A Photonic Accelerator for Deep Learning Training

Yuan Li
Infocomm Technology Cluster, Singapore Institute of Technology, Singapore
Email: li.yuan@singaporetech.edu.sg

***Abstract*—The exponential growth in computational demands driven by emerging deep learning models poses significant scaling challenges for the conventional digital computing systems. Analog computing systems have gained traction as a promising alternative owing to their intrinsic advantage in performing the prevalent matrix-vector multiplication (MVM) operations in deep learning workloads. However, existing technologies for constructing analog computing systems present critical tradeoffs: resistive memory offers superior power efficiency via non-volatility but suffers from resistive-capacitive delay that limits operating speed while silicon photonics enables high operating speed at the cost of excess power overhead due to sensitivity to thermal and process variations. In this work, we explore the photonic phase-change-memory (PCM) technology and introduce an accelerator named BETA for deep learning training workloads. Photonic PCM integrates the benefits of resistive memory and silicon photonics while mitigating their drawbacks. Furthermore, it unlocks one additional parallelism dimension in the frequency domain, which can effectively support the diverse computational patterns within deep learning training workloads. Specifically, BETA includes two key components: (1) a dual-datapath photonic PCM cell array capable of computing MVMs in both original and transposed formats, and (2) a pipeline design that leverages this dual-datapath capability to parallelize the forward and backward passes of the mini-batch training for minimum memory access and PCM cell programming overhead. Simulation results across a collection of deep learning workloads demonstrate the effectiveness of BETA in terms of performance and energy efficiency compared to other state-of-the-art digital and analog computing systems.**

***Index Terms*—silicon photonics, deep learning, training, phase-change-memory, back propagation**

I. Introduction

The rapid growth in the size and complexity of deep learning models [1], [2], [3] is driving an exponential increase in computational demands, thereby necessitating the accelerated scaling of the underlying computing systems. However, conventional digital computing systems are increasingly constrained by the slowdown of transistor scaling and stringent power budgets, making them difficult to meet the rising demands. In contrast, the analog computing systems have emerged as a compelling alternative due to the intrinsic efficiency in performing matrix-vector multiplication (MVM) operations often observed in deep learning workloads [4], [5].

Among the leading technologies for constructing the analog computing systems include resistive memory [6] and silicon photonics [7], each representing unique benefits over digital counterparts but also exhibiting critical limitations. The analog computing systems based on resistive memory [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19] achieve low power consumption yet suffer from limited operating speed due to resistive-capacitive delay. Conversely, the analog computing systems based on silicon photonics [20], [21], [22], [23], [24], [25], [26] enable high operating speed but incur substantial power overhead to counteract thermal and process variations [27], [28]. Photonic phase-change-memory (PCM) [29], [30] has emerged as a disruptive technology that combines the non-volatility of resistive memory with the high operating speed capability of silicon photonics while mitigating their drawbacks. In addition to this synergy, photonic PCM can operate across several wavelengths in parallel. While prior work has leveraged this property primarily to boost throughput [31], [32], we argue that its full potential lies in the strategic exploitation to support the complex and diverse computational patterns observed in deep learning training workloads.

Existing analog computing systems [8], [20], [31] are well-suited for deep learning inference workloads that include fixed MVM operations between input vectors and weight matrices. However, deep learning training workloads introduce increased complexity owing to the backpropagation algorithm [33], which incurs both MVM operations between input vectors and weight matrices in the forward pass, and between gradient vectors and transposed weight matrices in the backward pass. Supporting such diverse MVM operations on the current analog computing systems typically requires either frequent reordering of weight matrices [15] or the inclusion of the costly peripheral circuits inspired by transposable memory designs [9], [34], [35], [36].

In this paper, we present a photonic accelerator named BETA which is optimized for deep learning training workloads. BETA harnesses the unique capabilities of photonic PCM to deliver high performance and energy efficiency while addressing the computational diversity of deep learning training workloads. It includes two key innovations. First, it employs a dual-datapath photonic PCM cell array that can perform MVM operations in both the original and transposed formats. Second, it adopts a pipeline design which leverages this dual-datapath capability to process both forward and backward passes of mini-batch training in parallel, thereby minimizing costly memory access and photonic PCM cell programming. Simulations across a collection of deep learning workloads [1], [2], [37], [38], [39], [40] demonstrate that BETA achieves significantly better performance and energy efficiency than other state-of-the-art digital and analog computing systems [8], [20], [31], [41]. The key contributions of this work include:

Dual-Datapath Photonic PCM Cell Array: We propose a photonic PCM cell array architecture featuring two independent datapaths, in contrast to the single-datapath design adopted in most existing analog computing systems. One datapath supports MVM between a vector and a matrix like the single-datapath design, while the other supports MVM between a second vector and the same matrix but in its transposed format. This design presents a potential to eliminate the costly matrix reordering.

Customized Pipeline: We propose a pipeline design tailored to the dual-datapath capability. This design is built upon the insight that the dual-datapath capability naturally aligns with the computational patterns of backpropagation training, where the forward and backward passes respectively involve MVM operations on the original and transposed weight matrices. By parallelizing the forward and backward passes across training examples within a mini-batch, the proposed pipeline design maximizes data reuse and minimizes memory access and photonic PCM cell programming overhead.

Comprehensive Evaluation: We evaluate BETA against state-of-the-art digital and analog deep learning accelerators using a suite of diverse models spanning multiple application domains. Simulation results show that BETA achieves 82% and 84% reduction in execution time and energy consumption.

II. Background & Motivation

A. Deep Learning Training

Training a deep learning model using the backpropagation algorithm includes three parts: forward propagation of inputs, backward propagation of gradients, and updating weights. This work focuses on the first two parts while updating weights is assumed to be performed in the digital domain. Fig. 1 takes a three-layer convolutional neural network as an example to illustrate data and operations involved. Similar approaches can be adopted for more advanced models such as Transformer [1] and Long Short-Term Memory [2].

