# Lightning: A Reconfigurable Photonic-Electronic SmartNIC for Fast and Energy-Efficient Inference

Zhizhen Zhong Mingran Yang Jay Lang Christian Williams Liam Kronman
Alexander Sludds Homa Esfahanizadeh Dirk Englund Manya Ghobadi

Massachusetts Institute of Technology

## ABSTRACT

The massive growth of machine learning-based applications and the end of Moore's law have created a pressing need to redesign computing platforms. We propose Lightning, the first reconfigurable photonic-electronic smartNIC to serve real-time deep neural network inference requests. Lightning uses a fast datapath to feed traffic from the NIC into the photonic domain without creating digital packet processing and data movement bottlenecks. To do so, Lightning leverages a novel *reconfigurable count-action* abstraction that keeps track of the required computation operations of each inference packet. Our count-action abstraction decouples the compute control plane from the data plane by counting the number of operations in each task and triggers the execution of the next task(s) without interrupting the dataflow. We evaluate Lightning's performance using four platforms: a prototype, chip synthesis, emulations, and simulations. Our prototype demonstrates the feasibility of performing 8-bit photonic multiply-accumulate operations with 99.25% accuracy. To the best of our knowledge, our prototype is the highest-frequency photonic computing system, capable of serving real-time inference queries at 4.055 GHz end-to-end. Our simulations with large DNN models show that compared to Nvidia A100 GPU, A100X DPU, and Brainwave smartNIC, Lightning accelerates the average inference serve time by 337×, 329×, and 42×, while consuming 352×, 419×, and 54× less energy, respectively.

## CCS CONCEPTS

• **Hardware** → **Networking hardware**; **Emerging optical and photonic technologies**; *Hardware accelerators*; *Reconfigurable logic applications*; • **Computer systems organization** → **Optical computing**; *Real-time system architecture*; *Analog computers*;

## KEYWORDS

Photonic computing, Network hardware design, Computer architecture, Real-time AI, Machine learning inference

**ACM Reference Format:**

Zhizhen Zhong, Mingran Yang, Jay Lang, Christian Williams, Liam Kronman, Alexander Sludds, Homa Esfahanizadeh, Dirk Englund, Manya Ghobadi. 2023. Lightning: A Reconfigurable Photonic-Electronic SmartNIC for Fast and Energy-Efficient Inference. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10–14, 2023, New York, NY, USA.* ACM, New York City, NY, USA, 21 pages. https://doi.org/10.1145/3603269.3604821



*ACM SIGCOMM '23, September 10–14, 2023, New York, NY, USA*

ACM ISBN 979-8-4007-0236-5/23/09…$15.00
https://doi.org/10.1145/3603269.3604821

## 1 INTRODUCTION

Artificial Intelligence is redefining modern life by empowering novel machine learning inference applications. With the unprecedented growth of inference-based services hosted in datacenters, there is a pressing need for fast and energy-efficient systems to serve live inference queries in real time. As a prime example, in January 2023, ChatGPT served 600 million inference queries and consumed as much electricity as 175,000 people [59].

Photonic computing is an emerging area with the promise to revolutionize the computing world by using lightwaves and optical devices to perform fast and energy-efficient computation in the analog domain [31, 40, 42, 57, 61, 81, 85, 92, 113]. The key principle in photonic computing is that photonic devices have faster computing frequencies than transistors while generating less heat [84].

Recently, several papers demonstrated the potential to perform photonic computation at 100+ GHz frequency while consuming 40 atto Joules per operation [57, 67, 101, 111]. However, this paper demonstrates that data movement is a significant bottleneck in today's photonic computing approaches. In particular, once we factor in the digital datapath latency of today's proposals, the end-to-end inference latency explodes by five orders of magnitude, thereby eliminating the gains of photonic computing entirely (§3).

The root cause of this problem is that photonic computing cores are inherently passive devices without any memory or instructions to control the computation dataflow of complex real-world applications. As a result, prior work employed a stop-and-go approach by heavily involving a control software program (e.g., a Python script) in the photonic computing operations. This coupling of the control and data planes creates significant overhead in the datapath and hurts end-to-end latency.

This problem is worsened because the control plane is running with a digital clock frequency that is slower than the photonic cores. For instance, consider a photonic computing core operating at 100 GHz, controlled by digital software clocked at 1 GHz. In this case, any interactions between the photonics and digital domains (e.g., packet processing, data reads/writes) have the potential to pause 100 GHz operations to make control-plane decisions at 1 GHz, thereby slowing down the computation.

This paper demonstrates that to unleash the potential of photonic computing, we need to co-design the digital and photonic components together. Towards this vision, we propose Lightning, a reconfigurable photonic-electronic smartNIC with fast and energy-efficient *photonic computing cores* (§4).

Lightning addresses the critical datapath latency problem of today's photonic computing proposals using a novel *reconfigurable count-action* abstraction. Intuitively, our count-action abstraction decouples the control and data planes of inference requests by enabling the datapath to keep track of the directed acyclic graph

(DAG) of each inference request without interrupting the flow of the data in and out of photonic computing cores.

Lightning's count-action abstraction has three components: (*i*) a set of variables to count, (*ii*) a set of target results, and (*iii*) a set of actions to trigger when the result is equivalent to the target value. The count component keeps track of the required operations for each task of the DAG and triggers the execution of the next tasks immediately after the current task is finished without involving the control plane (§5). This technique is similar to the match-action abstraction in Tofino switches [34], where packets flow through a series of match-action units. However, unlike match-action units, Lightning's count-action units are reconfigurable at runtime.

We evaluate Lightning using four platforms: a hybrid photonic-electronic prototype (§6), an emulation environment (§7), chip synthesis (§8), and large-scale simulations (§9). Our prototype demonstrates the feasibility of performing 8-bit photonic multiply–accumulate (MAC) operations with 99.25% accuracy. Our experiments with the LeNet-300-100 [76] DNN show that Lightning achieves 96.2% image classification accuracy on the MNIST dataset [77] while accelerating inference serve time by 9.4× and 6.6× compared to Nvidia P4 and A100 GPUs, respectively. To the best of our knowledge, our prototype is the highest-frequency photonic computing system serving real-time inference queries at 4.055 GHz end-to-end. Our emulation results show that Lightning's top-5 inference accuracy is within 2.25% of digital accelerators. Our chip synthesis shows that the area footprint of a Lightning smartNIC is 2.55× smaller than a Stratix 10 FPGA used in Microsoft Brainwave smartNIC [51], and consumes 1.37× less power. Our large-scale simulations with seven representative DNN models (AlexNet [75], ResNet18 [65], VGG16 [44], VGG19 [70], BERT [46], GPT-2 [93], and DLRM [87]) show that compared to Nvidia A100 GPU, A100X DPU, and Microsoft Brainwave smartNIC, Lightning accelerates the average inference serve time by 337×, 329×, and 42×, while consuming 352×, 419×, and 54× less energy, respectively.

Finally, while this paper is focused on machine learning inference, this work is a first step towards a long-term vision of building full photonic computers connected with optical networks. Given the high barrier of entry to photonic computing research, to enable the SIGCOMM community to innovate in this space, we put together a photonic computing developer kit programmed through a Python API. Our source code is available at https://lightning.mit.edu.

## 2 BACKGROUND

Photonic computing is a revolutionary technology that has the potential to change the way we think about computation. Unlike traditional computers, which use electrical signals to perform calculations, photonic computing uses light. This allows for a much higher computation frequency, making it possible to perform fast and energy-efficient operations. This section provides a brief background on photonic computing and its potential benefits.

### 2.1 Photonic Vector Dot Product

**Amplitude modulation.** Amplitude modulation is a well-known technique to transmit digital data across optical fibers in datacenter and wide-area networks. Figure 1 shows a simplified representation of amplitude modulation in a commodity transceiver. An optical

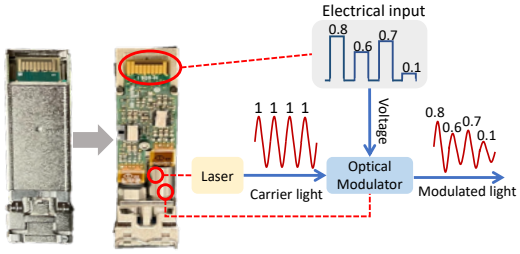


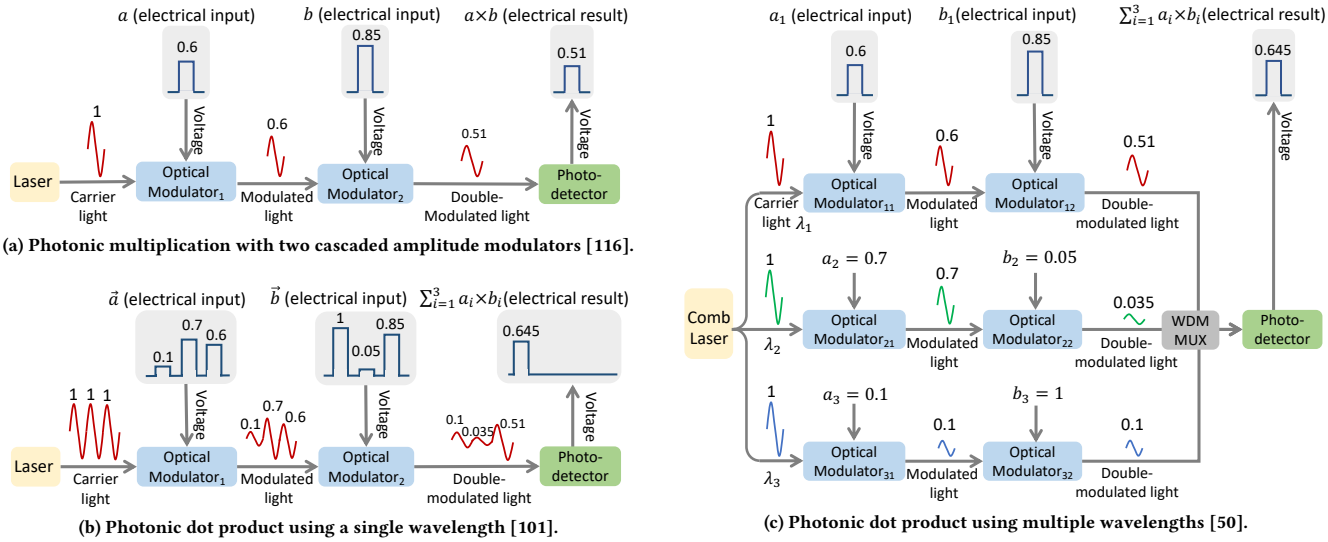

**Figure 1: Amplitude modulation in a commodity transceiver.**

modulator inside the transceiver adjusts the amplitude of a carrier light according to the electrical data represented as input voltages. In principle, the process of amplitude modulation is equivalent to *multiplying* the intensity of the carrier light by the input voltage in the analog domain [66]. The output of the modulator is a series of light intensities proportional to the input voltages (e.g., by applying 0.8 Volts to the modulator, the intensity of the modulated light becomes proportional to 0.8 dBm). This proportionality is constant and is determined by the modulator's materials [78].

**Photonic multiplication.** A common technique to perform multiplication in the photonic domain is to *cascade two amplitude modulators* back-to-back to multiply two input voltages [101, 102, 108]. As shown in Figure 2a, applying an input voltage $a$ to modulator$_1$ creates a lightwave with intensity proportional to $a$. This lightwave serves as a carrier signal for modulator$_2$. Now, applying a second input voltage $b$ to modulator$_2$ multiplies $a$ by $b$ in the photonics domain. The result is a double-modulated lightwave with its amplitude proportional to $a \times b$. The photodetector receives this light intensity from the second modulator and translates it into voltage. For example, let $a = 0.6$ and $b = 0.85$ represent the input numbers in the electrical domain. By feeding these numbers into the two optical modulators shown in Figure 2a, the intensity of the output light from the second modulator becomes proportional to the multiplication of the two input voltages, $a \times b = 0.51$.

**Photonic vector dot product.** There are two common techniques to leverage the photonic multiplication technique mentioned above to perform vector dot product in the photonic domain: (*i*) using a single wavelength, and (*ii*) using multiple wavelengths. Below we describe each technique in detail.

**Photonic vector dot product using a single wavelength.** A straightforward technique to accumulate element-wise multiplication of two vectors is to stream a series of input voltages $\vec{a}$ to modulator$_1$ and another synchronous stream of input voltages $\vec{b}$ to modulator$_2$, as shown in Figure 2b. A photodetector then detects the double-modulated lightwaves and generates a series of voltages proportional to the element-wise product $a_i \times b_i$. This constant proportionality in the photodetector is known as Einstein's photoelectric effect, a discovery that won the Nobel prize in 1921 [47]. Appendix A describes our calibration system to compute this proportionality factor. An integrating circuit, such as a capacitor attached to the photodetector's output port, accumulates the generated voltages streams and returns an electrical voltage proportional to the sum of the element-wise products [101]. For example, let $\vec{a} = [0.1, 0.7, 0.6]$ and $\vec{b} = [1, 0.05, 0.85]$ represent a series of input numbers in the electrical domain. By feeding these




(a) Photonic multiplication with two cascaded amplitude modulators [116].

(b) Photonic dot product using a single wavelength [101].

(c) Photonic dot product using multiple wavelengths [50].

**Figure 2: Illustration of the working principles of a photonic vector dot product core.**

numbers into the two optical modulators shown in Figure 2b, the intensity of the output light from the second modulator becomes proportional to the pairwise multiplication of each element in $\vec{a}$ and $\vec{b}$, or $[a_i \times b_i] = [0.1, 0.035, 0.51]$. The photodetector integrates the intensities of double-modulated light over multiple time steps and returns their sum value, $\sum_{i=1}^{3} a_i \times b_i = 0.645$.

**Photonic dot product using multiple wavelengths.** Another technique to multiply two vectors in the photonic domain is to represent each element of the vectors using a different wavelength. As shown in Figure 2c, to represent vectors $\vec{a}$ and $\vec{b}$, a comb laser [50, 52] generates a series of side-by-side wavelengths where $a_i$ and $b_i$ are represented by wavelength $\lambda_i$. Then, a set of cascaded modulators, shown horizontally, compute the element-wise multiplications of the two vectors simultaneously. A wavelength-division multiplexing multiplexer (WDM MUX) combines the double-modulated wavelengths $\lambda_i$. Finally, the photodetector returns an output voltage proportional to the sum of the incident light intensities [50]. Figure 2c illustrates this process using the same input vectors as the example in Figure 2b. The comb laser generates three wavelengths as carrier lightwaves. Vector $\vec{a}$'s elements $a_1$, $a_2$, and $a_3$ are fed into modulator$_{11}$, modulator$_{21}$, and modulator$_{31}$ and vector $\vec{b}$'s elements $b_1$, $b_2$, and $b_3$ are fed into modulator$_{12}$, modulator$_{22}$, and modulator$_{32}$, simultaneously. Each row of modulators performs the element-wise multiplication of $a_i \times b_i$ carried on wavelength $\lambda_i$. The photodetector detects all these wavelengths simultaneously and returns a single output voltage proportional to the sum of the element-wise products, $\sum_{i=1}^{3} a_i \times b_i = 0.645$.

