on: Jan. 20, 2024.

[3] X. Zhang, L. Cui, F. P. Tso, and W. Jia, "pHeavy: Predicting heavy flows in the programmable data plane," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4353–4364, 2021.
[4] T. Swamy, A. Rucker, M. Shahbaz, I. Gaur, and K. Olukotun, "Taurus: a data plane architecture for per-packet ML," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 1099–1114.
[5] K. Razavi, G. Karlos, V. Nigade, M. Mühlhäuser, and L. Wang, "Distributed DNN serving in the network data plane," in *Proceedings of the 5th International Workshop on P4 in Europe*, 2022, pp. 67–70.
[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*, 2022, pp. 513–533.
[7] C. Zheng, M. Zang, X. Hong, R. Bensoussane, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "Automating in-network machine learning," *arXiv preprint arXiv:2205.08824*, 2022.
[8] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE transactions on neural networks and learning systems*, 2021.
[9] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.
[10] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying IoT devices in smart environments using network traffic characteristics," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, 2018.
[11] Z. Zhong, W. Wang, M. Ghobadi, A. Sludds, R. Hamerly, L. Bernstein, and D. Englund, "IOI: In-network optical inference," in *Proceedings of the ACM SIGCOMM Workshop on Optical Systems*, 2021, pp. 18–22.
[12] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *ICISSp*, vol. 1, pp. 108–116, 2018.
[13] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2005, pp. 50–60.
[14] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan, "Measuring control plane latency in SDN-enabled switches," in *Proceedings of the 1st ACM SIGCOMM symposium on software defined networking research*, 2015, pp. 1–6.
[15] N. Ivkin, Z. Yu, V. Braverman, and X. Jin, "Qpipe: Quantiles sketch fully in the data plane," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, 2019, pp. 285–291.

**Xiaoquan Zhang** received the M.Eng. from Jinan University, China, in 2021. He is currently working toward the Ph.D. degree in Jinan University. His current research interests include programmable data planes, software-defined networking, and machine learning in computer networks.

**Lin Cui** is currently a professor in the Department of Computer Science at Jinan University, Guangzhou, China. He received the Ph.D. degree from City University of Hong Kong in 2013. He has broad interests in networking systems, with focuses on software defined networking (SDN), programmable networking, NFV, cloud networking, distributed systems and so on.

**Fung Po Tso** received his B.Eng., M.Phil. and Ph.D. degrees from City University of Hong Kong in 2006, 2007 and 2011 respectively. He is currently a Reader in the Department of Computer Science at the Loughborough University. His research interests include: network policy management, network measurement and optimisation, service chaining, network programmability, in network computation and edge computing.

**Wenzhi Li** is currently a postgraduate student in Jinan University. His current research interests include programmable data planes, software-defined networking, and machine learning in computer networks.

**Weijia Jia** is currently a Chair Professor and Director of BNU-UIC Institute of Artificial Intelligence and future Networks, VP for Research of BNU-HKBU United International College. His research interests include smart city, IoT, knowledge graph constructions, multicast and anycast QoS routing protocols, wireless sensor networks, and distributed systems. He has over 500 publications in prestigious international journals/conferences and research books and book chapters.