1) Input Forward Propagation: During forward propagation of inputs, the input vector $\boldsymbol{x}_l$ in the lth layer is multiplies with its corresponding weight matrix $\boldsymbol{W}_l$ to generate the output vector $\boldsymbol{y}_l$, which functions as the input vector $\boldsymbol{x}_{l+1}$ in the $(l+1)$th layer after processed by an activation function f_l. Following this approach, inputs are propagated from the first layer to the last one as shown in Equation (1) and (2). Please note that only weight matrices in the original format are needed during input forward propagation.

$$\boldsymbol{y}_l = \boldsymbol{W}_l \boldsymbol{x}_l \tag{1}$$

$$\boldsymbol{x}_{l+1} = f_l\left(\boldsymbol{y}_l\right) \tag{2}$$

2) Gradient Backward Propagation: During backward propagation of gradients, the gradient vector $\boldsymbol{\delta}_l$ in the lth layer is multiplied with its corresponding transposed weight matrix $\boldsymbol{W}_l^T$ to generate the gradient vector $\boldsymbol{\delta}_{l-1}$ in the $(l-1)$th layer, wherein the derivation of the activation function f_l' on the output vector $\boldsymbol{y}_{l-1}$ in the $(l-1)$th layer and the Hadamard product are also involved. The gradient vector of the last layer is derived by comparing the final output against ground truth $\boldsymbol{t}$. Following this approach, gradients are propagated from the last layer to the first one as shown in Equation (3). Please note that only weight matrices in the transposed format are needed during gradient backward propagation.

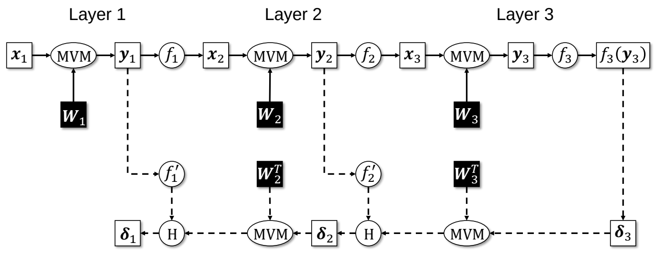

Fig. 1. Forward propagation of inputs (solid lines) and backward propagation of gradients (dashed lines) included in a three-layer convolutional neural network. This work concentrates on accelerating the matrix-vector multiplication (MVM) between either the original or transposed weight matrices, $\boldsymbol{W}_l$ or $\boldsymbol{W}_l^T$, and the corresponding input or gradient vectors, $\boldsymbol{x}_l$ or $\boldsymbol{\delta}_l$, in the respective forward propagation and backward propagation.

$$\boldsymbol{\delta}_{l-1} = \begin{cases} \left(\boldsymbol{W}_l^T \boldsymbol{\delta}_l\right) \circ f_{l-1}'\left(\boldsymbol{y}_{l-1}\right) & \textit{Hidden Layer} \\ f_{l-1}\left(\boldsymbol{y}_{l-1}\right) - \boldsymbol{t} & \textit{Output Layer} \end{cases} \tag{3}$$

3) Weight Update: During weight update, the weight gradient matrix $\boldsymbol{\Delta W}_l$ in the lth layer is obtained by multiplying the corresponding gradient vector $\boldsymbol{\delta}_l$ and the transposed input vector $\boldsymbol{x}_l$, together with a scaling factor η as shown in Equation (4), which is eventually used to update the weight matrix $\boldsymbol{W}_l$. Please note that this work concentrates on accelerating input forward propagation and gradient backward propagation. We assume that weight update is performed in the digital domain.

$$\boldsymbol{\Delta W}_l = -\eta \boldsymbol{\delta}_l \boldsymbol{x}_l^T \tag{4}$$

B. Photonic PCM Computation

1) Photonic PCM Device: Fig. 2 illustrates the architecture of a photonic PCM cell. One notable feature is its high contrast in both the electrical (resistivity) and optical (refractive index) properties between the crystalline and amorphous states. When programmed to the crystalline state, a minimum fraction of the input signal power can get through it with a loss factor of l^c. When programmed to the amorphous state, a maximum fraction of the input signal power can get through it with a loss factor of l^a. Hence, $(l^a - l^c)$ represents the most significant difference in loss factor. We incorporate a reference state with a loss factor of l^r to account for the offset induced by the non-zero l^c. When the crystalline state is used as the reference state ($l^r = l^c$), the crystalline and amorphous states correspond to 0 and 1, respectively. Similarly, when a phase state between the crystalline and amorphous states is used as the reference state ($l^r = \left(l^c + l^a\right)/2$), the crystalline and amorphous states correspond to -1 and 1, respectively. We adopt the latter setup in BETA to ensure flexible architectural support and high training accuracy and efficiency.

2) Scalar Multiplication: Two waveguides and one photonic PCM cell coupled at their intersection via directional couplers [31] as shown in Fig. 2, are used for scalar multiplication of two numbers, a and b. Number a is encoded in the input

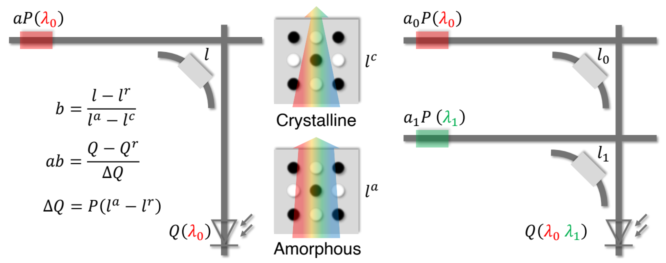
$aP(\lambda_0)$
l
$b = \frac{l - l^r}{l^a - l^c}$
$ab = \frac{Q - Q^r}{\Delta Q}$
$\Delta Q = P(l^a - l^r)$
$Q(\lambda_0)$
l^c
Crystalline
l^a
Amorphous
$a_0P(\lambda_0)$
l_0
$a_1P(\lambda_1)$
l_1
$Q(\lambda_0\ \lambda_1)$

Fig. 2. A photonic phase-change-memory (PCM) cell can be optically [31] programmed to the crystalline state with minimum loss factor $\boldsymbol{l}^c$, the amorphous state with maximum loss factor $\boldsymbol{l}^a$, or any other state between crystalline and amorphous states with a loss factor of $\boldsymbol{l} \in (\boldsymbol{l}^c, \boldsymbol{l}^a)$. A scalar multiplication ab is done by encoding a in the input signal power P and inscribing b in the photonic PCM cell phase state, then measuring the output signal power Q. A multiply-accumulate (MAC) operation is achieved by encoding multiple input signals (wavelengths) and corresponding photonic PCM cells, then measuring the accumulated output signal power. Parallel MAC operation is also supported.

signal power P of wavelength λ_0 at the left end of the horizontal waveguide via an attenuator [31]. The attenuated input signal with power aP is coupled to the photonic PCM cell whose phase state is programmed to have a loss factor of $l = l^r + (l^a - l^c)\, b$ such that the input signal can be further attenuated. The output signal with power aPl is then coupled back to the vertical waveguide and measured at the bottom end of the vertical waveguide using a photodetector. The scalar multiplication ab can be retrieved by following Equation (5), whereas $Q^r = aPl^r$ and $\Delta Q_{max} = (l^a - l^c)\, P$.

