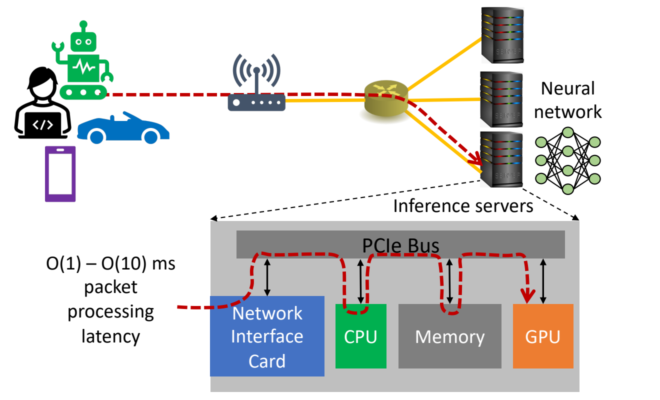
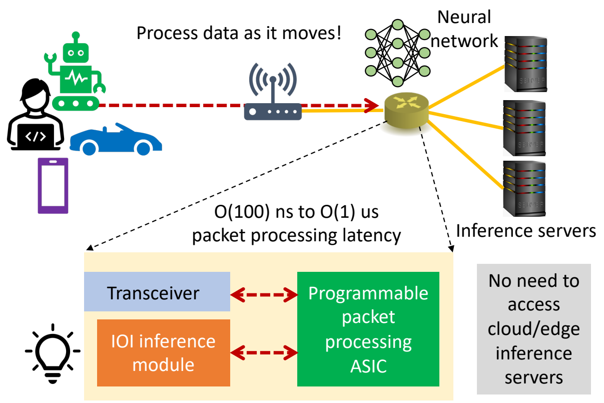
IOI: In-network Optical Inference

Zhizhen Zhong, Weiyang Wang, Manya Ghobadi,
Alexander Sludds, Ryan Hamerly, Liane Bernstein, Dirk Englund
Massachusetts Institute of Technology



(a) Today's cloud-based or edge-based machine learning inference on inference servers.



(b) IOI-based machine learning inference on programmable switches.

Figure 1: Illustrative comparison of IOI and today's cloud/edge-server-based machine learning inference.

ABSTRACT

We present In-network Optical Inference (IOI), a system providing low-latency machine learning inference by leveraging programmable switches and optical matrix multiplication. IOI consists of a novel transceiver module designed specifically to perform linear operations such as matrix multiplication in the optical domain. IOI's transceivers are plugged into programmable switches to perform non-linear activation and respond to inference queries. We demonstrate how to process inference queries inside the network, without the need to send the queries to cloud or edge inference servers, thus significantly reducing end-to-end inference latency experienced by users. We believe IOI is the next frontier for exploring real-time machine learning systems and opens up exciting new opportunities for low-latency in-network inference.

CCS CONCEPTS

• **Networks** → **In-network processing**; **Programmable networks**; *Network protocol design*; • **Hardware** → **Emerging optical and photonic technologies**;

KEYWORDS

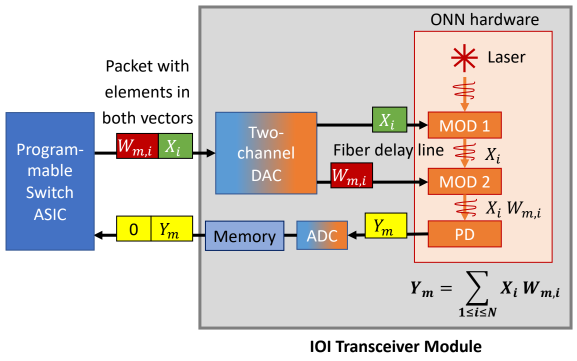
In-network computing, Edge computing, Optical neural networks, Programmable switches, Machine learning inference

ACM Reference Format:
Zhizhen Zhong, Weiyang Wang, Manya Ghobadi, Alexander Sludds, Ryan Hamerly, Liane Bernstein, Dirk Englund. 2021. IOI: In-network Optical Inference. In *ACM SIGCOMM 2021 Workshop on Optical Systems (OptSys '21), August 23, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3473938.3474508