## 2.2 Benefits of Photonic Computing

**Compute frequency.** Off-the-shelf Lithium Niobate ($LiNbO_3$) or Indium Phosphide ($InP$) optical modulators operate at 40 GHz frequency [6, 74]. These modulators are used ubiquitously for 400 Gbps+ optical networks in the Internet and datacenters [94]. Emerging materials, such as plasmonics and thin-film Lithium Niobate, can operate at 100 to 500 GHz frequency [37, 60, 72, 97, 104]. In contrast, the computation speed of today's digital accelerators is bounded by the digital clock frequency; i.e., ≈500 MHz for FPGAs [109], ≈1.05 GHz for TPUs [68], and ≈1.41 GHz for GPUs [3].

**Energy efficiency.** Prior work demonstrated that 8-bit photonic computing consumes 40 atto Joules per MAC using commodity optical devices at room temperature [101]. In contrast, the energy consumption of an 8-bit MAC in a 7 nm ASIC (e.g., GPUs and TPUs) is ≈0.07 pico Joules [68]. FPGAs consume ≈15 pico Joules for an 8-bit MAC operation using dedicated DSP blocks [89]. Hence, performing a MAC operation in the photonic domain is more energy efficient than ASICs and FPGAs by 1,750× and 375,000×, respectively.

**Parallel modulations on a single modulator.** An attractive feature of photonic computing is that optical modulators perform parallel multiplications with up to 200 co-propagating wavelengths [50, 111]. Such native parallelism provides significant performance gains by increasing the number of parallel photonic operations.

# 3 THE DATAPATH CHALLENGE

To reduce the response time of real-time inference queries, service providers serve machine learning models on computing-enabled smartNICs such as Microsoft Brainwave [51] and Nvidia A100X DPU [1].

Given the high compute frequency, low energy consumption, and native support for parallel operations in photonic computing, a natural question is: "Can we simply augment today's smartNICs with photonic computing cores to respond to live inference requests?"

To answer this question, we replicate the experimental setup of state-of-the-art photonic computing demonstrations [50, 101]. Figure 3 depicts our replication setup. This setup performs offline image recognition inference on hard-coded images by storing a series of inference images on a computer. The computer uses a streaming software application (e.g., a Python script) to create a series of vectors $\vec{a}$ and $\vec{b}$ corresponding to the images and DNN model parameters (step A). These vectors are then sent into an Arbitrary Waveform Generator (AWG) device [11] (step B). The AWG is a

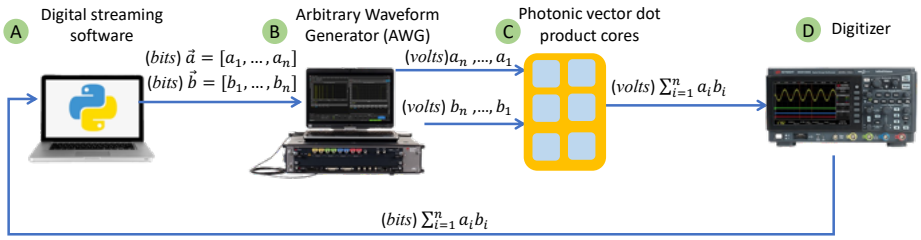



**Figure 3: The experimental setup of prior photonic computing work.**

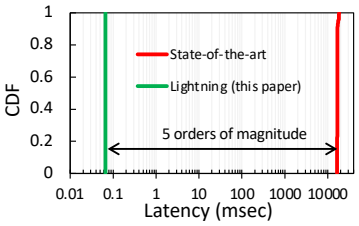



**Figure 4: End-to-end inference latency of Lightning vs. state-of-the-art photonic computing demonstrations.**

bulky and expensive lab device (≈ $100,000) that translates digital numbers into analog voltages. The voltages are then fed into one or more photonic vector dot product cores to perform a series of computations in the photonic domain (step C). The computation result is fed to another lab device, called a digitizer [26](≈ $50,000). The digitizer translates the analog voltages into the digital domain (step D) and feeds the bits into the Python program. Finally, the streaming software performs additional digital operations required by the DNN model (e.g., softmax, ReLU [76]) and collects all the required vector dot products for each layer. This process repeats until the final layer of the DNN model is computed.

Figure 4 plots the cumulative distribution function (CDF) of inference latency for 100 images. The figure shows that the state-of-the-art photonic computing approaches are five orders of magnitude slower than Lightning. The key reason behind the high latency in prior work is that their control plane is tightly coupled with the inference computation datapath.

Decoupling the control and data planes of photonic computing cores is non-trivial for several reasons. First, modulators and photodetectors are inherently passive devices without memory or instructions to control the computation dataflow of photonic operations or to distinguish meaningful results from noise. Second, the computation DAG for different DNNs is different, and the control plane must be able to adjust the operations for each DAG at runtime. Third, the DNN model parameters and user inference request data reside in the digital domain, while the vector dot products happen in the photonic domain with a much faster clock frequency than in the digital domain. Last, each DNN inference's DAG contains a series of operations with many task dependencies and non-photonic compute operations that must be carefully incorporated into the data plane. These operations must be finished at the digital clock frequency without slowing down the end-to-end application.

As a result, there is a need for a novel datapath that decouples the data and control planes of photonic computing systems while satisfying the following requirements:

- R1 Handle live user traffic arriving from remote users.
- R2 Support reconfigurability at runtime to serve inference requests for different DNNs.
- R3 Ensure the inference query data from remote users are multiplied correctly with the DNN model parameters.
- R4 Distinguish meaningful photonic computing result from noise.
- R5 Avoid making non-photonic compute operations a bottleneck.

## 4 LIGHTNING'S HIGH-LEVEL DESIGN

This section describes Lightning, a photonic-electronic smartNIC that enables fast and energy-efficient machine learning inference with a reconfigurable datapath that feeds traffic from the NIC into photonic vector dot product cores. Figure 5 illustrates Lightning's high-level design, described as follows.

**Packet parser.** To handle live user traffic (requirement R1), Lightning's packet parser receives packets from the 100 Gbps network interface (step 1). The parser identifies inference queries from regular packets based on the destination port number field in the incoming packet header. Once a packet is identified as an inference query, the parser extracts the DNN model ID and corresponding user data from the header. Depending on the DNN model, the inference query's data may be in the packet header or the payload. For instance, for a traffic classification inference use case, the packet parser uses header data (e.g., src IP, dst IP), whereas for language generation inference, the parser reads the payload as the user data (e.g., a search query typed by the user).

**DAG configuration loader.** Next, Lightning's DAG configuration loader reconfigures the datapath based on the computation DAG of the packet's DNN model (step 2). This module decouples the control-plane decisions from the computation operations in the data plane and enables Lightning to make control decisions in the data plane without stopping the data streams in and out of photonics. The DAG loader uses a key primitive in Lightning, called a reconfigurable count-action abstraction (§5). This abstraction enables the DAG configuration loader to reconfigure a series of datapath templates (e.g., fully-connected layers, convolution layers, attention layers, recurrent layers, adder tree modules, non-linear computation like ReLU and softmax, etc.) at runtime (requirement R2). Once the datapath is configured with the appropriate counts and actions for each DNN model, packets flow through the system without involving the control plane (unless an exception occurs).

**Memory controller.** While the DAG configuration loader reconfigures Lightning's datapath, it notifies the memory controller module to stream the corresponding DNN model parameters from off-chip memories, such as dynamic random-access memory (DRAM) or high bandwidth memory (HBM) (step 3). For fully-connected layers, the memory controller streams the weight matrices directly into the datapath. To reduce memory access overheads for convolution layers, the memory controller reads the convolution kernel only once and stores it in local register files for subsequent reuse.

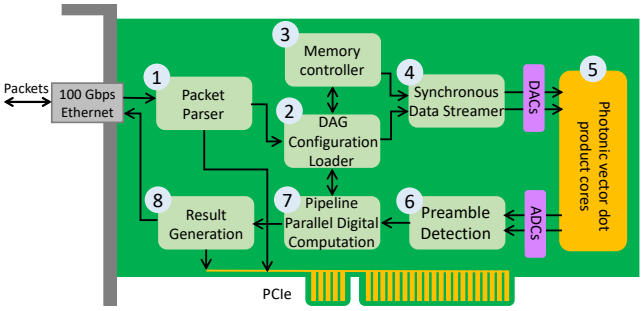



**Figure 5: High-level design of Lightning smartNIC.**

**Pipelined photonic-electronic computing.** Steps 4–7 are all performed in a pipelined fashion. In step 4, a data streamer module (§5.1) sends multiple parallel digital data into photonic vector dot product cores via on-chip digital-to-analog converters (DACs) [2, 82]. In steps 5 and 6, the photonic vector dot product cores compute the vector dot products of input data streams and return the results to analog-to-digital converters (ADCs), where a preamble detection module distinguishes the results from noise without stopping the flow of the data (§5.2). The vector dot product results are fed into a digital computation module (step 7). This step contains several pipeline parallel digital computation modules to perform any additional digital operations, such as ReLU and softmax (§5.3).

**Result generation.** Steps 4–7 are repeated until the DAG is completed and the inference result is ready. Depending on the inference packet, Lightning creates a response packet and sends it to the user through the Ethernet interface or the PCIe bus (step 8).

## 5 RECONFIGURABLE COUNT-ACTION ABSTRACTION

Lightning's key enabler is a reconfigurable count-action abstraction that keeps track of the required computation DAG to make control decisions at runtime on the data plane. Figure 6 provides a conceptual illustration of Lightning's reconfigurable count-action abstraction. This abstraction has three components: (*i*) *counts*, a set of variables to count; (*ii*) *targets*, a set of target results; and (*iii*) *actions*, a set of actions to trigger when the result is equivalent to the target value. The *count* primitive keeps accumulating the specified variables across digital datapath clock cycles. Once the result reaches the *target*, the count variable is set back to zero, and the *actions* are triggered.

This section describes how Lightning leverages this count-action abstraction to enable three of its datapath components: synchronous data streamer (§5.1), preamble detection (§5.2), and pipeline parallel digital computation (§5.3). Finally, we discuss how Lightning supports different DNN models at runtime (§5.4).

### 5.1 Synchronous Data Streamer

Lightning's synchronous data streamer module is responsible for creating a set of parallel data streams based on the ratio of the photonic computing frequency to the digital clock, as shown in Figure 7. For instance, suppose the clock frequency of photonic vector dot product cores, DACs, and ADCs is 4 GHz, but the clock frequency of the digital datapath is 1 GHz. In this case, the data

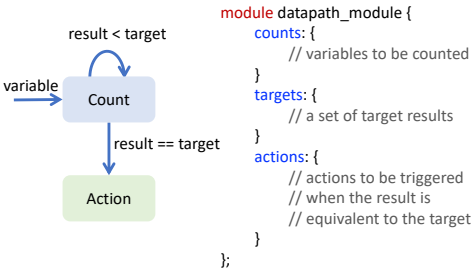



**Figure 6: Our reconfigurable count-action abstraction.**

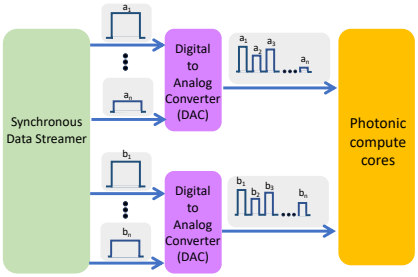


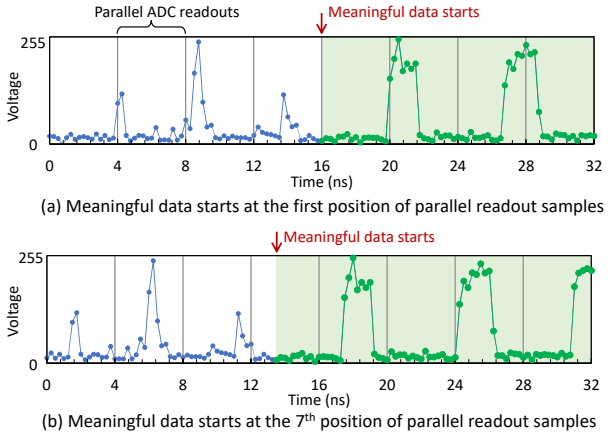

**Figure 7: Lightning's data streamer feeds a series of synchronous parallel data streams into photonic compute cores.**

streamer module creates four parallel streams at each digital clock cycle to feed data into photonic cores at 4 GHz.

However, computing the vector dot product of two streams of high-speed voltages $\vec{a} = [a_i]$ and $\vec{b} = [b_i]$ in the photonic domain requires precise element-wise multiplication across the input vectors. The $i^{th}$ element in $\vec{a}$ must be multiplied by its corresponding element in $\vec{b}$; otherwise, the result is incorrect. In digital computing, this is easily achieved, as arithmetic logic units (ALUs) have two input operands under the same clock. Thus, the ALU can fetch both elements from registers simultaneously. But feeding synchronous data into optical modulators is challenging because the vectors are fed into modulators as time series of analog voltages with a fine-grained time resolution (e.g., multiplying two vectors at 10 GHz frequency means the distance between consecutive voltages is only 0.1 ns). Moreover, photonic multiplication requires two modulators (§2), making it crucial to synchronize the modulators' inputs.

This problem is exacerbated for large DNNs since Lightning stores their parameters on DRAM or HBM, while data packets and intermediate activations reside on SRAM. Even a slight latency variation in the off-chip memory access [38] will result in out-of-sync elements to the DACs, producing out-of-sync voltages and incorrect computation results (requirement R3).

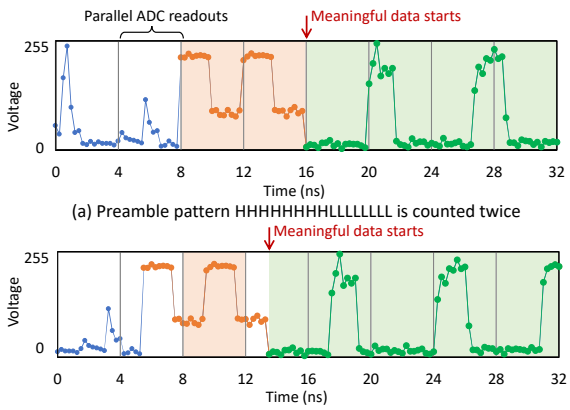
We address the above challenge using our count-action abstraction, as illustrated in Listing 1. We denote the $i$-th parallel AXI stream data on the $i$-th DAC by DAC[$i$]. Each DAC[$i$] has a flag called *valid*, which is automatically set to be 1 when a new 8-bit data sample is ready to be transferred. This flag flips back to 0 if no new data samples arrive after the currently valid data are sent out [8]. Lightning uses the count feature to keep track of the flags *across parallel DAC streams at each digital clock cycle.* To ensure the DACs are all synchronized before sending the voltages to the modulators, Lightning's streamer module counts the sum of the valid




(a) Meaningful data starts at the first position of parallel readout samples

(b) Meaningful data starts at the 7th position of parallel readout samples

**Figure 8: Sample readouts from a 4 GS/s ADC.**




(a) Preamble pattern HHHHHHHHLLLLLLLL is counted twice

(b) Preamble pattern LLLLLLHHHHHHHHLL (6-samples shifted) is counted once

**Figure 9: Preamble pattern distinguishes noise from data.**

bits for all DACs (i.e., $\sum_{i=1}^{n} DAC[i].valid$, where $n$ is the number of DACs). The data streamer module only triggers streaming voltages when the count result becomes equivalent to the number of DACs, indicating that all DACs hold valid data samples.

```
module synchornous_data_streamer {
    counts: {
        // count the sum of valid DAC flags
        sum DAC[i].valid (i = 1 to num_DACs)
    }
    targets: {
        // trigger when the sum equals the number of DACs
        num_DACs
    }
    actions: {
        // stream DACs' data into photonic cores
        stream DAC[i].data (i = 1 to num_DACs)
    }
};
```

**Listing 1: Synchronous data streamer module.**

## 5.2 Distinguishing Data from Noise

As soon as Lightning streams parallel data into its photonic vector dot product cores, it starts reading the results from its ADCs. Each ADC readout contains multiple parallel data samples simultaneously, and Lightning must identify which samples are noise and which are meaningful data (requirement R4). Figures 8a and 8b show the voltage readouts from an ADC with 4.055 Giga samples per second (GS/s) frequency in our prototype. The blue voltages are noise, and the green ones are meaningful data. Reading these streams into a digital datapath with 253.44 MHz clock frequency means that every ≈4 ns, the ADC delivers 16 parallel samples to the datapath. But the datapath logic has no additional information on which samples are noise and which are photonic compute results. For instance, in Figure 8a, meaningful data start at the beginning of the ≈4 ns interval, which means all 16 samples are useful photonic compute results. But the meaningful data starts at the 7th sample in Figure 8b, indicating that samples 6 to 15 are photonic compute results, and samples 0 to 5 are noise.