$$ab = (Q - Q^r) / \Delta Q_{max} \tag{5}$$

3) Multiply-Accumulate Computation: Deep learning workloads often involve MVM based on multiply-accumulate (MAC) operations. Considering a MAC example shown in Fig. 2, two horizontal and one vertical waveguides, and two photonic PCM cells coupled at the two intersections are used. Number a_0 and a_1 are encoded in the input signal power P of wavelengths λ_0 and λ_1 at the left ends of the two horizontal waveguides. The attenuated input signals with power a_0P and a_1P are coupled to the corresponding photonic PCM cells whose phase states are programmed to have loss factors of $l_0 = l^r + (l^a + l^c)b_0$ and $l_1 = l^r + (l^a + l^c)b_1$ such that the input signals can be further attenuated. The output signals with power a_0Pl_0 of wavelength λ_0 and a_1Pl_1 of wavelength λ_1 are then coupled back to the vertical waveguide and measured cumulatively at the bottom end of the vertical waveguide using a photodetector. The MAC result $a_0b_0 + a_1b_1$ can be similarly retrieved. In a more generic scenario, the MAC result can be retrieved by following Equation (6), whereas $Q_i^r = a_iPl^r$.

$$\sum_i a_ib_i = \left(Q - \sum_i Q_i^r\right) / \Delta Q_{max} \tag{6}$$

4) Implementation Details: The adopted photonic PCM is a relatively mature technology with prototypes available for demonstrating its application to accelerating deep learning workloads [31]. The dispersion in a photonic PCM cell can be largely neglected (same loss factor l across all wavelengths) or calibrated in the considered 180-205 THz wavelength range. We can also assume a universal input signal power P in the considered wavelength range or calibrate it at implementation time. 10^{12} cycling endurance has been reported and 10^{15} cycling endurance is expected for a photonic PCM cell [31], making them especially suitable for computation purpose. Programming the loss factor of a photonic PCM cell is done optically with picosecond pulses [31]. 8×8 photonic PCM cell array has been prototyped while 64×64 array is projected [31]. BETA assumes 64×64 photonic PCM cell array augmented with additional directional couplers to enable the proposed dual-datapath capability.

III. BETA Architecture

A. BETA System Architecture

Fig. 3 illustrates an example system architecture of BETA with nine processing tiles. These processing tiles are optically connected to an off-chip laser source and microcomb [31] using space-division multiplexing for input signal power distribution. Meanwhile, they are also electrically connected to each other and to the external High-Bandwidth Memory (HBM) [41] via an electrical torus network. Each processing tile includes a photonic PCM cell array augmented with encoding and measuring units, a share of the SRAM buffer, and other functioning blocks for activation f, derivative of activation f', Hadamard product H, etc.

B. BETA Processing Tile Architecture

1) Photonic PCM Cell Array: At the core of each BETA processing tile lies an $n \times n$ photonic PCM cell array, wherein photonic PCM cells are located at the intersections of n horizontal and n vertical waveguides.

According to Fig. 3(b), elements of a vector can be encoded on the input signal power of wavelengths λ_0 and λ_1 using the corresponding attenuators a_0 and a_1 at the top ends of two vertical waveguides. The encoded vector is multiplied by all the rows of the matrix. Specifically, 50% of the attenuated input signal power a_0P and a_1P are coupled from corresponding vertical waveguides to photonic PCM cells c_{00} and c_{01}, whose loss factors l_{00} and l_{01} represent elements in the first row of the matrix, while the rest 50% are coupled from corresponding vertical waveguides to photonic PCM cells c_{10} and c_{11}, whose loss factors l_{10} and l_{11} represent elements in the second row of the matrix. The output signal power $a_0Pl_{00}/2$ of wavelength λ_0 and $a_1Pl_{01}/2$ of wavelength λ_1 are coupled back to the top horizontal waveguide. We measure the cumulative output signal power at the right end of the top horizontal waveguide and then retrieve the first element of the resulting vector. The output signal power $a_0Pl_{10}/2$ of wavelength λ_0 and $a_1Pl_{11}/2$ of wavelength λ_1 are coupled back to the bottom horizontal waveguide. We measure the cumulative output signal power at the right end of the bottom horizontal waveguide and then retrieve the second element of the resulting vector.

Similarly, elements of another vector can be encoded on the input signal power of wavelengths λ_2 and λ_3 using the

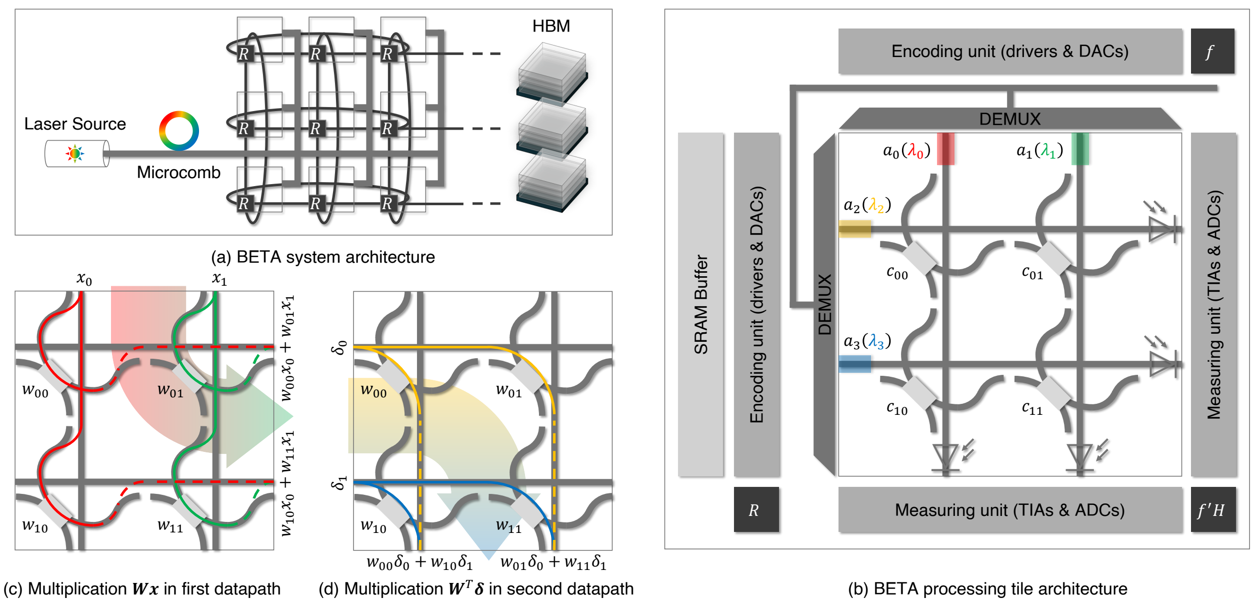
HBM
Laser Source
Microcomb
R
(a) BETA system architecture
Encoding unit (drivers & DACs)
f
DEMUX
SRAM Buffer
Encoding unit (drivers & DACs)
DEMUX
$a_0(\lambda_0)$
$a_1(\lambda_1)$
$a_2(\lambda_2)$
$a_3(\lambda_3)$
c_{00}
c_{01}
c_{10}
c_{11}
Measuring unit (TIAs & ADCs)
Measuring unit (TIAs & ADCs)
R
$f'H$
(b) BETA processing tile architecture
x_0
x_1
w_{00}
w_{01}
w_{10}
w_{11}
$w_{00}x_0 + w_{01}x_1$
$w_{10}x_0 + w_{11}x_1$
(c) Multiplication $\boldsymbol{Wx}$ in first datapath
δ_0
δ_1
$w_{00}\delta_0 + w_{10}\delta_1$
$w_{01}\delta_0 + w_{11}\delta_1$
(d) Multiplication $\boldsymbol{W}^T\boldsymbol{\delta}$ in second datapath