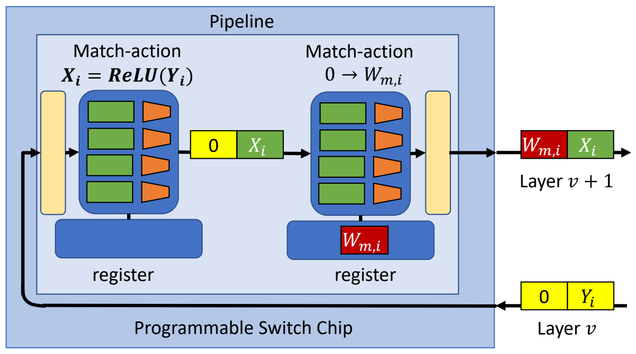
1 INTRODUCTION

Artificial Intelligence (AI) is reshaping our society by empowering novel services such as self-driving vehicles (e.g., Waymo [3]), smart personal assistants (e.g., Alexa [9]), virtual/augmented reality (e.g., Oculus [10]), and personalized recommendations [4, 11]. The success of AI-based services depends on the latency perceived by its users launching inference tasks to interact with the services. To maintain low latency, neural network models should be located as close as possible to the user, ideally on the user's device. However, to achieve high-accuracy, today's services often require large Deep Neural Networks (DNNs) that are constantly updated with new datasets. Despite research efforts to reduce the size of DNN models [7, 8], most commercial applications [9, 13] still execute inference tasks in the cloud or on edge servers instead of locally due to memory/power/computing limitations [15].

In cloud-based or edge-based machine learning inference tasks, data are sent from users' devices all the way to an inference server that hosts the DNN on a GPU/TPU, as depicted in Figure 1a. The



(a) IOI's optical vector-vector product



(b) Nonlinear activation at programmable switches

Figure 2: Synchronizing a neural network's input and weight by encapsulating them into the same data packet.

inference latency includes two components: 1) packet propagation and queueing delay on the network datapath, and 2) packet processing delay at the end-host inference server(s). The first component depends on the length of the datapath between users and the inference server; hence, the closer the server the better. The second component adds a significant amount of latency to inference queries that has been mostly ignored. To ensure reliable delivery, inference servers have to rely on TCP sockets and the host server's CPU computing capabilities to process the arriving inference queries. Given the clock frequency of today's CPUs (e.g., Intel Xeon Gold 6152 processor with 44 cores), one CPU can only process packets at about 5.2 million packets per second (PPS) per core [12] or 220 million packets per second (PPS) in total. We argue this step is unnecessary and propose performing the inference task *inside network switches* by leveraging programmable switch ASICs, such as Intel Tofino [2] or Juniper Networks Junos Trio chipset [1]. For instance, Intel Tofino can achieve 6 billion PPS packet processing throughput with a maximum port bandwidth of 12.8 Tbps. This is 1000× faster than a CPU core!

Although in-network inference with programmable switches has fundamental latency advantages over the conventional server-based solutions, each inference task requires intensive computation such as matrix multiplication, which is challenging in today's switches because of their limited resources [5, 16]. Therefore, we propose a novel architecture called In-network Optical Inference (IOI) to leverages emerging optical matrix multiplication hardware [6] using commercially available modulators and photodetectors that already exist in today's transceivers.

Figure 1b illustrates IOI's approach to enable today's switches with neural network inference at line rate. In IOI, the packet processing bottleneck induced by the inference server's CPU is lifted by the high-throughput packet processing ASIC of network switches. IOI transforms the conventional inference path from **user** → **network** → **inference server** → **network** → **user** into **user** → **switch** → **user**, thus significantly reduces the end-to-end inference latency.

2 IOI DESIGN

We propose IOI as a next-generation inference system using a novel transceiver module with optical computation capabilities, combined with the power of programmable switches, for high-efficiency line-rate inference query processing. Specifically, matrix multiplication is done with specially designed "IOI's transceiver modules" in the optical domain (§2.1), and non-linear activation is performed by programmable switch ASICs [2] in the electrical domain (§2.2). Multiple layers of the neural network are processed with scheduled recirculations using programmable switches pipelines (§2.3), and a novel packet structure and a protocol are proposed to enable efficient communications between users and IOI hardware (§2.4).

2.1 Optical Matrix Multiplication at IOI

Prior work has demonstrated the feasibility of Optical Neural Networks (ONNs) to perform linear algebra computations in the optical domain with the photoelectric effect under time/wavelength multiplexing [6]. Compared with electrical computing systems that carry out computation with binary logic electrical circuits on CMOS, optical systems have intrinsic energy and throughput advantages. This is because the computation is done in the analog domain using photons and, hence, ONNs have much lower latency and energy consumption. IOI leverages the ONN hardware to perform matrix multiplications entirely in the optical domain instead of the electrical domain. As shown in Fig. 2a, a vector-vector product $Y_m = \sum_{1\le i\le N} X_i \cdot W_{m,i}$ in IOI is performed by encoding the N-dimensional input $\mathbf{X} = [X_i]$ and $\mathbf{W_m} = [W_{m,i}]$ in T time steps and S spatial channels (e.g., wavelengths) such that $N = T \cdot S$. To compute Y_m, we leverage the photoelectric effect that accumulates the product of X_i and W_i over N units at the photodetector.