To address this challenge, Lightning adds a preamble pattern to each vector in the digital domain before streaming its data into the DACs. The preamble is a series of pre-determined single-cycle patterns (e.g., *HHHHLLLL*) repeated $P$ times, where $P$ is a configurable parameter that is model-agnostic and only depends on the signal-to-noise ratio (SNR) conditions of the setup, $L$ corresponds to a low voltage, and $H$ corresponds to a high voltage. To detect meaningful data in the ADC readout, Lightning uses our count-action abstraction to count the number of times the preamble pattern is detected in continuous ADC readouts.

```
module preamble_detection_per_ADC {
    counts: {
        // count the number of occurrences of k-shifted preamble
        (ADC.data == (preamble_pattern << k))
    }
    targets: {
        // trigger if a 0-shifted pattern is counted P times, else
        // trigger if a k-shifted pattern is counted P-1 times
        if (k == 0): P
        else: P - 1
    }
    actions: {
        // stream the ADC readout starting from the kth position
        stream ADC.data[k:]
    }
};
```

**Listing 2: Preamble detection module.**

Listing 2 shows our preamble detection module per ADC. The module counts the number of times the preamble pattern, or a $k$-shifted version, appears in the ADC readout, where $k$ ranges from 0 to the number of samples in one clock cycle minus one. If the preamble is counted exactly $P$ times, meaningful data starts at the beginning of the datapath clock cycle. But when meaningful data are shifted within the datapath clock cycle by $k$ samples, a $k$-shifted version of the preamble will be counted $P-1$ times. The variable $k$ in the $k$-shifted pattern indicates the position of the first meaningful data in the ADC readout. Figures 9a and 9b demonstrate the preamble patterns corresponding to Figures 8a and 8b, respectively. The orange voltages are Lightning's preamble patterns.

## 5.3 Pipeline Parallel Digital Computation

One inherent challenge of photonic computing is that the light intensity is always a non-negative value. Prior approaches require sending positive/negative values separately, either at different times or using different photonic hardware [101, 106], effectively halving

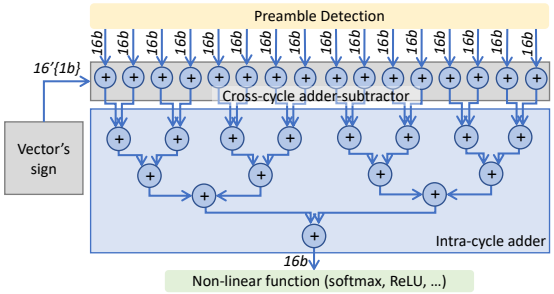



**Figure 10: Pipeline parallel digital adder module.**

the computation speed or doubling the required number of photonic devices. Unlike these approaches, Lightning addresses this challenge by using additional digital computations on the datapath to support DNN models with negative parameters. Moreover, the computation DAG of a DNN layer requires not only photonic vector dot products but also additional operations like non-linear functions. Lightning implements these non-linear functions in the digital domain, too. To avoid making these non-photonic computations a bottleneck (requirement R5), Lightning performs digital computations using a pipeline parallel adder module and a pipeline parallel non-linear function module.

Lightning's pipeline parallel adder module has two components: (*i*) a cross-cycle adder-subtractor and (*ii*) an intra-cycle adder. As we show in Figure 10, after our preamble detection module (§5.2) detects the starting point of the photonic vector dot product results, the parallel data samples from the ADC readout are first streamed into the cross-cycle adder-subtractor. In this figure, 16 parallel samples are read simultaneously at each ADC readout. Each data sample is 16 bits.[1] The non-negative photonic vector dot product results from the ADC readout that stream into this module pick up their paired signs as the control signals accordingly. The cross-cycle adder-subtractor component has 16 adder-subtractors to perform 16 addition or subtraction operations based on control signals per datapath clock cycle.[2] Moreover, in scenarios where the length of a vector is larger than the number of photonic accumulation wavelengths (§2.1), Lightning uses its adder-subtractor module to aggregate partial vector dot products until the entire vector is accumulated.

```
module cross_cycle_adder_subtractor_module {
    counts: {
        // count the number of summations in the cross-cycle adder-subtractor
        sum cross_cycle_adder_subtractor[i].valid
    }
    targets: {
        // trigger when the entire vector is accumulated
        vector_length / num_accumulation_wavelengths
    }
    actions: {
        // stream the result to intra-cycle adder
        stream cross_cycle_adder_subtractor[i].data
    }
};
```

**Listing 3: Cross-cycle adder-subtractor module.**

[1] Lightning's data samples are 8 bits, but to avoid digital accumulation overflow, we pad each 8-bit sample with eight additional zeros.

[2] The signs of photonic vector dot products are pre-processed and separated from the absolute values of vectors in an offline phase ahead of the inference.

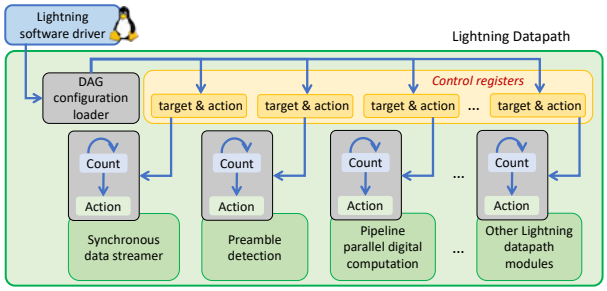



**Figure 11: Lightning's count-action abstraction enables implementing runtime reconfigurable modules on its datapath.**

Listing 3 shows how Lightning uses its count-action abstraction in its adder-subtractor module to trigger the execution of the intra-cyle adder step. The module counts the number of times each adder-subtractor performs accumulation. The required number of cross-cycle additions/subtractions is the length of the vector divided by the number of wavelengths used for accumulation in the photonic domain. When the count reaches this value, the subsequent intra-cycle adder tree is triggered. The intra-cycle adder tree aggregates multiple parallel samples within the same digital cycle into one result. The intra-cycle adder requires $log\ k$ clock cycles, where $k$ is the number of parallel data samples in each ADC readout.

Once the final accumulation result is computed in the intra-cycle adder, the next step is to perform non-linearity functions, such as ReLU or softmax, on the result. Lightning's count-action abstraction triggers the computation of non-linear modules based on the count of the number of elements in the vector dot product. Depending on the complexity of the non-linear function, this step may require additional clock cycles to compute the final result.[3] Since the non-linear computations are performed once per vector dot product, these additional clock cycles are pipelined through all the vector dot products of DNN layers and only add a few extra cycles to the last vector dot product. Finally, to finish the computation of a layer, Lightning's count-action abstraction counts the number of the results and triggers the next layer's computation as soon as the size of the results matches the input size of the next layer.

## 5.4 Supporting Runtime Reconfigurability

Lightning supports a variety of DNN models by embedding multiple instances of its count-action logic in its datapath, as shown in Figure 11. The count-action instances decide when to start or finish the operations for each datapath module by reading the *target* and *action* values from the centralized control registers. To reconfigure these count-action instances for different DNN layers, the DAG configuration loader modifies the values of corresponding control registers at runtime based on the computation DAG of the DNN. A customized kernel driver or a userspace I/O interface specifies the appropriate *target* and *action* values for different DNN layers and transfers them to the DAG configuration loader.

For example, when Lightning receives a packet requesting a LeNet-300-100 model [76], the DAG configuration module loads the appropriate count-action values for performing inference on the first layer of this model and writes these parameters to the control

[3] Our ReLU and softmax implementations take one and eight clock cycles, respectively.

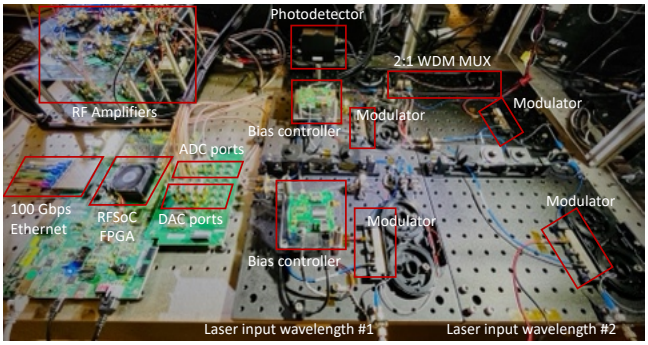



**Figure 12: [Testbed] Photo of our prototype.**

registers to reconfigure the datapath modules to perform the computation for a fully-connected layer size of 784×300. Meanwhile, if a second packet carrying an ImageNet [45] picture arrives requesting inference on a VGG-16 model [44], the DAG configuration loader writes another set of parameters to the control registers, such that the datapath modules are reconfigured to perform convolutions with kernel size $3 \times 3$ on ImageNet images size $224 \times 224$.

## 6 PROTOTYPE

We build a fully-functional prototype of Lightning to demonstrate the feasibility of a hybrid photonic-electronic smartNIC.

### 6.1 Prototype Setup

**Digital components.** Figure 12 shows a photo of our prototype and Figure 13 shows its detailed hardware architecture. We implement Lightning's smartNIC functionalities and datapath in Register Transfer Level (RTL) with ≈10K lines of code using the Verilog language on a Xilinx Zynq UltraScale+ RFSoC ZU28DR FPGA [23], hosted on the ZCU111 board [25]. We verify the RTL implementation using a Verilator-based cycle-accurate testbench [22]. We implement Lightning's datapath RTL design together with Xilinx's Zynq UltraScale+ RFSoC RF data converter (ADC/DAC) IP [24], 100 Gbps Ethernet (CMAC) IP [20], and DDR4 DRAM IP [21] using Xilinx Vivado 2022.2 and generate the system bitstream. We use the AXI stream [8] and the AXI lite protocols [7] to exchange data and control signals between the FPGA programmable logic, the Xilinx IPs, and the embedded Linux system. The FPGA is configured to run at 253.44 MHz with 16 samples per FPGA clock cycle, resulting in an analog data sampling rate of 4.055 GS/s for each DAC and ADC, where each sample represents an 8-bit fixed-point number in the analog domain. This 4.055 GS/s analog data rate allows Lightning to perform computation at 4.055 GHz. To the best of our knowledge, this is the highest-frequency photonic computing prototype capable of serving real-time inference requests. Upgrading to higher photonic computing frequencies only requires increasing the degree of parallelism in Lightning's count-action modules and AXI stream bit widths, but it does not require changes to the architecture dataflow.

**Photonic components.** Our prototype includes one photonic vector dot product core with two wavelengths using four off-the-shelf 15 GHz modulators [5]. To generate light, we use two tunable telecom laser sources and set them to 1544.53 nm and 1552.52 nm wavelengths, respectively. Each pair of modulators perform element-wise multiplication on a different wavelength. To aggregate the element-wise multiplication results, we use a commercially-packaged 9.5 GHz photodetector [18] that accumulates light intensities from different wavelengths (§2.1).

**Packet processing.** To receive inference requests from remote users, we implement a 100 Gbps Ethernet interface using Xilinx's CMAC IP core. The CMAC connects the Ethernet PHY to Lightning's packet parser module. The packet parser is capable of identifying Lightning's inference packets and forwarding them to our DAG configuration loader to trigger our pipelined photonic-electronic compute logic. The packet parser forwards the packets to a packet processing module before entering the kernel space. The packet processing module implements default NIC functionalities and advanced smartNIC features, such as intrusion detection [114] and transport protocol offload [73].

**DRAM access.** Lightning has access to a 4 GB DDR4 memory directly attached to the datapath. To support large DNN models, Lightning stores pre-trained DNN models in its DRAM. We implement a DDR controller in Lightning's datapath to manage the memory access and data exchanges. The DDR4 processes $2.67 \times 10^9$ transactions with 64 bits per transaction, resulting in a data rate of ≈170 Gbps, higher than both the CMAC input data rate (100 Gbps) and the aggregate data rate of the two DACs in our prototype that are responsible for converting DNN parameters ($2 \times 4.055$ GS/s $\times$ 8b/S = 64.88 Gbps). Because of this data rate difference, we implement a back-pressure AXI stream with a DRAM buffer to alleviate data burstiness when reading from DRAM. Note that upgrading to a higher photonic computing frequency or number of DACs to support larger photonic parallelism requires increasing the DRAM interface bandwidth or utilizing HBM with multiple stacks. For example, state-of-the-art HBM2 chips provide 15.2 Tbps bandwidth [90] requiring 468 wavelengths at the current 4.055 GHz frequency, or at least 20 wavelengths at 97 GHz frequency (§8).

**PCIe interface with the local host.** Lightning does not require any PCIe interactions for incoming inference packets because all inference operations are performed on the NIC, and the inference packets are not punted through PCIe. Therefore, Lightning uses the PCIe interface to interact with the local host for forwarding regular packets or updating DNN model parameters.

**Customized embedded Linux environment.** We extend Xilinx PetaLinux [15] to build a customized embedded Linux environment running on the ARM core of the RFSoC FPGA. Our Linux environment manages the hardware interfaces between Lightning's datapath, DACs, ADCs, clock chips, DRAM, CMAC, and USB.