Fig. 3. BETA system architecture (a) with nine processing tiles electrically connected via a torus network and optically connected to the laser source and microcomb using space-division multiplexing. High-Bandwidth Memory (HBM) is also electrically connected to the processing tiles via the corresponding routers [42]. Each processing tile (b) includes an $n \times n$ ($n = 2$ in this example) photonic phase-change-memory (PCM) cell array, augmented with encoding and measuring units, SRAM buffer, and other functioning blocks for activation, derivative of activation, Hadamard product, etc. The parallel multiplications between the weight matrix $\boldsymbol{W}$ and input vector $\boldsymbol{x}$ (c), and between the transposed weight matrix $\boldsymbol{W}^T$ and gradient vector $\boldsymbol{\delta}$ (d) using non-overlapping sets of wavelengths, have also been demonstrated.

corresponding attenuators a_2 and a_3 at the left ends of two horizontal waveguides. The encoded vector is multiplied by all the columns of the matrix (all the rows of the transposed matrix). The two elements of the resulting vector are retrieved from the measured cumulative output signal power at the bottom ends of the two vertical waveguides. Please note that a BETA processing tile adopts two sets of non-overlapping wavelengths (λ_0/λ_1 and λ_2/λ_3 in Fig. 2(b)). Each photonic PCM cell processes signals of two wavelengths in parallel, representing the additional dimension of parallelism in the frequency domain.

2) Encoding and Measuring Units: There are two encoding units located close to the top and left edges of the photonic PCM cell array. Similarly, there are two measuring units located close to the bottom and right edges of the photonic PCM cell array. Each encoding unit includes n pairs of digital-to-analog converter (DAC) and driver whereas each pair is connected to a specific attenuator. Each measuring unit includes n pairs of transimpedance amplifier (TIA) and analog-to-digital converter (ADC) whereas each pair is connected to a specific photodetector. The number of pairs of DAC and driver, or TIA and ADC, is $2n$, which is the same as the number of wavelengths involved. Such a design aligns well with BETA as the peripheral circuit cost is proportional to the number of wavelengths (system throughput). However, it does not align well with analog computing systems based on resistive memory [9], as only one encoding and one measuring units can be utilized at any given time to avoid a short circuit.

3) Miscellaneous Modules: The on-chip SRAM buffer is evenly distributed to each BETA processing tile. The local share of the SRAM buffer on a processing tile interacts with the local photonic PCM cell array through the encoding and measuring units while interacting with other processing tiles and the external HBM through the torus network via the dedicated router (R). The local share of the SRAM buffer stores input vectors $\boldsymbol{x}$, output vectors $\boldsymbol{y}$, gradient vectors $\boldsymbol{\delta}$, and weight gradient matrices $\boldsymbol{\Delta W}$ for future reuse. As the SRAM buffer takes a significant fraction of the die size [41], we need to strategically schedule computations to maximize data reuse and minimize the costly memory access and photonic PCM cell programming overhead. There are several more functioning blocks on each processing tile. The activation block f performs activation on an output vector to generate an input vector of the subsequent layer during the forward propagation of inputs. The derivative of activation and Hadamard product block $f'\&H$ performs Hadamard product between the derivative of activation of the output vector of the previous layer and MVM between the transposed weight matrix and the gradient vector to generate the gradient vector of the previous layer during the backward propagation of gradients. We assume that both functioning blocks work in the digital domain. Other necessary functioning blocks such as pooling are not to be discussed here.

4) Dual-Datapath Capability: The dual-datapath capability naturally aligns with the computational patterns of backpropagation training, where the forward and backward passes respectively involve MVM on original and transposed weight matrices. Fig. 3(c) and Fig. 3(d) show the MVM between an input vector and a weight matrix and MVM between a gradient vector and the same weight matrix but in its transposed format, respectively, while the weight matrix $(w_{00}, w_{01}; w_{10}, w_{11})$ is fixated in the processing PCM cell array throughout the process.

During the forward pass, input vector $\boldsymbol{x} = (x_0, x_1)$ is encoded on wavelengths λ_0 and λ_1 and transmitted via two vertical waveguides. The input vector is multiplied by the first row of the weight matrix (w_{00}, w_{01}). The result $x_0 w_{00} + x_1 w_{01}$ is retrieved from the measurement of the output signal power at the right end of the top horizontal waveguide. Similarly, the input vector is also multiplied by the second row of the weight matrix (w_{10}, w_{11}). The result $x_0 w_{10} + x_1 w_{11}$ is retrieved from the measurement of the output signal power at the right end of the bottom horizontal waveguide. The resulting output vector $\boldsymbol{y} = (x_0 w_{00} + x_1 w_{01}, x_0 w_{10} + x_1 w_{11})$ is therefore obtained. This path of feeding a vector at the top ends of vertical waveguides and retrieving the resulting vector at the right ends of horizontal waveguide represents the first datapath.

During the backward pass, gradient vector $\boldsymbol{x} = (\delta_0, \delta_1)$ is encoded on wavelengths λ_2 and λ_3 and transmitted via two horizontal waveguides. The gradient vector is multiplied by the first column of the weight matrix (w_{00}, w_{10}) (first row of the transposed weight matrix). The result $\delta_0 w_{00} + \delta_1 w_{10}$ is retrieved from the measurement of the output signal power at the bottom end of the left vertical waveguide. Similarly, the gradient vector is also multiplied by the second column of the weight matrix (w_{10}, w_{11}) (second row of the transposed weight matrix). The result $\delta_0 w_{01} + \delta_1 w_{11}$ is retrieved from the measurement of the output signal power at the bottom end of the right vertical waveguide. The resulting vector $(\delta_0 w_{00} + \delta_1 w_{10}, \delta_0 w_{01} + \delta_1 w_{11})$ is therefore obtained. This path of feeding a vector at the left ends of horizontal waveguides and retrieving the resulting vector at the bottom ends of vertical waveguides represents the second datapath.