Handling frequency differences between the modulator and photodetector. Due to the T-step time integration, the operation speeds of modulators and photodetectors are different. When both modulators work at a frequency of M, the photodetector has to work at a reduced frequency of M/T to perform a T-step time integration. This effect also agrees with the nature of the vector-vector product whereby the volume of input data (two vectors, each with multiple values) is much larger than the volume of output data (a single value). Hence, the datarate that enters the ONN hardware will be faster than the exiting datarate. This creates a rate-mismatch problem for line rate processing. To address this challenge, we propose to put a memory module right after the analog-to-digital converter (ADC) so that the readout y_m from the photodetector can be temporally stored before all the y_m are calculated (refer to Fig. 2a). As soon as the entire activation vector $Y = [y_m]$ has been completed by the ONN hardware, the memory will flush the entire Y vector into the programmable switches at line rate for nonlinear activation processing and then start the computation of the next layer. The size of the memory should be sufficient to store the inter-layer activation vector, which would be equivalent to the largest number of neurons of all layers.

Hybrid time/wavelength multiplexing. Optical matrix multiplication can be done in either time or frequency domains with the same system throughput, because they are exchangeable in the Fourier space. Time-division multiplexing needs the fewest optical devices (modulator, photodetector), but it requires the entire vector to be serialized in time so that the computation time is prolonged; wavelength-division multiplexing allows encoding on parallel wavelength channels at the same time so that the integration happens in the frequency domain without the need to wait over T time steps to get a photodetector readout value at the cost of more optical devices. For matrix multiplication, having W parallel channels will result in a computation time speedup of up to W times.

Synchronizing modulators inputs with weight pickup and fiber delay line. For a vector-vector product $\sum_{1 \leq i \leq N} x_i w_{m,i}$ on IOI, the first challenge is to synchronize x_i and $w_{m,i}$ so that the coded waveforms are executed on the modulator at the same time. IOI tackles this challenge by encapsulating x_i and $w_{m,i}$ into the same packet using programmable switches and feeds this packet into the DAC, so that the analog values of x_i and $w_{m,i}$ can arrive at the second modulator (MOD 2 in the Fig. 2a) at precisely the same time. We also design a fiber delay line connecting modulators (MOD 1 and MOD 2) to compensate for the elapsed time of x_i and ensure the x_i in optical signals and $w_{m,i}$ in electrical voltages will arrive at the second modulator simultaneously. To encapsulate both x_i and $w_{m,i}$ into the same data packet, we design a *weight-pickup* mechanism, as shown in Fig. 2b. Weights of a neural network $w_{m,i}$ are stored in the registers of the programmable switches ASICs in each stage. To achieve this, we need to design the packet structure, and leverage programmable switches to generate such packets at line rate for the high-speed DACs.

2.2 Electrical Nonlinear Activation at Programmable Switches

Nonlinear computation is challenging in the optical domain [14]. Fortunately, nonlinear operations are simple in the electrical domain. For example, ReLU, a widely-used nonlinear activation function in various neural networks, can be interpreted as programmable switch match-action tables as follows:

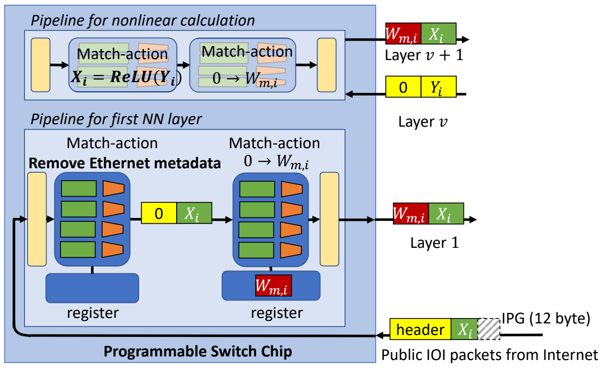


Figure 3: Packet processing for incoming packets.

$$ReLU(x) = \begin{cases} \text{null action}, & x.sign = 1 \\ \text{all bits set to zero}, & x.sign = 0 \end{cases} \quad (1)$$

ReLU on programmable switch pipeline. In IOI, we implement the nonlinear activation function inside the programmable switches pipeline with match-action logic using P4 language. As depicted in Fig. 2b, for a 32-bit or 16-bit floating number, its binary form is coded as three parts: sign (1 bit), exponent (5 bits or 8 bits) and mantissa (10 bits or 23 bits). The ReLU function can be implemented on programmable switches using P4 language to describe the match-action as follows: for incoming data x, perform "null" *action* if the x's "sign" bit *matches* 1 or "set-to-zero" *action* if the x's "sign" bit *matches* 0.