**Python API.** In addition to Lightning's specialized RTL-based datapath that is designed for fast data movement, we develop a customized Python software stack on our FPGA based on PYNQ [17] and QICK [16] libraries. Our Python API enables programmers access to DACs/ADCs that are directly connected to photonic cores for micro-benchmarking and debugging. Lightning's Python API supports (*i*) sending/receiving data to/from photonic vector dot product cores to benchmark the computing accuracy, (*ii*) characterizing the SNR of photonic cores for calibration, and (*iii*) configuring

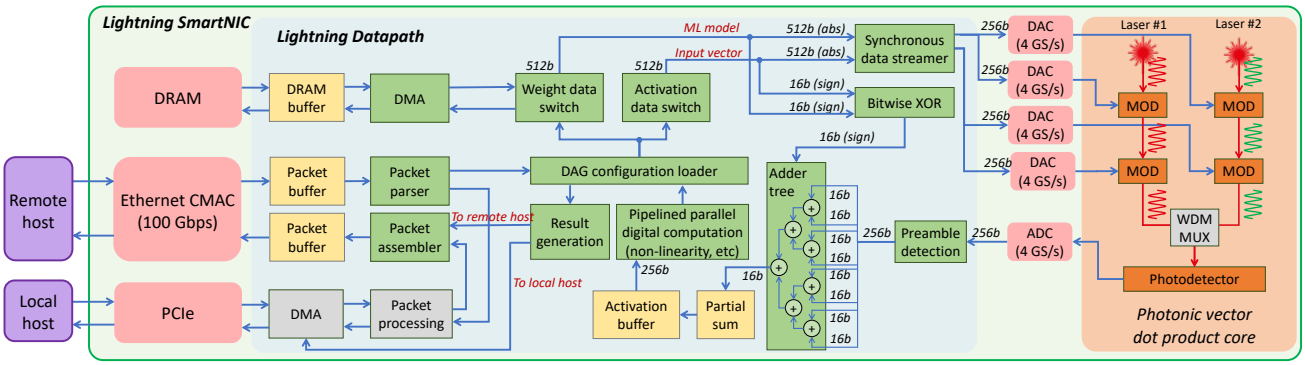



**Figure 13: [Testbed] Lightning's prototype architecture.**

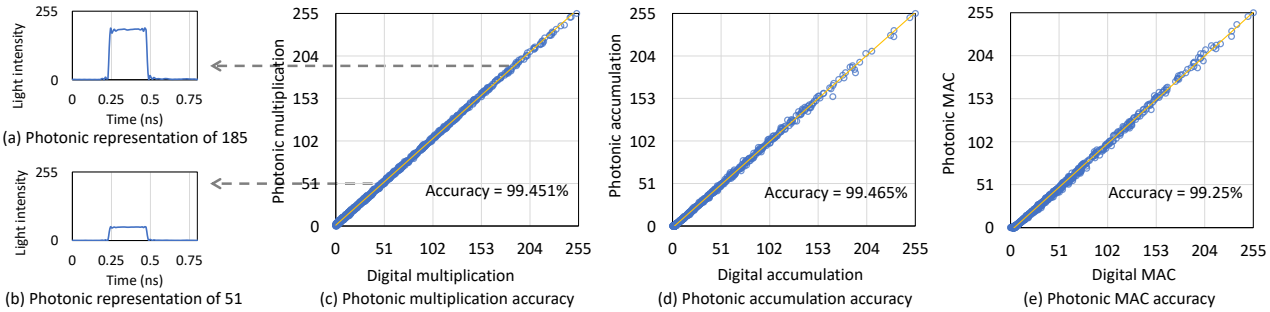



**Figure 14: [Testbed] Benchmarking the accuracy of photonic computing operations with unsigned 8-bit fixed-point operands.**

the bias voltage input of optical modulators. Figure 27 in Appendix G shows a code snippet of our Python API.

## 6.2 Micro Benchmarks

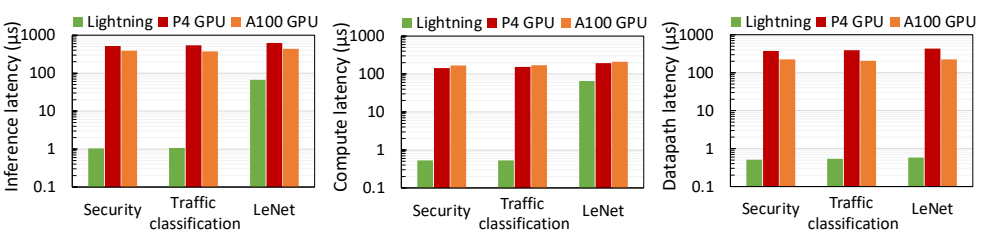
**Data encoding.** The photonic computing accuracy in Lightning depends on the number of distinct levels that are distinguishable in the analog domain. Following prior work [61], we use 256 levels in our prototype to encode unsigned fixed-point 8-bit numbers into the light. As shown in Figures 14a and 14b, we take the amplitude of the carrier light as maximum intensity (represented by 255) and normalize all other light intensities.

**Photonic MAC accuracy.** To evaluate the accuracy of photonic MAC operations in our Lightning prototype, we first generate 1,000 pairs of unsigned fixed-point 8-bit random numbers and compute their multiplication and accumulation results in the photonic domain using our Python API. We then perform the same operations in the digital domain. We define the photonic computing error to be the difference between the photonic result and its corresponding digital result. Figures 14c and 14d show that the standard deviations of errors are 0.549% and 0.535% for photonic multiplication and accumulation operations, respectively. This means Lightning achieves 99.451% and 99.465% accuracy for photonic multiplication and accumulation, respectively. We further test 1,000 photonic MAC operations using the same setup and plot the results in Figure 14e. We find that the standard deviation of photonic MAC's error is 0.75%, achieving a 99.25% photonic MAC accuracy.

## 6.3 Real-time DNN Inference

**Methodology and setup.** We use our count-action abstraction to reconfigure Lightning's datapath to support three different DNN models: (*i*) a security DNN model (1,568 parameters) for network traffic anomaly detection [99] with UNSW-NB15 intrusion dataset [86], (*ii*) a traffic classification DNN model [99] (1,696 parameters) with IoT traffic traces dataset [100], and (*iii*) an image classification model, called LeNet-300-100 [76] (266,200 parameters) with MNIST handwriting recognition dataset [77]. We implement the first two models based on N3IC's open-source code [14], except that we use 8-bit operations instead of binary. We train the third model using PyTorch for 500 epochs on a GPU server with 8-bit quantized parameters. We then measure the inference latency and accuracy of these models on Lightning's prototype and compare the results with two Nvidia Triton servers [4]. Each Triton server has a 100 Gbps NIC for serving traffic. One server is equipped with an Nvidia P4 GPU, and the other one has an A100 GPU.

**End-to-end inference latency.** The end-to-end latency reflects the time from the moment an inference request arrives until the moment the inference result packet leaves the system. Figure 15a compares the end-to-end latency of our DNN models using Lightning versus Nvidia Triton servers. The figure shows that Lightning's prototype accelerates the end-to-end inference latency of the security (and traffic classification) DNN(s) by 499× (508×) and 379× (350×), compared to Nvidia Triton serves with P4 and A100 GPUs, respectively. The figure also shows that Lightning accelerates the end-to-end inference latency of the LeNet DNN by 9.4× and 6.6× compared to P4 and A100 GPUs, respectively.




(a) End-to-end latency (b) Compute latency (c) Datapath latency

**Figure 15: [Testbed] End-to-end inference latency breakdown for three DNN models.**

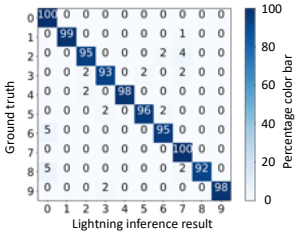



**Figure 16: [Testbed] LeNet inference accuracy on Lightning.**

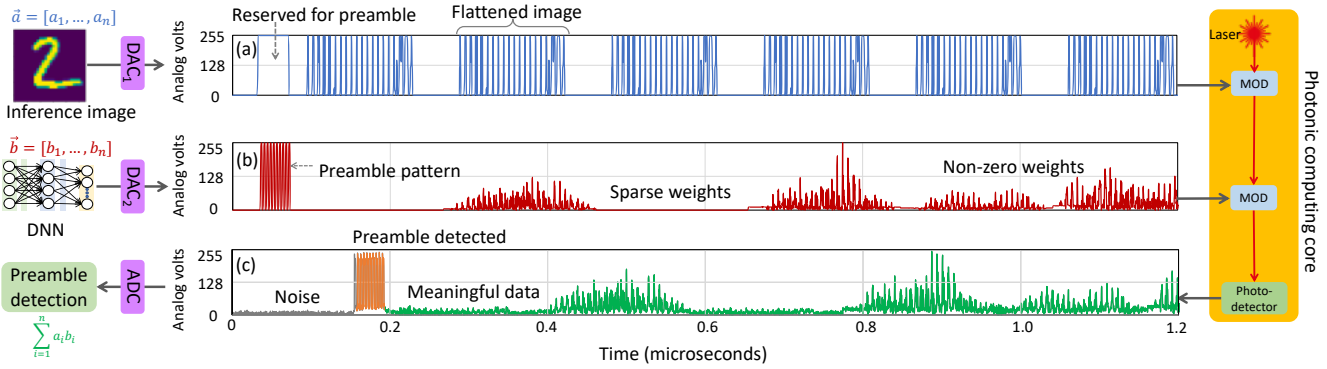



**Figure 17: [Testbed] Synchronous parallel data streams and preamble detection for serving LeNet inference queries.**

**Inference latency breakdown.** To demonstrate the impact of Lightning's datapath on inference latency speedups, we break down the end-to-end inference latency results into two components: datapath and compute. The datapath latency of Lightning includes all the digital components in its datapath during inference, while its compute latency captures all stages of computing (including photonic vector dot product, adder-subtractor, and non-linearity). We obtain the datapath and compute latency of P4 and A100 GPUs from our Triton servers. Figure 15b shows Lightning's compute latency increases as the model size increases while Figure 15c shows Lightning's datapath latency is stable because these three models have the same set of count-action modules in their datapath.

**Inference accuracy.** We measure the inference accuracy of the LeNet model by serving 1000 inference requests on our prototype. Figure 16 shows that Lightning's top-1 inference accuracy is 96.2%. We also measure that the inference accuracy of this model on a GPU is 97.45% at 8-bit precision.

**Synchronous data streaming.** Figures 17a and Figures 17b show the time series of serving a LeNet inference request using the Lightning prototype. The datapath uses two parallel streams to send inference data ($\vec{a}$) and DNN parameters ($\vec{b}$) to two optical modulators. Using our synchronous data streamer module (§5.1), the datapath synchronizes the streams before sending them to the photonic vector dot product core. The figures also show Lightning's preamble pattern added to each vector. The preamble pattern we use in our testbed is *HHHHHHHHLLLLLLLL*, repeated ten times.

**Preamble detection.** Figure 17c shows the data readout from the ADC in our prototype. To identify the starting point of meaningful results, Lightning uses its count-action abstraction to count the number of times the preamble pattern is repeated (§5.2). The orange lines in Figure 17c show the detected preamble pattern and the position of meaningful data in the ADC readout.

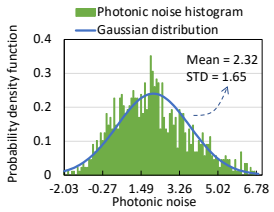



**Figure 18: [Testbed] photonic multiplication noise.**

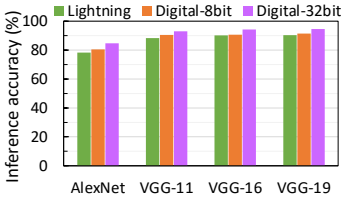



**Figure 19: [Emulation] Top-5 inference accuracy.**

## 7 ACCURACY EMULATION

To evaluate the impact of errors in the photonic domain on large DNNs, we develop a Python-based photonic emulator and calibrate it with the device specifications and noise models of our prototype.

**Analog noise model.** There are two major sources of noise in our prototype: shot-noise and thermal noise. Prior work [101] showed both noise sources can be modeled using a Gaussian distribution. As shown in Figure 18, we measure the photonic multiplication noise on our testbed and fit a Gaussian distribution with a mean of 2.32 and a standard deviation of 1.65 (0.65% out of 255).

**Python emulator.** Using the noise model, we develop an emulator capable of performing inference with 8-bit photonic, 8-bit digital, and 32-bit digital computation schemes. To emulate Lightning, we quantize multiplication operands and results to 8 bits and apply our

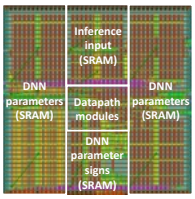



**Figure 20: [Synthesis] Datapath chip layout for one photonic MAC.**

| Datapath modules | Area ($mm^2$) | Power (W) |
|---|---|---|
| Packet I/O (Steps 1,8) | 0.08 | 0.034 |
| Memory controller (Step 3) | 0.12 | 0.067 |
| Count-action modules (Steps 2,4,6,7) | 1.26 | 0.156 |
| Total | 1.46 | 0.257 |

**Table 1: [Synthesis] Chip area and power breakdown of digital datapath modules for one photonic MAC.**

Gaussian noise model to the results of each MAC while performing inference. Our emulator implements the entire operations of AlexNet [75], VGG11 [98], VGG16 [98], and VGG19 [98] models. We validate the accuracy of the model using the ImageNet dataset [45] and report the average accuracy over ten experiments.

**Inference accuracy.** Figure 19 shows Lightning's top-5 inference accuracy is within 2.09% of an 8-bit digital accelerator for AlexNet, and within 2.25%, 0.51%, and 1.05% for VGG11, VGG16, and VGG19, respectively. To evaluate the impact of 8-bit versus 32-bit precision on inference accuracy, the figure includes the accuracy of each model with the default 32-bit precision. The results show that Lightning's quantization into 8 bits and its photonic noise do not significantly reduce the inference accuracy. Prior studies made a similar observation and suggested 8-bit quantization to reduce the footprint of DNN models [32, 54, 61, 103].

## 8 ASIC SYNTHESIS

To evaluate the area and power of a production-level Lightning chip, we propose a full photonic-electronic chip design capable of performing 576 photonic MAC operations in a single step using 24 wavelengths. We achieve this by combining two native features of photonic computing (§2). The first feature enables us to perform 24 parallel multiplications with 24 wavelengths on a single modulator. The second feature supports vector dot product calculation by accumulating the element-wise multiplication results of 24 wavelengths on a single photodetector. Putting these two features together, our proposed Lightning chip is capable of performing $24 \times 24 = 576$ photonic MAC operations simultaneously. We use 97 GS/s DACs and ADCs, as well as 100 GHz modulators [105] and photodetectors [83] in our proposed chip. As a result, the computing frequency of our proposed chip is 97 GHz. Appendix E provides further details.