C. BETA Pipeline

We take the neural network example from Fig. 1 to illustrate how the mapping of neural network layers to BETA processing tiles is done and show the result in Fig. 4. Please note that we assume each processing tile can fully accommodate the computation and data of a neural network layer for simplicity. When mapping real deep learning models to BETA architecture, the combination of multiple BETA processing tiles can be regarded as a virtual tile to accommodate a neural network layer, whereas inter-tile communication overhead through the torus network is considered. When the combination of all available BETA processing tiles still cannot accommodate a neural network layer, the tiling technique as in [43], [44] is utilized with the target of minimizing off-chip memory access.

As shown in Fig. 4, three layers of the example neural network are mapped to three processing tiles (or virtual processing tiles each including several physical tiles). Each tile fixes the weight matrix on its photonic PCM cell array while holding involved input vectors, output vectors, gradient vectors, and weight gradient matrix in the local share of the SRAM buffer.

During the first itinerary, Tile 1 is activated and performs MVM between input vector $\boldsymbol{x}_1$ and weight matrix $\boldsymbol{W}_1$ for

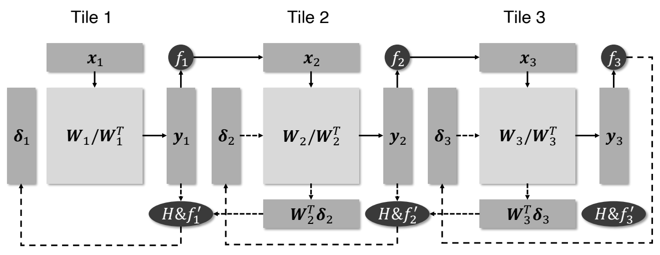

Fig. 4. Mapping the three-layer neural network shown in Fig. 1 to three BETA processing tiles, assuming each processing tile can fully accommodate an individual neural network layer. We use solid and dashed lines to explain data exchanges between the processing tiles during forward propagation and backward propagation, respectively.

output vector $\boldsymbol{y}_1$, which is processed by the activation function f_1 and forwarded to Tile 2. During the second itinerary, Tile 2 is activated and performs MVM between input vector $\boldsymbol{x}_2$ and weight matrix $\boldsymbol{W}_2$ for output vector $\boldsymbol{y}_2$, which is processed by the activation function f_2 and forwarded to Tile 3. During the third itinerary, Tile 3 is activated and performs MVM between input vector $\boldsymbol{x}_3$ and weight matrix $\boldsymbol{W}_3$ for output vector $\boldsymbol{y}_3$, which is processed by the activation function f_3 to get the final output vector of the neural network. This marks the end of the forward pass. During the fourth itinerary, the final output vector is compared against the ground truth $\boldsymbol{t}$ to generate the gradient vector $\boldsymbol{\delta}_3$. Tile 3 is activated and performs MVM between the gradient vector $\boldsymbol{\delta}_3$ and the transposed weight matrix $\boldsymbol{W}_3^T$. The resulting gradient vector is forwarded to Tile 2. During the fifth itinerary, the gradient vector $\boldsymbol{\delta}_2$ is generated through the $f_2' \& H$ function. Tile 2 is activated and performs MVM between the gradient vector $\boldsymbol{\delta}_2$ and the transposed weight matrix $\boldsymbol{W}_2^T$. The resulting gradient vector is forwarded to Tile 1. During the last itinerary, the gradient vector $\boldsymbol{\delta}_1$ is generated through the $f_1' \& H$ function. This marks the end of the backward pass.

Please note that although the dual-datapath capability of BETA architecture eliminates the costly reordering of the weight matrix between the forward and backward passes, we are still far away from optimal execution as only one processing tile is activated at any given time, leading to significant system underutilization. This is due to the strict sequential execution order within a training example. We propose to pipeline the forward and backward passes of training examples within a mini-batch. In this way, all BETA tiles can be fully utilized while the strict sequential execution order per training example is preserved. Such a pipeline leads to reduction in execution time as demonstrated in [45] and smaller on-chip footprint.

IV. Evaluation Methodology

A. Evaluation Setup

We develop a cycle-accurate simulator to model BETA and other baseline architectures. The key parameters of the involved components are listed below. We assume 8-bit DACs with 14 GS/s sampling speed, 50 mW power consumption, and a footprint of 11000 μm^2 [46], 8-bit ADCs with 10 GS/s sampling speed, 15 mW power consumption, and a footprint

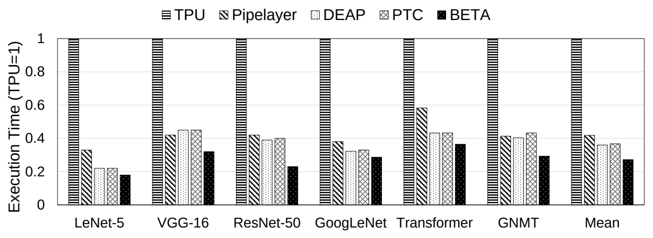

Fig. 5. Execution time comparison between BETA and baseline architectures when applying stochastic gradient descent training.

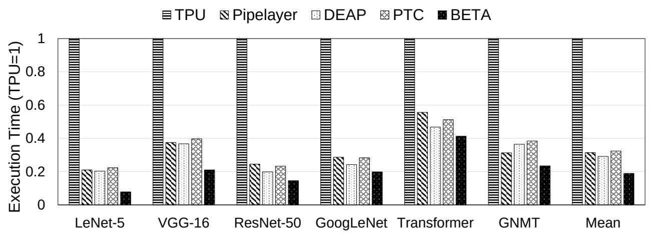

Fig. 7. Execution time comparison between BETA and baseline architectures when applying mini-batch ($b = 128$) gradient descent training.