IOI packet design. To involve the programmable switches into the machine learning inference and utilize their flexible programmability, we need to carefully design the packet structure, as only packet metadata can be processed in the programmable switches' pipeline architecture. As discussed in Section 2.1, to perform matrix multiplication while satisfying the synchronization requirements of the ONN hardware, we propose to encapsulate the corresponding X_i and $W_{m,i}$ into the same packet. Fig. 3 shows the programmable switches logic for incoming data packets from users. The packets sent from users will contain Ethernet/IP information so that they can arrive at designated switches instead of host servers. The programmable switch is responsible for removing this routing information from the packet headers, and replacing it with weights $W_{i,m}$, which will be used for matrix multiplication in the following optical computation. We name these processed packets as IOI packets. Figure 4 shows the structure of a typical IOI packet. Depending on the source and destination the IOI packet, it has two variants: 1) public IOI packet and 2) private IOI packet. The public IOI packet is the data packet between user clients and the IOI system in the public Internet: its metadata part is encoded as the packet header carrying IP/Ethernet routing information and zero paddings, and its payload part is encoded as the packet payload carrying user input data or final inference results. On the other hand, the private IOI packet circulates within the programmable switch carrying activation vectors between neural network layers. Because both X_i

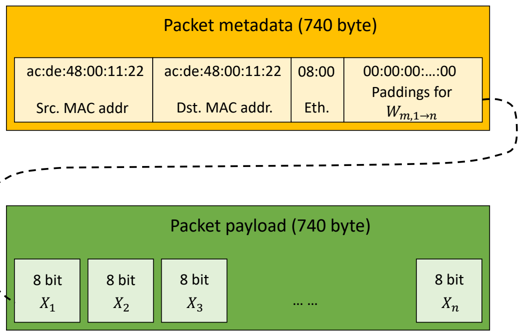


Figure 4: Public IOI packet structure, private IOI packet will put the payload into header and leave the payload empty.

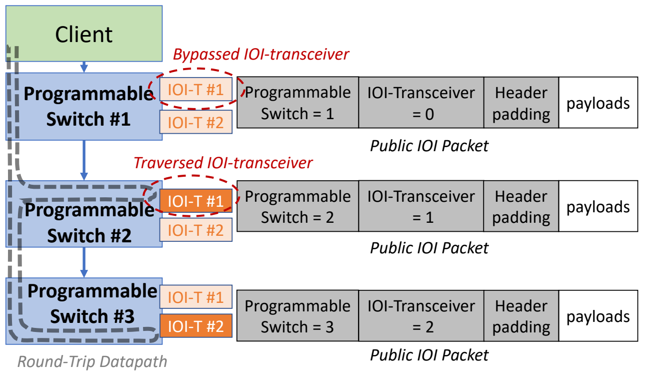


Figure 5: Routing of IOI packets.

and $W_{m,i}$ of the packet need to be processed by the programmable switch ASICs, they are both encoded as packet headers, while the payload is left empty. In the IOI packet, the size of the packet metadata is equal to the size of the payload, regardless of the Ethernet header's size (typically 14 bytes, the rest is padded with zeros) , as they will be removed in the first stage of the programmable switches pipeline. After we set the entire metadata area to be zero in the first programmable switch stage, the second stage will pick up the corresponding $W_{m,i}$ so that we get the packets ready for the first layer of the neural network. After the first layer's computation finishes, the programmable switch will follow the packet recirculation logic introduced in Figs 2a and 2b. The size of the IOI packet depends on the maximum number of weights $W_{m,i}$ that we can pick up after one pass of the programmable switch pipeline. Note that for a typical Ethernet packet, the size can be as large as 1500 bytes, which means we can encapsulate an input vector with a maximum size of 750 (coded as 8-bit integers) into the same packet.

2.3 Recirculation for Multiple Neural Network Layers

Sections 2.1 and 2.2 explain how IOI finishes matrix multiplication and nonlinear activation for one neural network layer. For a deep neural network with multiple layers, we need to recirculate these two processes and pass private IOI packets between adjacent layers to turn output vectors from the previous layer into the input of the next layer.