**Digital datapath area and power.** We start by evaluating the area and power of digital datapath modules required for one photonic MAC operation. To do so, we synthesize the Lightning datapath RTL with one wavelength using Cadence Genus synthesis software with a commercial 65 nm process library. The software outputs a gate-level description of the electronic circuits in a standard format called netlist. We feed the chip area and the netlist representation into the Cadence Innovus Implementation System to obtain the chip layout. To measure the power consumption, we take the netlist representation from the chip area analysis and annotate the toggle rate of the digital gates using waveforms generated from our Verilator testbench. Figure 20 shows the layout of the datapath components for one MAC operation, and Table 1 lists the area and power breakdown of different components. The area of this one MAC datapath

| Type | Component | Count | Unit area ($mm^2$) | Area ($mm^2$) | Unit power (W) | Power (W) |
|---|---|---|---|---|---|---|
| Digital | Packet I/O | 24 | 0.0086 | 0.207 | 0.009 | 0.227 |
| | Memory controller | 576 | 0.129 | 7.444 | 0.0186 | 10.72 |
| | Count-action | 576 | 0.136 | 78.159 | 0.0433 | 24.96 |
| | HBM2 [41, 90] | 1 | 81.1 | 81.1 | 7.41 | 7.41 |
| | DAC [88] | 600 | 0.58 | 348 | 0.077 | 46.2 |
| | ADC [88] | 24 | 0.58 | 13.92 | 0.075 | 1.8 |
| | Total | | | 528.829 | | 91.317 |
| Photonic | Modulator [105] | 600 | 2.5 | 1500 | 3.88e-6 | 2.23e-3 |
| | Photodetector [83] | 24 | 3.2e-5 | 7.68e-4 | | |
| | Laser [112] | 1 | 0.01 | 0.01 | | |
| | Total | | | 1500.01 | | 2.23e-3 |
| Total | | | | 2028.839 | | 91.319 |

**Table 2: Area and power of a Lightning chip with 576 photonic MACs (details in Appendix E).**

| Platforms | Lightning | P4 | A100 | A100X | Brainwave |
|---|---|---|---|---|---|
| Power (W) | 91.319 | 75 | 250 | 300 | 125 |
| # MAC units | 576 | 2560 | 6192 | 6192 | 96000 |
| Single unit power (W/core) | 0.1585 | 0.0293 | 0.0362 | 0.0434 | 0.0013 |
| Clock frequency (GHz) | 97 | 1.114 | 1.41 | 1.41 | 0.25 |
| Energy per operation (pJ) | 1.634 | 26.299 | 25.652 | 30.782 | 5.208 |
| Lightning energy savings | 1× | 16.09× | 15.69× | 18.83× | 3.19× |

**Table 3: End-to-end energy consumption per MAC.**

is 1.46 $mm^2$, where 0.08 $mm^2$, 0.12 $mm^2$, and 1.26 $mm^2$ are occupied by packet I/O, memory controller, and count-action modules, respectively. Its power consumption is 0.257 W, of which 0.034 W, 0.067 W, and 0.156 W are consumed by packet I/O, memory control, and count-action modules, respectively.

**Full chip area and power projection.** Using the above synthesis results on the datapath area and power in 65 nm of one photonic MAC, we approximate the area and power of the digital datapath components of a Lightning chip in 7 nm with 576 MAC operations. Following prior work's comparison of 45 nm and 7 nm processes [69], we expect a 9.3× (65 nm/7 nm) and 3.6× scale down in area and power, respectively. The first three rows in Table 2 show the projected area and power of different datapath components. For these datapath modules, we expect the packet I/O modules to scale by 24× because the inference requests are allocated on 24 parallel wavelengths. Meanwhile, we estimate the memory controller and count-action modules to scale by at most 576× to support 576 simultaneous photonic MAC operations. We use HBM2 to store the DNN model parameters and estimate the power of HBM2 based on memory bandwidth × energy per bit [90]. Finally, we obtain the area and power requirements of DACs and ADCs from state-of-the-art reports [88]. Overall, the required chip area and power consumption of digital components of a Lightning chip with 576 MAC operations is 528.829 $mm^2$ and 91.317 W, respectively. The photonic components of a full Lightning chip include modulators, photodetectors, and a comb laser. We obtain the unit area of photonic components from prior work [83, 105, 112]. We estimate the power consumption of its photonic components to be 40 atto Joules/MAC × 97 GHz × 576 MACs = 0.00223 W based on prior work's report on 40 atto Joule per MAC [101]. Combining the area of digital and photonic components together, the total area of Lightning's chip is 2028.839 $mm^2$, which is 2.55× smaller than the area of an Intel Stratix 10 FPGA used in Brainwave (5180 $mm^2$) [10].

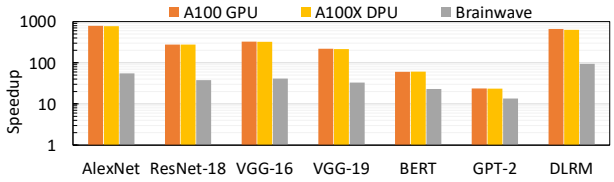



**Figure 21: [Simulation] Lightning's average inference serve time speedup compared to state-of-the-art accelerators.**

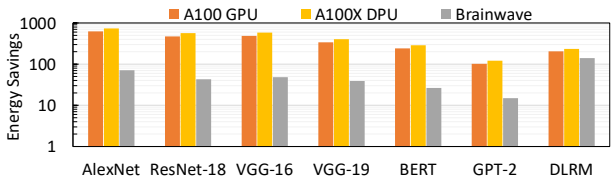



**Figure 22: [Simulation] Lightning's average energy consumption savings compared to state-of-the-art accelerators.**

Moreover, the total power of Lightning's chip is 91.319 W, which is 1.37× and 3.29× less power than a Brainwave smartNIC (125 W) and an Nvidia's A100X DPU (300 W), respectively.

**End-to-end energy consumption per MAC.** Using the projected chip power above, we now compare Lightning's end-to-end energy consumption per MAC with the state-of-the-art digital accelerators. For each accelerator, we first divide its total power by its total number of MAC units to obtain the power of a MAC operation (denoted by $P$). We then compute the energy consumption per MAC by dividing $P$ by the accelerator's clock frequency. Note that this calculation represents a system-level end-to-end metric. As a result, it considers the energy consumption for the MAC operation, as well as the accelerator's control and memory access logic. Table 3 shows that for a single MAC operation, Lightning consumes 16.09×, 15.69×, 18.83×, and 3.19× less energy compared to Nvidia P4 GPU, A100 GPU, A100X DPU, and Microsoft Brainwave, respectively.

## 9 LARGE-SCALE SIMULATIONS

This section evaluates the performance of Lightning using simulations. We begin by describing our simulation environment. Then, we compare the average inference serve time and energy consumption of Lightning with several state-of-the-art benchmarks.

**Event-driven simulator.** We develop a discrete-time event-driven simulator that considers dynamic DNN inference requests arrivals. To ensure fast resource allocation at nanosecond speed without slowing down the dataflow, we decompose each DNN inference request into a series of layer-wise vector dot product tasks according to the DNN model's computation DAG. We then map these tasks to photonic vector dot product cores (for Lightning) or digital MAC cores (for benchmarks) using a round-robin scheduler with a First-In-First-Out (FIFO) queue.

**DNN models.** We evaluate seven real-world large DNN models: AlexNet [75], ResNet18 [65], VGG16 [44], VGG19 [70], BERT [46], GPT-2 [93], and DLRM [87]. All DNN models' inference queries have an equal probability of occurrence.

**Benchmarks.** We compare the performance of the Lightning chip capable of performing 576 photonic MAC operations at 97 GHz (§8) to smartNIC-based solutions such as Nvidia A100X DPU [1] and Microsoft Brainwave [51]. We also consider server-based inference systems with Nvidia A100 [3].

**Datapath latency.** We define datapath latency as the time it takes to start the DNN's first-layer computation from the moment it arrives at the NIC. For smartNICs like A100X DPU and Brainwave, we use an ideal (zero) datapath latency because they can process the inference query on the NIC. For GPUs, we measure the real-world datapath latency for serving each DNN model with the A100 GPU using our Nvidia Triton servers [4]. For Lightning, we consider an extra datapath latency of 193 ns per DNN layer measured from our prototype.[4] We then calculate each model's datapath latency by multiplying this latency number with the number of layers for different DNNs. Table 6 (Appendix F) lists the datapath latency we use for each model in our simulations.

**Inference request arrivals.** We use a Poisson distribution for inference request arrivals such that the average utilization of the most congested accelerator is ≈90%-99%. Pushing the inference request arrival rate large will incur significant queuing overheads among inference queries because the accelerators are fully utilized. In our simulations, we simulate ten randomized-generated inference request traces and report average gains across all traces.

**Inference serve time.** We define the inference serve time as the time it generates the result of a DNN inference query from the moment it arrives at the NIC. Therefore, it contains the datapath latency ($t_d$), the queuing latency ($t_q$), and the computation latency ($t_c$). The datapath latency is described above. The queuing latency is the amount of time when the inference query is temporarily stored on the local host's DRAM when the accelerator is busy. The computation latency is the time it takes to perform computation on the accelerator. When the inference request arrival rate is low, the inference serve time is dominated by the datapath latency and the computation latency. As the inference request arrival rate increases, the queuing latency gradually grows because the accelerators become highly utilized and may not always have available computing cores to serve the incoming inference queries. Figure 21 presents the average speedup of Lightning compared to our digital inference benchmarks across different DNNs. The figure shows that Lightning improves the average inference serve time by 337×, 329×, and 42× compared to Nvidia A100 GPU, Nvidia A100X DPU, and Microsoft Brainwave, respectively.

**Energy consumption.** There are three major sources of energy consumption as each accelerator serves inference requests: packet processing the NIC, inference requests queuing on the DRAM, and DNN computation on the accelerator. We calculate the datapath energy consumption of an A100 GPU by multiplying $t_d$ with the power of a 100 Gbps NIC card [28]. For Lightning, the computation energy contains the datapath energy consumption because the packet I/O function is integrated into Lightning's datapath. We then calculate the energy consumption of the computation step by multiplying $t_c$ with the power of each accelerator, and the energy consumption of DRAM queuing by multiplying $t_q$ with the power

[4]This datapath latency covers the time it takes to perform Lightning-specific functions like DACs, ADCs, and count-action modules.

consumption of DRAM [29]. We then aggregate these two energy numbers to obtain the total energy consumption for each DNN on different accelerators. Figure 22 presents the energy savings of serving inference queries of different DNN models, showing that Lightning improves the average energy consumption by 352×, 419×, and 54× compared to Nvidia A100 GPUs, Nvidia A100X DPU, and Microsoft Brainwave, respectively.

## 10 DISCUSSION

**Lightning cost.** It is extremely challenging to provide accurate cost projections for Lightning. Following prior work [71], we provide a cost estimation based on chip area. As shown in §8, the photonic components in a Lightning chip occupy 1500.01 $mm^2$ area. The cost of manufacturing this chip on LioniX silicon-nitride multi-wafer run is ≈$25,312.5 (the cost of 4 samples of 200 $mm^2$ is ≈13,500) based on the 2023 Europractice pricelist [27]. Assuming this cost will drop by 10× with mass production, we anticipate the cost of Lightning's photonic components to be ≈$2,531.25. We further estimate the cost of Lightning's electronic components to be ≈$108.7. This estimate is based on TSMC's 7 nm wafer cost ($10,000) with 80% yield in 2022 [19]. A standard 300 mm diameter silicon wafer holds ≈115 Lightning chips occupying a 609.93 $mm^2$ CMOS chip area (Table 2). Together, we estimate the cost of a Lightning smartNIC to be $2,639.95.

**Beyond 8-bit precision.** In scenarios where more than 8-bit precision is required, we augment Lightning using techniques similar to Microsoft's Floating Point [43]. The key idea is to represent a 32-bit floating point number as four 8-bit numbers. The four 8-bit numbers require four Lightning photonic vector dot product cores with an additional fix-point-to-float converter to be implemented in Lightning's datapath for post-processing. This extension enables Lightning to support high-precision computation beyond 8 bits, with an expected chip area and power increase. In particular, the chip area and power of the photonics part are estimated to scale by 4×, and the area and power of the digital part will increase, but we expect the increase factor to be smaller than four.

## 11 RELATED WORK

**Photonic DNN accelerators.** The concept of photonic computing has been proposed for several decades [48, 49, 58, 84]. Recent photonic computing papers leverage properties such as wavelength multiplexing to perform parallel multiplications [50, 101, 111, 115], spatial parallelism through Mach-Zehnder modulator (MZM) meshes [80, 96], or passive fan-out and coherent detection [61]. There are also previous proposals [40, 81, 92] on hybrid photonic-electronic co-design. However, these hybrid systems are proposed through simulations. Unlike Lightning, these efforts did not take system-level design challenges into consideration. In contrast, Lightning addresses the datapath challenge of photonic computing, with a fully-functional photonic-electronic prototype serving real-time inference requests.

**Digital DNN accelerators.** There is a plethora of prior work on accelerating DNN computation using commodity GPUs [3, 30], custom-designed ASICs [39, 63, 64, 103] or FPGAs [56, 95, 117]. For example, Nvidia Triton serves high-throughput DNN inference queries on modern commodity GPUs connected to NIC through PCIe [4]. Eyeriss designed and taped out a customized ASIC with an energy-efficient dataflow for AlexNet inference [39]. EIE proposed a processing element architecture for compressing sparse neural networks [64]. DNNWeaver proposed a framework for automatically generating a synthesizable DNN accelerator for FPGAs target [95]. Lightning outperforms all these DNN accelerators in terms of computing frequency, hence reducing the inference serve time of real-time inference requests.

**DNN inference on smartNICs.** SmartNICs [51, 99, 107] are prime candidates for processing user-facing inference packets. N3IC [99] proposed to compile binary neural network models to be directly implemented on the data plane of SmartNICs to enable online traffic analysis in a few microseconds. Lightning goes beyond N3IC to achieve not only online traffic classification in one microsecond (Figure 15a) but also enable larger DNN models on the smartNIC. Microsoft Brainwave smartNIC enables real-time DNN inference [51]. In comparison, Lightning outperforms Brainwave in terms of inference serve time and energy because of the reconfigurable count-action datapath with photonic computing cores.

**In-network DNN inference.** There are several prior works that proposed performing DNN inference inside network switches [55, 110, 115] or on edge devices [79, 101]. For example, Taurus proposed to augment switch ASICs for per-packet inference [103]. IOI and NetCast introduced a smart transceiver module with photonic computing capability that plugs into network switches [101, 115]. Lightning is applicable to support these scenarios as well, and we leave extending Lightning for in-network inference use cases to future work.

## 12 CONCLUSION

We propose Lightning, a photonic-electronic smartNIC for serving live machine learning inference requests in datacenters. Lightning uses a novel count-action abstraction to feed traffic from the NIC into the photonic domain without making digital packet processing and data movement a bottleneck. We evaluate Lightning using four platforms: prototype, emulation, chip-level synthesis, and large-scale simulations. This work does not raise any ethical issues.

## ACKNOWLEDGMENT

We thank our shepherd David Maltz and anonymous SIGCOMM reviewers for their insightful comments. Thanks to Ryan Hamerly, Joud Khoury, Weiyang Wang, Moein Khazraee, Saumil Bandyopadhyay, and Lingling Fan for helpful discussions. Special thanks to Yongyi Zhao for assembling the first version of our developer kit. We are grateful to Dustin Nicholes, Jamey Hicks, and Luis Hernandez for their help with FPGA development. Thanks to Arthur Migdal for installing the Nvidia Triton inference server and measuring the datapath latency of DNN models. The authors are supported by DARPA FastNICs 4202290027, Air Force AI Accelerator, ARPA-E ENLITENED PINE DE-AR0000843, NSF CNS-2008624, NSF SHF-2107244, NSF ASCENT-2023468, NSF CAREER-2144766, NSF PPoSS-2217099, NSF CNS-2211382, NSF FuSe-TG-2235466, Sloan fellowship FG-2022-18504, the U.S. Army Research Office through the Institute for Soldier Nanotechnologies (ISN) (W911NF-18-2-0048), and the NSF Center for Quantum Networks.

# REFERENCES


[1] [n. d.]. Nvidia converged accelerators. ([n. d.]). https://www.nvidia.com/content/dam/en-zz/Solutions/gtcf21/converged-accelerator/pdf/datasheet.pdf, year=2022.