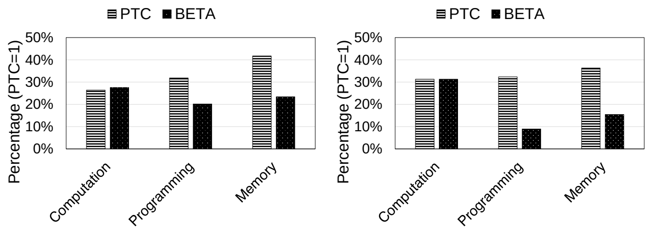

Fig. 6. Execution time breakdown of baseline architecture PTC and BETA when applying stochastic (left) and mini-batch (right) gradient descent training.

of 2850 μm^2 [47]. The sampling speeds of the adopted DACs and ADCs are above or equal to 10 GS/s as we assume 10 GHz operating speed for BETA and other optical baseline architectures. We assume TIA with 3 mW power consumption and a footprint 11000 of μm^2 [48], and photodetectors with 1.1 mW power consumption, a footprint of 40 μm^2 [49], and sensitivity of -23 dBm [50]. The insertion loss values of direction couplers and waveguide crossing are 0.1 dB [25] and 0.03 dB [51], respectively. The size of the photonic PCM array of each BETA processing tile is 64×64, as projected in [31]. For a fair comparison, we assume a 600 mm^2 die size, 128 MB SRAM buffer, 32 GB external HBM, and 1200 GB/s external memory bandwidth across all baselines and BETA architecture. The computation formats for digital and analog baselines are 16-bit floating-point and 16-bit fixed-point [52], respectively. The technologies for constructing analog computing systems often do not support 16-bit resolution per device cell. In such a case, multiple cells are cascaded as in [8] to support fixed-point 16-bit computation. The bandwidth per torus link is set at 50 GB/s.

B. Baseline Architectures

The proposed BETA architecture is compared against four state-of-the-art digital and analog computing systems for DNN training workloads, namely TPU [41], Pipelayer [8], DEAP [20], and PTC [31]. TPU baseline, which is the only digital computing system here, exhibits 16-bit floating-point format for computation with an operating frequency of 1.05 GHz. Pipelayer exhibits a 16-bit fixed-point format for computation with an operating frequency of 1 GHz. Since each resistive memory cell is assumed to have 4-bit resolution, four such resistive memory cells are cascaded to support 16-bit fixed-point computation. DEAP exhibits a 16-bit fixed-point format for computation with an operating frequency of 10 GHz. Three microring resonators (MRRs) are cascaded to support 16-bit fixed-point computation, as each MRR has a 6-bit resolution. PTC also exhibits a 16-bit fixed-point format for computation with an operating frequency of 10 GHz. Two photonic PCM cells are cascaded to support 16-bit fixed-point computation, as each photonic PCM cell has an 8-bit resolution [53], which is the same as in BETA architecture.

C. Benchmark

We evaluate BETA and other baseline architectures using six DNN models: LeNet-5 [37] with MNIST dataset, VGG-16 [38], ResNet-50 [39], and GoogLeNet [40] with ImageNet dataset, Transformer [1] with WMT17 EN-DE dataset, and GNMT [2] with WMT16 EN-DE dataset. The selection aims to cover a wide spectrum of DNN models with different characteristics and application scenarios for a comprehensive evaluation. For each selected DNN model, BETA and other baseline architectures are tested using stochastic and mini-batch gradient descent approaches. BETA is expected to deliver the most significant improvements in terms of performance and energy consumption when the mini-batch gradient descent approach is adopted.

V. Experiment Results

A. Execution Time

1) Stochastic Gradient Descent: Fig. 5 shows the execution time comparison between BETA and other baseline architectures when adopting the stochastic gradient descent approach. All values are normalized to that of the TPU baseline architecture. Compared with the TPU baseline architecture, all its analog counterparts achieve significant reductions in execution time. Specifically, Pipelayer, DEAP, PTC, and BETA achieve on average 58%, 64%, 63%, and 73% reductions in execution time, proving the advantages of analog computing systems in performing MVM operations. DEAP and PTC baseline architectures achieve similar execution time values as they follow the same operating mechanism, only on different optical devices (MRRs and photonic PCM cells for DEAP and PTC, respectively). The execution time for PTC baseline architectures is on average 2% higher than the execution time for DEAP baseline architecture, mainly due to the higher delay of programming the photonic PCM cells. BETA achieves 25% execution time reduction as compared to the PTC baseline architecture, demonstrating the effectiveness of the BETA

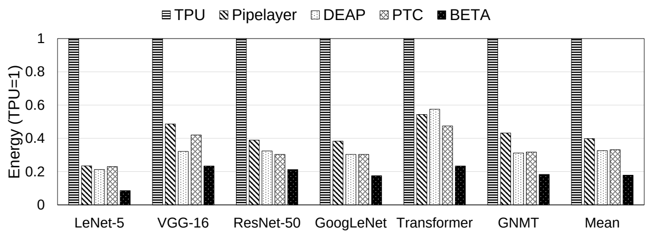

Fig. 8. Energy comparison between BETA and baseline architectures when applying stochastic gradient descent training.

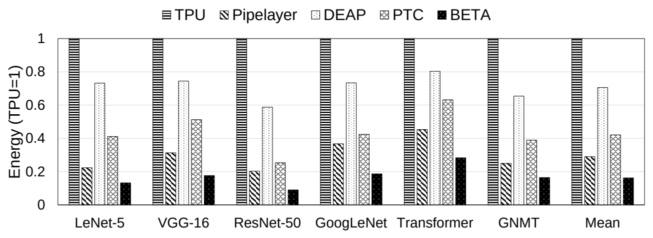

Fig. 9. Energy comparison between BETA and baseline architectures when applying mini-batch ($b = 128$) gradient descent training.

pipeline, which minimizes the costly memory access and photonic PCM cell programming operations. Fig. 6 (left) shows the execution time breakdown analysis of BETA and the PTC baseline architecture when training the VGG-16 model using stochastic gradient descent. The overall execution time is divided into three parts: time for actual computation, time for memory access, and time for programming photonic PCM cells. We conclude that BETA and the PTC baseline architecture take similar time for actual computation, while BETA takes less time for memory access and photonic PCM cell programming.

2) Mini-Batch Gradient Descent: Fig. 7 shows the execution time comparison between BETA and other baseline architectures when applying mini-batch gradient descent training ($b = 128$). All values are normalized to that of the TPU baseline architecture. Compared with the TPU baseline architecture, Pipelayer, DEAP, PTC, and BETA achieve on average 60%, 67%, 67%, and 82% reductions in execution time. BETA achieves 48% execution time reduction as compared to the PTC baseline architecture, which is higher than 25% execution time reduction when applying stochastic gradient descent training, because BETA pipeline exhibits higher benefits by processing the forward propagation and backward propagation of different training examples in the same mini-batch in parallel while maintaining the sequential processing order of layers in each training example. Fig. 6 (right) shows the execution time breakdown analysis of BETA and the PTC baseline architecture when training the VGG-16 model using mini-batch gradient descent ($b = 128$). The reduction in time for memory access and photonic PCM cell programming is even more significant in the mini-batch gradient descent case, because of the BETA pipeline design.