Provision resources for the largest layer. As we discussed in Section 2.1, the previous layer output vector Y_i is first cached in the memory, and then flushed to the programmable switch ASIC to start the computation of the next layer. During this process, we need to make sure there are sufficient resources (registers in programmable switch ASICs, modulators, and photodetectors on IOI transceivers). Therefore, we provision the resources for the largest layer of the neural network (the layer with the most neurons).

2.4 IOI Client and Protocol

IOI packet routing. To allow IOI packets to arrive at their designated IOI system, we introduce a routing protocol for IOI. As shown in Fig. 5, we assign an IP address to each IOI transceiver module and leverage the segment routing. Since different switches may store different neural networks, we need to ensure an IOI packet will only be processed at its destination switch, bypassing intermediate ones. In addition, if a neural network is so large that it cannot fit into a single switch, we want to distribute its layers into multiple switches as a chain, and leverage the segment routing to allow IOI packets to traverse these switches in a designated order.

IOI client design. IOI protocol also needs to be implemented on the client side so that when users initiate an inference task using IOI, the input data can be accurately encapsulated into the IOI packet format. For the segment routing paradigm, the IOI client also contains a routing path finder component which calculates the routing path among programmable switches, and selects the chain of switches to finish the inference task. The results of the routing path finder algorithm are encoded into the packet headers and guide packets to arrive at their designated switches in IOI. Note that we aim to design a light-weighted client which does not require complex operations. The IOI protocol and the routing algorithm can be directly implemented on the top of clients' current operating systems.

3 CONCLUDING REMARKS

Is in-network machine learning inference possible for deep neural networks? This problem is challenging with existing programmable switches, given their limited computing resources. IOI is the first step towards enabling in-network optical inference by incorporating advanced optical matrix multiplexing with non-linear activation on programmable switches. We believe that IOI provides new and exciting opportunities to design the next-generation machine learning systems with in-network computing for line-rate processing and ultra low latency.

4 ACKNOWLEDGMENT

We would like thank anonymous OptSys reviewers. This work was partly supported by ARPA-E ENLITENED PINE, DARPA FastNICs, and NSF ASCENT-2023468.

REFERENCES

[1] Juniper MX Series Universal Routing Platforms. https://www.juniper.net/us/en/products-services/routing/mx-series/datasheets/1000597.page.

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

[3] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018.

[4] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[5] Nadeen Gebara, Alberto Lerner, Mingran Yang, Minlan Yu, Paolo Costa, and Manya Ghobadi. Challenging the stateless quo of programmable switches. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, pages 153–159, 2020.

[6] Ryan Hamerly, Liane Bernstein, Alexander Sludds, Marin Soljačić, and Dirk Englund. Large-scale optical neural networks based on photoelectric multiplication. *Physical Review X*, 9(2):021032, 2019.

[7] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations*, 2015.

[8] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[9] Young-Bum Kim. The scalable neural architecture behind alexa's ability to select skills. *Amazon Science*, 2018. https://www.amazon.science/blog/the-scalable-neural-architecture-behind-alexas-ability-to-select-skills.

[10] Young-Bum Kim. Powered by ai: Oculus insight. *Facebook AI*, 2019. https://ai.facebook.com/blog/powered-by-ai-oculus-insight/.

[11] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091*, 2019.

[12] Kartik Pandit, Bianny Bian, Vishwa M Prasad, Atul Kwatra, Patrick Lu, Mike Riess, Wayne Willey, Huawei Xie, and Gen Xu. Modeling the impact of cpu properties to optimize and predict packet-processing performance. *Intel Case Study*. https://www.intel.com/content/dam/www/public/us/en/documents/case-studies/att-cpu-impact-on-packet-processing-perfomance-paper.pdf.

[13] Siri Team. Hey siri: An on-device dnn-powered voice trigger for apple's personal assistant. *Apple Machine Learning*, 2017. https://machinelearning.apple.com/research/hey-siri.

[14] Gordon Wetzstein, Aydogan Ozcan, Sylvain Gigan, Shanhui Fan, Dirk Englund, Marin Soljačić, Cornelia Denz, David AB Miller, and Demetri Psaltis. Inference in artificial intelligence with deep optics and photonics. *Nature*, 588(7836):39–47, 2020.

[15] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, et al. Machine learning at facebook: Understanding inference at the edge. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 331–344. IEEE, 2019.

[16] Zhaoqi Xiong and Noa Zilberman. Do switches dream of machine learning? toward in-network classification. In *Proceedings of the 18th ACM workshop on hot topics in networks*, pages 25–33, 2019.