[2] 2021. DAC Performance Survey 1997-2021. (2021). https://github.com/pietro-caragiulo/survey-DAC.

[3] 2021. Nvidia A100 GPU. (2021). https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf.

[4] 2021. Nvidia Triton Inference Server. (2021). https://developer.nvidia.com/nvidia-triton-inference-server.

[5] 2022. 10 GHz Intensity Modulator. (2022). https://www.thorlabs.com/thorproduct.cfm?partnumber=LN81S-FC.

[6] 2022. 40 GHz Intensity Modulator, Z-Cut, FC/PC Connectors, 1525 nm - 1605 nm, Small Form Factor Housing . (2022). https://www.thorlabs.com/thorproduct.cfm?partnumber=LNA6112.

[7] 2022. Advanced eXtensible Interface. (2022). https://en.wikipedia.org/wiki/Advanced_eXtensible_Interface,.

[8] 2022. AMBA® AXI-Stream Protocol Specification. (2022). https://developer.arm.com/documentation/ihi0051/a/Interface-Signals/Transfer-signaling/Handshake-process.

[9] 2022. Electro-optic modulator. (2022). https://en.wikipedia.org/wiki/Electro-optic_modulator,.

[10] 2022. Intel Stratix 10 FPGA and SoC Family Plan. (2022). https://www.intel.com/content/www/us/en/docs/programmable/683729/current/fpga-and-soc-family-plan.html.

[11] 2022. Keysight M8100 Series Arbitrary Waveform Generator. (2022). https://www.keysight.com/us/en/products/arbitrary-waveform-generators/m8100-series-arbitrary-waveform-generators.html.

[12] 2022. LMH5401 Evaluation Module. (2022). https://www.ti.com/tool/LMH5401EVM,.

[13] 2022. Mach–Zehnder interferometer . (2022). https://en.wikipedia.org/wiki/Mach%E2%80%93Zehnder_interferometer.

[14] 2022. N3IC github repository. (2022). https://github.com/nec-research/n3ic-nsdi22,.

[15] 2022. Petalinux Tools. (2022). https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html.

[16] 2022. QICK: Quantum Instrumentation Control Kit . (2022). https://github.com/openquantumhardware/qick.

[17] 2022. RFSOC-PYNQ . (2022). http://www.rfsoc-pynq.io/.

[18] 2022. Thorlabs InGaAs Fixed Gain Amplified Detector, 750 - 1650 nm, DC - 9.5 GHz . (2022). https://www.thorlabs.com/thorproduct.cfm?partnumber=PDA8GS.

[19] 2022. TSMC 3 nm Wafer Pricing to Reach $20,000; Next-Gen CPUs/GPUs to be More Expensive. (2022). https://www.techpowerup.com/301393/tsmc-3-nm-wafer-pricing-to-reach-usd-20-000-next-gen-cpus-gpus-to-be-more-expensive.

[20] 2022. UltraScale+ Devices Integrated 100G Ethernet Subsystem v3.1. (2022). https://docs.xilinx.com/v/u/en-US/pg203-cmac-usplus.

[21] 2022. UltraScale™ architecture-based FPGAs Memory IP core. (2022). https://www.xilinx.com/content/dam/xilinx/support/documents/ip_documentation/ultrascale_memory_ip/v1_4/pg150-ultrascale-memory-ip.pdf.

[22] 2022. Verilator. (2022). https://www.veripool.org/verilator/,.

[23] 2022. Zynq UltraScale+ RFSoC. (2022). https://www.xilinx.com/products/silicon-devices/soc/rfsoc.html.

[24] 2022. Zynq UltraScale+ RFSoC RF Data Converter v2.6 Gen 1/2/3 LogiCORE IP Product Guide. (2022). https://docs.xilinx.com/v/u/en-US/pg269-rf-data-converter.

[25] 2022. Zynq UltraScale+ RFSoC ZCU111 Evaluation Kit. (2022). https://www.xilinx.com/products/boards-and-kits/zcu111.html,.

[26] 2023. 125 MS/s 16 bit multi-purpose digitizer. (2023). https://spectrum-instrumentation.com/products/details/M2p5943-x4.php.

[27] 2023. 2023 General Europractice Pricelist. (July 2023). https://europractice-ic.com/schedules-prices-2023/.

[28] 2023. ConnectX 100Gbps SmartNICs. (2023). https://www.nvidia.com/en-us/networking/ethernet-adapters/.

[29] 2023. How Much Power Does Memory Use? (2023). https://www.crucial.com/support/articles-faq-memory/how-much-power-does-memory-use.

[30] 2023. Nvidia Tesla P4 GPU. (2023). https://images.nvidia.com/content/pdf/tesla/184457-Tesla-P4-Datasheet-NV-Final-Letter-Web.pdf.

[31] Hitesh Ballani. 2023. Unlocking the future of computing: The Analog Iterative Machine's lightning-fast approach to optimization. (2023). https://www.microsoft.com/en-us/research/blog/unlocking-the-future-of-computing-the-analog-iterative-machines-lightning-fast-approach-to-optimization/?secret=O92oxp.

[32] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. 2018. Scalable Methods for 8-Bit Training of Neural Networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 5151–5159.

[33] Tamal Bose and Francois Meyer. 2003. *Digital signal and image processing*. John Wiley & Sons, Inc.

[34] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM '13)*. Association for Computing Machinery, New York, NY, USA, 99–110. https://doi.org/10.1145/2486001.2486011

[35] E Oran Brigham. 1988. *The fast Fourier transform and its applications*. Prentice-Hall, Inc.

[36] Madhukar Budagavi, Arild Fuldseth, Gisle Bjøntegaard, Vivienne Sze, and Mangesh Sadafale. 2013. Core transform design in the high efficiency video coding (HEVC) standard. *IEEE Journal of Selected Topics in Signal Processing* 7, 6 (2013), 1029–1041.

[37] Maurizio Burla, Claudia Hoessbacher, Wolfgang Heni, Christian Haffner, Yuriy Fedoryshyn, Dominik Werner, Tatsuhiko Watanabe, Hermann Massler, Delwin L Elder, Larry R Dalton, et al. 2019. 500 GHz plasmonic Mach-Zehnder modulator enabling sub-THz microwave photonics. *Apl Photonics* 4, 5 (2019).

[38] Kevin K Chang, Abhijith Kashyap, Hasan Hassan, Saugata Ghose, Kevin Hsieh, Donghyuk Lee, Tianshi Li, Gennady Pekhimenko, Samira Khan, and Onur Mutlu. 2016. Understanding latency variation in modern DRAM chips: Experimental characterization, analysis, and optimization. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*. 323–336.

[39] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits* 52, 1 (2016), 127–138.

[40] Qixiang Cheng, Jihye Kwon, Madeleine Glick, Meisam Bahadori, Luca P Carloni, and Keren Bergman. 2020. Silicon photonics codesign for deep learning. *Proc. IEEE* 108, 8 (2020), 1261–1282.

[41] Jin Hee Cho, Jihwan Kim, Woo Young Lee, Dong Uk Lee, Tae Kyun Kim, Heat Bit Park, Chunseok Jeong, Myeong-Jae Park, Seung Geun Baek, Seokwoo Choi, et al. 2018. A 1.2 V 64Gb 341GB/S HBM2 stacked DRAM with spiral point-to-point TSV structure and improved bank group data control. In *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 208–210.

[42] Devin Coldewey. 2023. Lightmatter's photonic AI hardware is ready to shine with $154M in new funding. (May 2023). https://techcrunch.com/2023/05/31/lightmatters-photonic-ai-hardware-is-ready-to-shine-with-154m-in-new-funding/.

[43] Bita Darvish Rouhani, Daniel Lo, Ritchie Zhao, Ming Liu, Jeremy Fowers, Kalin Ovtcharov, Anna Vinogradsky, Sarah Massengill, Lita Yang, Ray Bittner, et al. 2020. Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point. *Advances in neural information processing systems* 33 (2020), 10271–10281.

[44] Abhipraya Kumar Dash. [n. d.]. VGG-16 Architecture. ([n. d.]). https://iq.opengenus.org/vgg16/.

[45] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[46] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805

[47] Albert Einstein. 1905. On a heuristic point of view concerning the production and transformation of light. *Annalen der Physik* (1905), 1–18.

[48] Nabil H. Farhat, Demetri Psaltis, Aluizio Prata, and Eung Paek. 1985. Optical implementation of the Hopfield model. *Appl. Opt.* 24, 10 (May 1985), 1469–1475. https://doi.org/10.1364/AO.24.001469

[49] Dror G Feitelson. 1988. Optical Computing: A survey for computer scientists. (1988).

[50] J. Feldmann, N. Youngblood, M. Karpov, H. Gehring, X. Li, M. Stappers, M. Le Gallo, X. Fu, A. Lukashchuk, A. S. Raja, J. Liu, C. D. Wright, A. Sebastian, T. J. Kippenberg, W. H. P. Pernice, and H. Bhaskaran. 2021. Parallel convolutional processing using an integrated photonic tensor core. *Nature* 589, 7840 (2021), 52–58. https://doi.org/10.1038/s41586-020-03070-1

[51] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steven K. Reinhardt, Adrian M. Caulfield, Eric S. Chung, and Doug Burger. 2018. A Configurable Cloud-Scale DNN Processor for Real-Time AI. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 1–14. https://doi.org/10.1109/ISCA.2018.00012

[52] Alexander L Gaeta, Michal Lipson, and Tobias J Kippenberg. 2019. Photonic-chip-based frequency combs. *nature photonics* 13, 3 (2019), 158–169.

[53] Robert Gallager. 1962. Low-density parity-check codes. *IRE Transactions on information theory* 8, 1 (1962), 21–28.

[54] Sahaj Garg, Joe Lou, Anirudh Jain, and Mitchell Nahmias. 2021. Dynamic precision analog computing for neural networks. *arXiv preprint arXiv:2102.06365* (2021).

[55] Manya Ghobadi, Zhizhen Zhong, Weiyang Wang, Alexander Sludds, Ryan Hamerly, Liane Bernstein, and Dirk Englund. 2021. In-network optical inference. (May 20 2021). US Patent 63,191,120.

[56] Soroush Ghodrati, Byung Hoon Ahn, Joon Kyung Kim, Sean Kinzer, Brahmendra Reddy Yatham, Navateja Alla, Hardik Sharma, Mohammad Alian, Eiman Ebrahimi, Nam Sung Kim, Cliff Young, and Hadi Esmaeilzadeh. 2020. Planaria: Dynamic Architecture Fission for Spatial Multi-Tenant Acceleration of Deep Neural Networks. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 681–697. https://doi.org/10.1109/MICRO50266.2020.00062

[57] Heedong Goh and Andrea Alù. 2022. Nonlocal Scatterer for Compact Wave-Based Analog Computing. *Phys. Rev. Lett.* 128 (Feb 2022), 073201. Issue 7. https://doi.org/10.1103/PhysRevLett.128.073201

[58] J W Goodman, A R Dias, and L M Woody. 1978. Fully parallel, high-speed incoherent optical method for performing discrete Fourier transforms. *Opt. Lett.* 2, 1 (Jan. 1978), 1–3.

[59] Kasper Groes and Albin Ludvigsen. 2023. ChatGPT's Electricity Consumption. (March 2023). https://towardsdatascience.com/chatgpts-electricity-consumption-7873483feac4.

[60] Christian Haffner, Daniel Chelladurai, Yuriy Fedoryshyn, Arne Josten, Benedikt Baeuerle, Wolfgang Heni, Tatsuhiko Watanabe, Tong Cui, Bojun Cheng, Soham Saha, et al. 2018. Low-loss plasmon-assisted electro-optic modulator. *Nature* 556, 7702 (2018), 483–486.

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

[62] Richard W Hamming. 1950. Error detecting and error correcting codes. *The Bell system technical journal* 29, 2 (1950), 147–160.

[63] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. 2017. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 75–84.

[64] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 243–254.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[66] Mingbo He, Mengyue Xu, Yuxuan Ren, Jian Jian, Ziliang Ruan, Yongsheng Xu, Shengqian Gao, Shihao Sun, Xueqin Wen, Lidan Zhou, et al. 2019. High-performance hybrid silicon and lithium niobate Mach–Zehnder modulators for 100 Gbit s- 1 and beyond. *Nature Photonics* 13, 5 (2019), 359–364.

[67] Philip Jacobson, Mizuki Shirao, Kerry Yu, Guan-Lin Su, and Ming C. Wu. 2022. Hybrid Convolutional Optoelectronic Reservoir Computing for Image Recognition. *Journal of Lightwave Technology* 40, 3 (2022), 692–699. https://doi.org/10.1109/JLT.2021.3124520

[68] Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, Cliff Young, Zongwei Zhou, and David Patterson. 2021. Ten Lessons From Three Generations Shaped Google's TPUv4i : Industrial Product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 1–14. https://doi.org/10.1109/ISCA52012.2021.00010

[69] Norman P Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, et al. 2021. Ten lessons from three generations shaped google's tpuv4i: Industrial product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1–14.

[70] Aakash Kaushik. [n. d.]. VGG-19 Architecture. ([n. d.]). https://iq.opengenus.org/vgg19-architecture/.

[71] Mehrdad Khani, Manya Ghobadi, Mohammad Alizadeh, Ziyi Zhu, Madeleine Glick, Keren Bergman, Amin Vahdat, Benjamin Klenk, and Eiman Ebrahimi. 2021. SiP-ML: high-bandwidth optical network interconnects for machine learning training. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 657–675.

[72] Prashanta Kharel, Christian Reimer, Kevin Luke, Lingyan He, and Mian Zhang. 2021. Breaking voltage–bandwidth limits in integrated lithium niobate modulators using micro-structured electrodes. *Optica* 8, 3 (2021), 357–363.

[73] Taehyun Kim, Deondre Martin Ng, Junzhi Gong, Youngjin Kwon, Minlan Yu, and KyoungSoo Park. 2023. Rearchitecting the TCP Stack for I/O-Offloaded Content Delivery. In *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022*. USENIX.

[74] Ueli Koch, Christopher Uhl, Horst Hettrich, Yuriy Fedoryshyn, Claudia Hoessbacher, Wolfgang Heni, Benedikt Baeuerle, Bertold I Bitachon, Arne Josten, Masafumi Ayata, et al. 2020. A monolithic bipolar CMOS electronic–plasmonic high-speed transmitter. *Nature Electronics* 3, 6 (2020), 338–345.

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

[76] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[77] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[78] Mo Li and Hong X Tang. 2019. Strong pockels materials. *Nature Materials* 18, 1 (2019), 9–11.

[79] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. 2020. Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems* 33 (2020), 11711–11722.

[80] Xing Lin, Yair Rivenson, Nezih T Yardimci, Muhammed Veli, Yi Luo, Mona Jarrahi, and Aydogan Ozcan. 2018. All-optical machine learning using diffractive deep neural networks. *Science* 361, 6406 (2018), 1004–1008.

[81] Weichen Liu, Wenyang Liu, Yichen Ye, Qian Lou, Yiyuan Xie, and Lei Jiang. 2019. Holylight: A nanophotonic accelerator for deep learning in data centers. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1483–1488.

[82] Susan Luschas, Richard Schreier, and Hae-Seung Lee. 2004. Radio frequency digital-to-analog converter. *IEEE Journal of Solid-State Circuits* 39, 9 (2004), 1462–1467.