B. Energy Consumption

Fig. 8 and Fig. 9 show the energy consumption comparison between BETA and other baseline architectures when applying stochastic and mini-batch gradient descent training approaches, respectively. All values are normalized to that of the TPU baseline architecture. Compared with the TPU baseline architecture, all its analog counterparts achieve reductions in energy consumption. Specifically, Pipelayer, DEAP, PTC, and BETA achieve on average 70%, 46%, 69%, and 80% reductions in energy consumption, respectively, when applying stochastic gradient descent training. Pipelayer, DEAP, PTC, and BETA achieve on average 71%, 29%, 58%, and 84% reductions in energy consumption, respectively, when applying mini-batch gradient descent training. When comparing optical baseline architectures and BETA, we observe that the DEAP baseline architecture consumes the highest energy because it requires excessive tuning power to compensate for temperature and process variations. The PTC baseline architecture also exhibits relatively high energy consumption due to the power consumed for photonic PCM cell programming. BETA achieves the lowest energy consumption because it significantly minimizes the costly memory access and photonic PCM cell programming operations and does not require additional power due to the non-volatility property of photonic PCM.

VI. Conclusion

This paper introduces BETA, a photonic accelerator optimized for deep learning training workloads. BETA harnesses the disruptive photonic PCM technology and exploits the additional dimension of parallelism in the frequency domain. BETA features (1) a dual-datapath photonic PCM cell array that computes MVM in both original and transposed matrix formats, and (2) a pipeline design tailored to the dual-datapath capability to parallelize the forward and backward passes of training examples within a mini-batch, minimizing memory access and photonic PCM cell programming overhead.

References

[1] A. Vaswani *et al.*, "Attention is All You Need," in *NeurIPS*, 2017, pp. 1–11, https://doi.org/10.48550/arXiv.1706.03762.

[2] Y. Wu *et al.*, "Google's Neural Machine Translation System: Bridging the Gap Between Human and Machine Translation," in *ArXiv Preprint*, 2016, pp. 1–23, https://doi.org/10.48550/arXiv.1609.08144.

[3] P. Mattson *et al.*, "MLPerf Training Benchmark," in *MLSys*, 2020, pp. 336–349, https://doi.org/10.48550/arXiv.1910.01500.

[4] J. Yang, D. Strukov, and D. Stewart, "Memristive Devices for Computing," *Nature Nanotechnology*, vol. 8, no. 1, pp. 13–24, 2012, https://doi.org/10.1038/nnano.2012.240.

[5] Q. Cheng *et al.*, "Silicon Photonics Codesign for Deep Learning," *Proceedings of the IEEE*, vol. 108, no. 8, pp. 1261–1282, 2020, https://doi.org/10.1109/JPROC.2020.2968184.

[6] P. Wong *et al.*, "Metal-Oxide RRAM," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012, https://doi.org/10.1109/JPROC.2012.2190369.

[7] D. Miller, "Device Requirements for Optical Interconnects to Silicon Chips," *Proceedings of the IEEE*, vol. 97, no. 7, pp. 1166–1185, 2009, https://doi.org/10.1109/JPROC.2009.2014298.

[8] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," in *HPCA*, 2017, pp. 541–552, https://doi.org/10.1109/HPCA.2017.55.

[9] A. Ranjan *et al.*, "X-MANN: A Crossbar Based Architecture for Memory Augmented Neural Networks," in *DAC*, 2019, pp. 1–6, https://doi.org/10.1145/3316781.3317935.

[10] A. Shafiee *et al.*, “ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars,” in *ISCA*, 2016, pp. 14–26, https://doi.org/10.1109/ISCA.2016.12.

[11] P. Chi *et al.*, “PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory,” in *ISCA*, 2016, pp. 27–39, https://doi.org/10.1109/ISCA.2016.13.

[12] F. Alibart, E. Zamanidoost, and D. Strukov, “Pattern Classification by Memristive Crossbar Circuits Using Ex-Situ and In-Situ Training,” *Nature Communications*, vol. 4, no. 1, p. 2072, 2013, https://doi.org/10.1038/ncomms3072.

[13] M. Prezioso *et al.*, “Training and Operation of An Integrated Neuromorphic Network Based on Metal-Oxide Memristors,” *Nature*, vol. 521, no. 7550, pp. 61–64, 2015, https://doi.org/10.1038/nature14441.

[14] M. Hu *et al.*, “Dot-Product Engine for Neuromorphic Computing: Programming 1T1M Crossbar to Accelerate Matrix-Vector Multiplication,” in *DAC*, 2016, pp. 1–6, https://doi.org/10.1145/2897937.2898010.

[15] B. Kim, S. Li, and H. Li, “INCA: Input-Stationary Dataflow at Outside-the-Box Thinking About Deep Learning Accelerators,” in *HPCA*, 2023, pp. 29–41, https://doi.org/10.1109/HPCA56546.2023.10070992.

[16] A. Ankit *et al.*, “PUMA: A Programmable Ultra-Efficient Memristor-Based Accelerator for Machine Learning Inference,” in *ASPLOS*, 2019, pp. 715–731, https://doi.org/10.1145/3297858.3304049.

[17] M. Imani, S. Gupta, Y. Kim, and T. Rosing, “FloatPIM: In-Memory Acceleration of Deep Neural Network Training with High Precision,” in *ISCA*, 2019, pp. 802–815, https://doi.org/10.1145/3307650.3322237.

[18] Y. Ji *et al.*, “FPSA: A Full System Stack Solution for Reconfigurable ReRAM-Based NN Accelerator Architecture,” in *ASPLOS*, 2019, pp. 733–747, https://doi.org/10.1145/3297858.3304048.

[19] G. Yuan *et al.*, “FORMS: Fine-Grained Polarized ReRAM-Based In-Situ Computation for Mixed-Signal DNN Accelerator,” in *ISCA*, 2021, pp. 265–278, https://doi.org/10.1109/ISCA52012.2021.00029.

[20] V. Bangari *et al.*, “Digital Electronics and Analog Photonics for Convolutional Neural Networks (DEAP-CNNs),” *JSTQE*, vol. 26, no. 1, pp. 1–13, 2020, https://doi.org/10.1109/JSTQE.2019.2945540.

[21] F. Ashtiani, A. Geers, and F. Aflatouni, “An On-Chip Photonic Deep Neural Network for Image Classification,” *Nature*, vol. 606, no. 7914, pp. 501–506, 2022, https://doi.org/10.1038/s41586-022-04714-0.

[22] X. Xu *et al.*, “11 TOPS Photonic Convolutional Accelerator for Optical Neural Networks,” *Nature*, vol. 589, no. 7840, pp. 44–51, 2021, https://doi.org/10.1038/s41586-020-03063-0.