[83] Dennis Maes, Luis Reis, Stijn Poelman, Ewoud Vissers, Vanessa Avramovic, Mohammed Zaknoune, Gunther Roelkens, Sam Lemey, Emilien Peytavit, and Bart Kuyken. 2022. High-speed photodiodes on silicon nitride with a bandwidth beyond 100 Ghz. In *CLEO: Science and Innovations*. Optica Publishing Group, SM3K–3.

[84] Peter L. McMahon. 2023. The physics of optical computing. *arXiv preprint arXiv:2308.00088* (2023).

[85] Microsoft. 2023. Project AIM (Analog Iterative Machine). (2023). https://www.microsoft.com/en-us/research/project/aim/.

[86] Nour Moustafa and Jill Slay. 2015. 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, 1–6.

[87] 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. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).

[88] RL Nguyen, AM Castrillon, A Fan, A Mellati, Benjamin T Reyes, Cindra Abidin, E Olsen, F Ahmad, Geoff Hatcher, J Chana, et al. 2021. 8.6 A Highly Reconfigurable 40-97GS/s DAC and ADC with 40GHz AFE Bandwidth and Sub-35fJ/conv-step for 400Gb/s Coherent Optical Applications in 7nm FinFET. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64. IEEE, 136–138.

[89] Tan Nguyen, Samuel Williams, Marco Siracusa, Colin MacLean, Douglas Doerfler, and Nicholas J Wright. 2020. The performance and energy efficiency potential of FPGAs in scientific computing. In *2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 8–19.

[90] Mike O'Connor, Niladrish Chatterjee, Donghyuk Lee, John Wilson, Aditya Agrawal, Stephen W Keckler, and William J Dally. 2017. Fine-grained DRAM: Energy-efficient DRAM for extreme bandwidth systems. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. 41–54.

[91] OZ Optics. 2021. Super modulator bias controller. (June 2021). https://www.ozoptics.com/ALLNEW_PDF/DTS0165.pdf.

[92] Jiaxin Peng, Yousra Alkabani, Shuai Sun, Volker J Sorger, and Tarek El-Ghazawi. 2020. Dnnara: A deep neural network accelerator using residue arithmetic and integrated photonics. In *Proceedings of the 49th International Conference on Parallel Processing*. 1–11.

[93] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. (2019).

[94] Kim Roberts, Qunbi Zhuge, Inder Monga, Sebastien Gareau, and Charles Laperle. 2017. Beyond 100 Gb/s: capacity, flexibility, and network optimization. *Journal of Optical Communications and Networking* 9, 4 (2017), C12–C24.

[95] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. 2016. From high-level deep neural models to FPGAs. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 1–12.

[96] Yichen Shen, Nicholas C Harris, Scott Skirlo, Mihika Prabhu, Tom Baehr-Jones, Michael Hochberg, Xin Sun, Shijie Zhao, Hugo Larochelle, Dirk Englund, et al. 2017. Deep learning with coherent nanophotonic circuits. *Nature Photonics* 11, 7 (2017), 441–446.

[97] Shawn Yohanes Siew, Bo Li, Feng Gao, Hai Yang Zheng, Wenle Zhang, Pengfei Guo, Shawn Wu Xie, Apu Song, Bin Dong, Lian Wee Luo, et al. 2021. Review of

silicon photonics technology and platform development. *Journal of Lightwave Technology* 39, 13 (2021), 4374–4389.

[98] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[99] 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 22)*. USENIX Association, Renton, WA, 513–533.

[100] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. 2018. Classifying IoT devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing* 18, 8 (2018), 1745–1759.

[101] Alexander Sludds, Saumil Bandyopadhyay, Zaijun Chen, Zhizhen Zhong, Jared Cochrane, Liane Bernstein, Darius Bunandar, P Ben Dixon, Scott Hamilton, Matthew Streshinsky, Ari Novack, Tom Baehr-Jones, Michael Hochberg, Manya Ghobadi, Ryan Hamerly, and Dirk Englund. 2022. Delocalized Photonic Deep Learning on the Internet's Edge. *Science* 378, 6617 (2022), 270–276. https://doi.org/10.1126/science.abq8271

[102] Alexander Sludds, Ryan Hamerly, Saumil Bandyopadhyay, Zhizhen Zhong, Zaijun Chen, Liane Bernstein, Manya Ghobadi, and Dirk Englund. 2022. Demonstration of WDM-Enabled Ultralow-Energy Photonic Edge Computing, In Optical Fiber Communication Conference (OFC) 2022. *Optical Fiber Communication Conference (OFC) 2022*, Th3A.3. https://doi.org/10.1364/OFC.2022.Th3A.3

[103] Tushar Swamy, Alexander Rucker, Muhammad Shahbaz, Ishan Gaur, and Kunle Olukotun. 2022. 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 (ASPLOS 2022)*. Association for Computing Machinery, New York, NY, USA, 1099–1114.

[104] Cheng Wang, Mian Zhang, Xi Chen, Maxime Bertrand, Amirhassan Shams-Ansari, Sethumadhavan Chandrasekhar, Peter Winzer, and Marko Lončar. 2018. Integrated lithium niobate electro-optic modulators operating at CMOS-compatible voltages. *Nature* 562, 7725 (2018), 101–104.

[105] Cheng Wang, Mian Zhang, Xi Chen, Maxime Bertrand, Amirhassan Shams-Ansari, Sethumadhavan Chandrasekhar, Peter Winzer, and Marko Lončar. 2018. Integrated lithium niobate electro-optic modulators operating at CMOS-compatible voltages. *Nature* 562, 7725 (2018), 101–104. https://doi.org/10.1038/s41586-018-0551-y

[106] Tianyu Wang, Shi-Yuan Ma, Logan G Wright, Tatsuhiro Onodera, Brian C Richard, and Peter L McMahon. 2022. An optical neural network using less than 1 photon per multiplication. *Nature Communications* 13, 1 (2022), 1–8.

[107] Zeke Wang, Hongjing Huang, Jie Zhang, Fei Wu, and Gustavo Alonso. 2022. FpgaNIC: An FPGA-based Versatile 100Gb SmartNIC for GPUs. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 967–986. https://www.usenix.org/conference/atc22/presentation/wang-zeke

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

[109] AMD Xilinx. 2021. Virtex UltraScale+ FPGA Data Sheet: DC and AC Switching Characteristics. (2021). https://docs.xilinx.com/v/u/en-US/ds923-virtex-ultrascale-plus.

[110] 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*. 25–33.

[111] Xingyuan Xu, Mengxi Tan, Bill Corcoran, Jiayang Wu, Andreas Boes, Thach G Nguyen, Sai T Chu, Brent E Little, Damien G Hicks, Roberto Morandotti, et al. 2021. 11 TOPS photonic convolutional accelerator for optical neural networks. *Nature* 589, 7840 (2021), 44–51.

[112] Xiaoxiao Xue, Pei-Hsun Wang, Yi Xuan, Minghao Qi, and Andrew M. Weiner. 2017. Microresonator Kerr frequency combs with high conversion efficiency. *Laser & Photonics Reviews* 11, 1 (2017), 1600276. https://doi.org/10.1002/lpor.201600276 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/lpor.201600276

[113] Javier Yanes. 2020. Optical Computing: Solving Problems at the Speed of Light. (Feb. 2020). https://www.bbvaopenmind.com/en/technology/future/optical-computing-solving-problems-at-the-speed-of-light/.

[114] Zhipeng Zhao, Hugo Sadok, Nirav Atre, James C Hoe, Vyas Sekar, and Justine Sherry. 2020. Achieving 100Gbps intrusion prevention on a single server. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 1083–1100.

[115] Zhizhen Zhong, Weiyang Wang, Manya Ghobadi, Alexander Sludds, Ryan Hamerly, Liane Bernstein, and Dirk Englund. 2021. IOI: In-network Optical Inference. In *Proceedings of the ACM SIGCOMM 2021 Workshop on Optical Systems*. 18–22.

[116] Hailong Zhou, Jianji Dong, Junwei Cheng, Wenchan Dong, Chaoran Huang, Yichen Shen, Qiming Zhang, Min Gu, Chao Qian, Hongsheng Chen, et al. 2022. Photonic matrix multiplication lights up photonic accelerator and beyond. *Light: Science & Applications* 11, 1 (2022), 30.

[117] Yu Zhu, Zhenhao He, Wenqi Jiang, Kai Zeng, Jingren Zhou, and Gustavo Alonso. 2021. Distributed recommendation inference on fpga clusters. In *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 279–285.

# APPENDIX

Appendices are supporting material that has not been peer-reviewed.

## A PHOTONIC SETUP CALIBRATION

Though photonic vector dot products are calculated in the photonic domain, their input and output voltages are in the electrical domain. For a photonic vector dot product core to perform faithful and reproducible computation, it is important to derive the transfer functions that first encode a digital number into the light intensities through modulators, and then decode the light intensities detected by the photodetector into digital bits. To do so, the calibration system should answer the following questions to perform accurate computation:

- How should we encode data from *digital bits* to *analog light intensities* on optical modulators?
- How should we decode data from *analog light intensities* back to *digital bits* on photodetectors?

**Encoding scheme on modulators.** Optical modulators follow the rule of Mach–Zehnder interferometers [13] and hence translate input optical voltages $V_0$ into light intensities $I_0$. The transfer function of the optical modulator is a repetitive sinusoid wave $I_0 = I_{max} \sin(V_0 + V_{DC}) + I_{max}$ where $I_{max}$ is the optical extinction ratio of the maximum and minimum light intensities [9]. In Lightning, we are encoding using one of its monotonic periods from the minimum light intensity 0 to the maximum light intensity $I_{max}$. By feeding a series of input voltages $V_0$ sweeping from the minimum to the maximum FPGA DAC output voltage, into the optical modulator and measuring the modulator output light intensity $I_0$, we fit a polynomial function $f_{MOD}$ that maps the modulator output light intensity $I$ to any applied input voltage $v$: $I = f_{MOD}(v)$.

**Decoding scheme after photodetectors.** Photodetectors work under the law of Einstein's photoelectric effect, stating that the intensity of the output current is proportional to the intensity of incident light [47]. Therefore, we use a linear mapping $f_{PD}$ from photodetector incoming light intensity $I$ to ADC digital readout value $r$, using $r_{max} \rightarrow I_{max}$ and $r_{min} \rightarrow I_{min}$, where $I_{max}, I_{min}$ are the maximum and minimum intensity values measured, and $r_{max}, r_{min}$ are the maximum and minimum digital numbers: $r = f_{PD}(I)$.

## B FURTHER DETAILS ON THE PROTOTYPE

As we discussed in §6, we build a photonic computing prototype for Lightning to demonstrate real-time machine learning inference at the record-breaking 4.055 GHz computing frequency. In addition to the setup already explained in §6, there are two more pieces of equipment that enable the prototype: the modulator bias controller and the RF amplifier. Note that these devices are not fundamental to photonic computing. We further discuss possible paths to replace or alter them in the next version of our prototype.

**Modulator bias voltage determination.** The Lithium Niobate Mach-Zehnder modulator [5] used in the Lightning testbed has two electrical inputs: bias voltage and signal voltage. Based on the principles of optical modulators, the transfer function from the input signal voltage to the output intensity can be modeled as a sine function, biased by the bias voltage. As introduced in §2, we use modulators to perform amplitude-modulated photonic multiplication, and a photodetector to detect the intensity of light. Therefore, the intensity of light hitting the photodetector should be proportional to the multiplication result. In the principle of multiplication, if any element of the multiplication is zero, the product becomes zero. Therefore, we should set the bias voltage of both modulators to achieve their *max extinction ratio*, such that no (or minimal) light can go through the modulator, creating a zero light intensity on the photodetector side. In our experiment, we tap 1% light at each modulator's output port for bias voltage determination purposes. To identify the bias voltage that achieves the *max extinction ratio* of the modulators, we first use our Python API to automatically sweep the bias voltage from -9V to 9V on both modulators and read the photodetector output to derive both modulators' transfer function (shown in Figure 23). Then, a packaged bias controller [91] utilizes the tapped 1% modulator output light to lock the operation point during the entire computation process. Moving forward, the bias controller can be extended to support other modulator materials beyond Lithium Niobate.

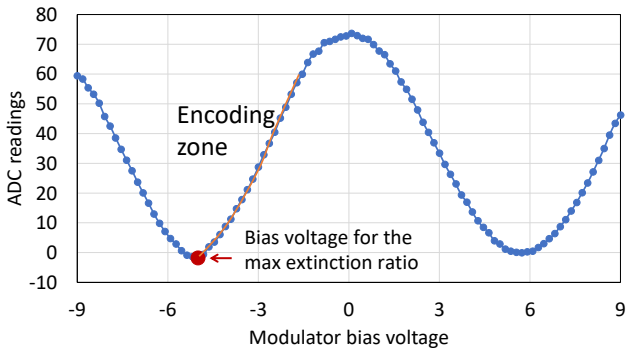



**Figure 23: Sweeping the modulator bias voltage to find the max extinction ratio for Lightning.**

**RF amplifier between the DAC and the modulator.** The modulator used in our prototype has a typical half-wave voltage ($V_\pi$) of 5 V [5]. Therefore, the input RF signal needs to be able to cover the $V_\pi$ to enable intensity modulation. However, the output voltage of the DAC of the RFSoC FPGA we use is only around 1 V [24]. To compensate for the voltage range mismatching while keeping the signal chain to be DC-coupled, we use National Instruments RF amplifiers LHM5401 [12] to amplify the voltage signal generated by the DACs before entering the modulators to gain a larger input voltage range to match the $V_\pi$. Figure 23 shows a 3 V encoding range measured from our prototype. In the future, this RF amplifier can be eliminated by using low-$V_\pi$ electro-optic modulators operating at CMOS-compatible voltages [104].

**RF amplifier between the photodetector and the ADC.** The ADC of the RFSoC FPGA requires a 1.2 V common-mode voltage ($V_{cm}$) to added to the input signal [24]. Therefore, we use the same LHM5401 amplifier to add the $V_{cm}$ to the output voltage of the photodetector [18].

| Experimental demonstrations | Compute frequency | Parallel wavelengths | Bit precision |
|---|---|---|---|
| Feldmann, et al, *Nature*, 2021. [50] | 2 GHz<br>1 kHz | 4<br>200 | 8<br>5 |
| Sludds, et al, *Science*, 2022. [101] | 500 MHz | 16 | 8 |
| Lightning prototype | 4.055 GHz | 2 | 8 |

**Table 4: Comparison with prior experimental demonstrations on photonic machine learning inference.**

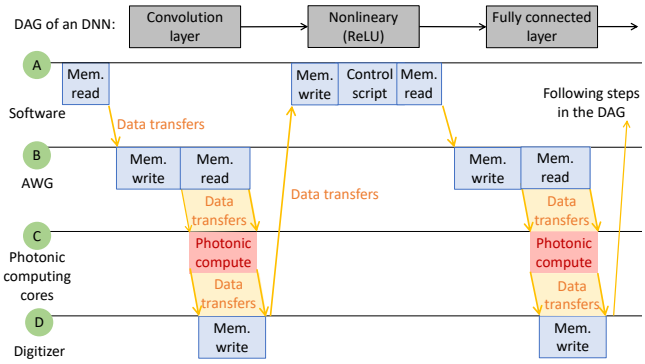



**Figure 24: The coupling of control and data plane results in stop-and-go computations in today's photonic computing.**

## C COMPARISON WITH PRIOR EXPERIMENTAL DEMONSTRATIONS

Table 4 shows a comparison of Lightning to state-of-the-art photonic computing experimental demonstrations. Lightning has the best performance in terms of demonstrated compute frequency. Following prior work, we provide a detailed analysis in §8 on how these numbers scale up to more cores with more wavelengths and project their performance. Note that both Nature'21 [50] and Science'22 [101] handle negative values by doubling the hardware resources or by running the computation process twice, cutting their effective computation frequency by a factor of 2. Lightning solves this problem by separating the sign and absolute values, to compute only the multiplication of absolute values in photonics and re-assemble the signs in the summation stage of the datapath. Hence, Lightning's computing frequency is not affected by negative values.

## D STOP-AND-GO DATAPATH EXAMPLE

We identify the datapath challenge as the bottleneck for performing real-time machine learning inference on photonic computing systems in §3. To present a detailed explanation, Figure 24 illustrates an example of how the stop-and-to approach is blocking the datapath. We consider a simple two-layer DNN with one convolution layer with a non-linear (ReLU) function, followed by a fully-connected layer. To execute the computation DAG of this DNN, the control plane in the state-of-the-art photonic computing approaches first loads the DNN weights and input images from its memory. It then sends these vectors to the AWG device. The AWG device converts digital data into analog voltages and streams them into the photonic computing cores. After the computation, the digitizer receives the output analog voltages from the photonics cores and streams the data back to the controller software to perform the ReLU function in the digital domain. The photonic cores remain idle until the Python program initiates the next computing operation of the fully-connected layer, following a similar stop-and-go fashion. This coupling of the control and data planes creates significant latency bottlenecks in the datapath.

## E ASIC SYNTHESIS DETAILS

As discussed in §8, our goal is to design a Lightning chip to perform 576 MAC operations in one shot at 97 GHz computing frequency. To achieve this goal with a minimal amount of devices, we carefully utilize several favorable features of photonic computing (discussed in §2.1) to scale a photonic multiplication unit (shown in Figure 2a) that only performs one multiplication at one step to support multiple MAC operations at one shot. The first feature is to leverage the photonic broadcasting to only encode the weight matrix once while performing batch inference on multiple input vectors (batch size $B$). The second feature is to send multiple modulated wavelengths carrying into the modulator to enable parallel modulations using a single modulator ($W$ parallel modulations). The third feature is to use a single photodetector to detect the light intensities of multiple wavelengths simultaneously for accumulation (accumulating on $N$ wavelengths). Table 5 presents a detailed analysis on the number of modulators and photodetectors required. Thanks to these three photonic features, we scale the number of MACs per step by $NWB$ times without the need to scale the number of devices with the same factor.

For example, as depicted in Figure 25, we use a comb laser to generate three different wavelengths $\lambda_1$, $\lambda_2$, and $\lambda_3$ and split the light into two identify copies to be used later. Consider a DNN weight matrix with two row vectors $\vec{w}_1 = [w_{11}, w_{12}, ..., w_{1n}]$ and $\vec{w}_2 = [w_{21}, w_{22}, ..., w_{2n}]$ and a batch-2 inference request with two input vectors $\vec{x} = [x_1, x_2, ..., x_n]$ and $\vec{y} = [y_1, y_2, ..., y_n]$. As we explained in §2.1, we encode $\vec{w}_1$ onto one copy of the three wavelengths generated by the comb laser and encode $\vec{w}_1$ onto the second copy of the three wavelengths. In particular, $w_{11}$, $w_{12}$, and $w_{13}$ are simultaneously fed onto modulator$_1$, modulator$_2$, and modulator$_3$, respectively. Similarly, in the next time step, $w_{14}$, $w_{15}$, and $w_{16}$ are fed onto the same set of modulators. We follow the same principle for encoding $\vec{w}_2$. After the six modulators, the two-row vectors are encoded onto six different lightwaves running on three distinct wavelengths. Then, we use three WDM MUXes to combine every two wavelengths together for parallel modulation in the following stage. Note that, when selecting which wavelengths to combine, we only combine wavelengths that are carrying the signal in the same column of the weight matrix. For example, because $\lambda_1$ carries $[w_{11}, w_{14}, ..., w_{1n-2}]$ and $\lambda_3$ carries $[w_{21}, w_{24}, ..., w_{2n-2}]$, they are combined using a WDM MUX. The combined wavelengths are first split into two copies to leverage the favorable feature of photonic broadcasting. Then, one copy is sent into a modulator as its carrier lightwave to perform parallel modulation (the second favorable feature of photonic computing mentioned above). For the modulator$_7$ that takes in $\lambda_1$ and $\lambda_3$, an electrical signal stream representing the corresponding sub-vector

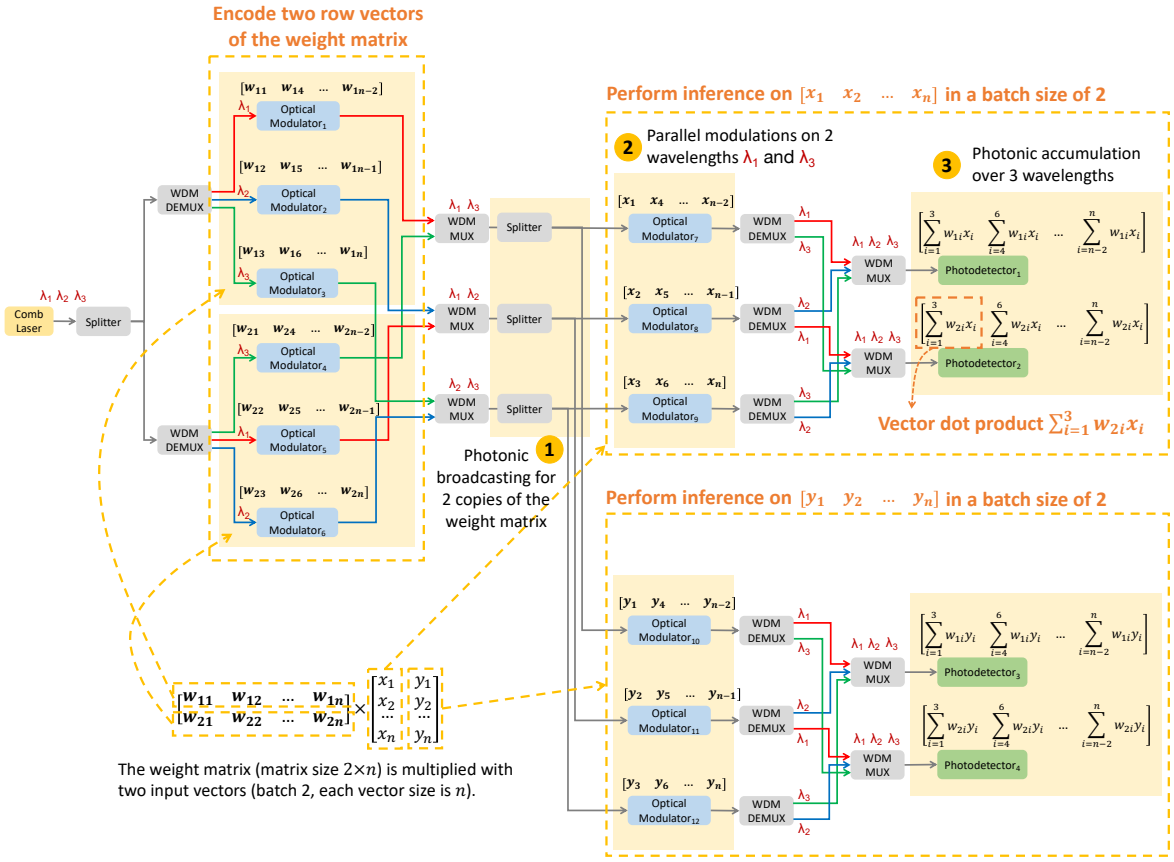



**Figure 25: An example design of the LIGHTNING's photonic vector dot product core with 12 photonic MACs in a single time step.**

| Photonic vector dot product core architecture | Computing primitive | MACs per time step | Modulators for encoding weight matrix | Modulators for encoding input vector | Photodetectors for accumulation | Total distinct wavelengths |
|---|---|---|---|---|---|---|
| Vector multiplication unit (Figure 2a) | Scalar multiplication | 1 multiplication | 1 | 1 | 1 | 1 |
| Vector dot product core accumulating on $N$ wavelengths (Figure 2c) | Vector dot product | $N$ multiplication, $N$ accumulation | $N$ | $N$ | 1 | $N$ |
| Vector dot product core accumulating on $N$ wavelengths with $W$ parallel modulations | Matrix-vector products | $NW$ multiplication, $NW$ accumulation | $NW$ | $N$ | $W$ | $max(N, W)$ |
| Vector dot product core accumulating on $N$ wavelengths with $W$ parallel modulations and inference batch size of $B$ (Figure 25) | Matrix multiplication | $NWB$ multiplication, $NWB$ accumulation | $NW$ | $NB$ | $WB$ | $max(N, W)$ |

**Table 5: Different photonic vector dot product core architectures.**

$[x_1, x_4, ..., x_{n-2}]$ of the input vector $\vec{x}$ is applied to. The output lightwave intensities of modulator$_7$ become proportional to the element-wise product of the input vectors, $[w_{11}x_1, w_{14}x_4, ..., w_{1n-2}x_{n-2}]$ on $\lambda_1$ and $[w_{21}x_1, w_{24}x_4, ..., w_{2n-2}x_{n-2}]$ on $\lambda_2$ ($W = 2$). modulator$_8$ and modulator$_9$ follow the same operating principle. Finally, we DEMUX and MUX the output of modulator$_7$, modulator$_8$ and modulator$_9$, and reassembly the wavelengths such that the three wavelengths ($\lambda_1$, $\lambda_2$, and $\lambda_3$) that are carrying the results of the same vector dot product will be sent to one photodetector for accumulation (the third favorable feature of photonic computing mentioned above). On the photodetector$_1$ output port, we receive a voltage proportional to the vector dot product results $[w_{11}, w_{12}, w_{13}] \cdot [x_1, x_2, x_3] = \sum_{i=1}^{3} w_{1i}x_i$ ($N = 3$). Similarly, photodetector$_2$ returns $\sum_{i=1}^{3} w_{2i}x_i$ at the same time. The LIGHTNING datapath's adder tree module (§5.3) will be responsible for further accumulating these partial vector dot products and assembling them with appropriate signs to complete the full vector dot product. Similarly, the second copy of the weights will perform the same operation on another input vector of this batch $\vec{y}$ and returns a stream of vector dot product results on photodetector$_3$ and photodetector$_4$ ($B = 2$). Putting it together, the setup shown in Figure 25 is capable of performing $NWB = 2 \times 3 \times 2 = 12$ MACs at one time step.

| DNN | Model Size (MB) | Inference query Size (KB) | Dataset | Type | Datapath latency used in simulation ($\mu$s) | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Lightning | A100 GPU | A100X DPU | Brainwave |
| AlexNet [75] | 233 | 150 | ImageNet | Vision | 1.544 | 581 | 0 | 0 |
| ResNet18 [65] | 45 | 150 | ImageNet | Vision | 4.053 | 615 | 0 | 0 |
| VGG16 [98] | 528 | 150 | ImageNet | Vision | 3.088 | 607 | 0 | 0 |
| VGG19 [98] | 548 | 150 | ImageNet | Vision | 3.667 | 596 | 0 | 0 |
| BERT (Large) [46] | 1380 | 5.12 | Synthetic | Language | 32.617 | 1176 | 0 | 0 |
| GPT-2 (XL) [93] | 6263 | 10.24 | Synthetic | Language | 65.234 | 6605 | 0 | 0 |
| DLRM [87] | 12400 | 5.12 | Synthetic | Recommendation | 1.544 | 13210 | 0 | 0 |

**Table 6: DNN models and datapath latency used in §9 for simulations.**

## F SIMULATION SETTINGS

Table 6 shows the model size, inference query size, and dataset of the DNN models used in §9. There are multiple versions of BERT and GPT-2 models. Specifically, we use BERT (Large) and GPT-2 (Extra Large) in our simulation. Table 6 also specifies the datapath latency of individual DNNs and processors used in our simulation. For A100 GPUs, we measure the real-world inference serving latency for different DNNs using an Nvidia Triton server [4], and obtain the datapath latency by subtracting compute latency from end-to-end latency. For A100X DPU and Brainwave, we assume an ideal scenario and use zero datapath latency, even though these two devices also incur packet parsing and model loading overheads. For Lightning, we measured the datapath latency for a single DNN layer as 193 ns using our prototype. We calculate the model datapath latency by multiplying this latency number with the number of layers for different DNNs. Note that within a single DNN model, when multiple layers can be processed in parallel, we apply the single-layer datapath latency only once. This characteristic is applicable to BERT, GPT-2, and DLRM models.

## G LIGHTNING DEVELOPER KIT

**Open-source developer kit.** Today's lab devices used in photonic computing demonstrations are prohibitively expensive, creating a barrier to entry for the SIGCOMM community to experiment with real-time photonic computing systems (§3). To lower this barrier, we built an academic developer kit with all off-the-shelf components and 3D printing parts. Our developer kit is operational using Lightning's Python API. A photo of our developer kit is shown in Figure 26. The developer kit is designed to be "plug-and-play" such that a developer without deep knowledge in photonics and FPGAs can get started easily. The detailed documentation about our developer kit is available at https://lightning.mit.edu.

**Benchmarking photonic vector dot product with the Python API.** We build a Python API based on the PYNQ [17] and QICK [16] libraries on the Xilinx Zynq RFSoC FPGA platform. We show a screenshot of hosting a Jupyter notebook server on the embedded Linux kernel of the FPGA board and use the Lightning Python API to test the photonic MAC functionality of Lightning. Developers can easily import backend classes of Lightning to interact with the developer kit in real-time. For instance, in Figure 27, we are using the Python API to calculate photonic MACs. We first import the Lightning python libraries that enables users to interact with photonic vector dot product cores. In our current prototype (§6), our photonic vector dot product core has two wavelengths, hence performs two MAC operations at one step. We normalize the input numbers within 0 to 1. The four inputs numbers for calculations are $x_1 = 0.85$, $w_1 = 0.26$, $x_2 = 0.5$, and $w_2 = 0.93$. The photonic vector dot product computes the result of $x_1w_1 + x_2w_2$. In this case, the photonic vector dot product core returns a result of 0.664, which represent about 0.6% error with respect to the ground truth result of 0.66.

**Supporting use cases beyond ML.** While this paper focuses on machine learning inference as its primary use case, we believe our evaluation kits will enable the networking community to integrate photonic computing into a wide variety of applications. Besides machine learning inference, Lightning's photonic cores can be used to accelerate video encoding [36], forward error correction (FEC) [53, 62], fast fourier transform (FFT) [35], and image signal processing (ISP) [33]. We look forward to working together with the community to explore these new exciting applications with the Lightning developer kit.


**Figure 26: Lightning's photonic computing developer kit.**


**Figure 27: Benchmarking photonic vector dot product on Lightning's prototype with our Python API.**