[23] A. Tait, M. Nahmias, B. Shastri, and P. Prucnal, “Broadcast and Weight: An Integrated Network for Scalable Photonic Spike Processing,” *JLT*, vol. 32, no. 21, pp. 4029–4041, 2014, https://doi.org/10.1109/JLT.2014.2345652.

[24] K. Shiflett, A. Karanth, R. Bunescu, and A. Louri, “Albireo: Energy-Efficient Acceleration of Convolutional Neural Networks via Silicon Photonics,” in *ISCA*, 2021, pp. 860–873, https://doi.org/10.1109/ISCA52012.2021.00072.

[25] H. Zhu *et al.*, “Lightening-Transformer: A Dynamically-Operated Optically-Interconnected Photonic Transformer Accelerator,” in *HPCA*, 2024, pp. 686–703, https://doi.org/10.1109/HPCA57654.2024.00059.

[26] G. Wetzstein *et al.*, “Inference in Artificial Intelligence with Deep Optics and Photonics,” *Nature*, no. 7836, pp. 39–47, 2020, https://doi.org/10.1038/s41586-020-2973-6.

[27] Y. Li, A. Louri, and A. Karanth, “SPRINT: A High-Performance, Energy-Efficient, and Scalable Chiplet-Based Accelerator with Photonic Interconnects for CNN Inference,” *TPDS*, vol. 33, no. 10, pp. 2332–2345, 2021, https://doi.org/10.1109/TPDS.2021.3139015.

[28] Y. Li, K. Wang, H. Zheng, A. Louri, and A. Karanth, “Ascend: A Scalable and Energy-Efficient Deep Neural Network Accelerator with Photonic Interconnects,” *TCAS-I*, vol. 69, no. 7, pp. 2730–2741, 2022, https://doi.org/10.1109/TCSI.2022.3169953.

[29] C. Rios *et al.*, “Integrated All-Photonic Non-Volatile Multi-Level Memory,” *Nature Photonics*, vol. 9, no. 11, pp. 725–732, 2015, https://doi.org/10.1038/nphoton.2015.182.

[30] J. Feldmann *et al.*, “Calculating with Light Using a Chip-Scale All-Optical Abacus,” *Nature Communications*, vol. 8, no. 1, p. 1256, 2017, https://doi.org/10.1038/s41467-017-01506-3.

[31] J. Feldmann *et al.*, “Parallel Convolutional Processing Using an Integrated Photonic Tensor Core,” *Nature*, vol. 589, no. 7840, pp. 52–58, 2021, https://doi.org/10.1038/s41586-020-03070-1.

[32] Y. Li, A. Louri, and A. Karanth, “MERIT: A Sustainable DNN Accelerator Design with Photonic Phase-Change Memory,” *TSUSC*, vol. 10, no. 4, pp. 705–716, 2024, https://doi.org/10.1109/TSUSC.2024.3521847.

[33] D. Rumelhard, G. Hinton, and R. Williams, *Learning Internal Representations by Error Propagation*. MIT Press, 1986, pp. 318–362, http://ieeexplore.ieee.org/document/6302929.

[34] B. A. Chappell, Y.-C. Lien, and J. Y. Tang, “Transposable Memory Architecture,” 1989, U.S. Patent No. 4,845,669. https://patents.google.com/patent/US4845669A/en.

[35] W. Wan *et al.*, “A 74 TMACS/W CMOS-RRAM Neurosynaptic Core with Dynamically Reconfigurable Dataflow and In-Situ Transposable Weights for Probabilistic Graphical Models,” in *ISSCC*, 2020, pp. 498–500, https://doi.org/10.1109/ISSCC19947.2020.9062979.

[36] S. Abel *et al.*, “Silicon Photonics Integration Technologies for Future Computing Systems,” in *OECC/PSC*, 2019, pp. 1–3, https://doi.org/10.23919/PS.2019.8818051.

[37] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998, https://doi.org/10.1109/5.726791.

[38] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” in *ICLR*, 2015, pp. 1–14, https://doi.org/10.48550/arXiv.1409.1556.

[39] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *CVPR*, 2016, pp. 770–778, https://doi.org/10.1109/CVPR.2016.90.

[40] C. Szegedy *et al.*, “Going Deeper with Convolutions,” in *CVPR*, 2015, pp. 1–9, https://doi.org/10.1109/CVPR.2015.7298594.

[41] N. Jouppi *et al.*, “TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings,” in *ISCA*, 2023, pp. 1–14, https://doi.org/10.1145/3579371.3589350.

[42] Y. Li and A. Louri, “ALPHA: A Learning-Enabled High-Performance Network-on-Chip Router Design for Heterogeneous Manycore Architectures,” *TSUSC*, vol. 6, no. 2, pp. 274–288, 2020, https://doi.org/10.1109/TSUSC.2020.2981340.

[43] Y. Li, A. Louri, and A. Karanth, “SPACX: Silicon Photonics-based Scalable Chiplet Accelerator for DNN Inference,” in *HPCA*, 2022, pp. 831–845, https://doi.org/10.1109/HPCA53966.2022.00066.

[44] Y. Li, A. Louri, and A. Karanth, “A High-Performance and Energy-Efficient Photonic Architecture for Multi-DNN Acceleration,” *TPDS*, vol. 35, no. 1, pp. 46–58, 2023, https://doi.org/10.1109/TPDS.2023.3327535.

[45] D. Narayanan *et al.*, “PipeDream: Generalized Pipeline Parallelism for DNN Training,” in *SOSP*, 2019, pp. 1–15, https://doi.org/10.1145/3341301.3359646.

[46] P. Caragiulo, O. E. Mattia, A. Arbabian, and B. Murmann, “A Compact 14 GS/s 8-Bit Switched-Capacitor DAC in 16 nm FinFET CMOS,” in *VLSI Symposium*, 2020, pp. 1–2, https://doi.org/10.1109/VLSICircuits18222.2020.9162776.

[47] J. Liu, M. Hassanpourghadi, and M. S.-W. Chen, “A 10 GS/s 8b 25fJ/c-s 2850 μm^2 Two-Step Time-Domain ADC Using Delay-Tracking Pipelined-SAR TDC with 500 fs Time Step in 14 nm CMOS Technology,” in *ISSCC*, 2022, pp. 160–162, https://doi.org/10.1109/ISSCC42614.2022.9731625.

[48] M. Rakowski *et al.*, “Hybrid 14 nm FinFET - Silicon Photonics Technology for Low-Power Tb/s/mm^2 Optical I/O,” in *VLSI Symposium*, 2018, pp. 221–222, https://doi.org/10.1109/VLSIT.2018.8510668.

[49] Z. Huang *et al.*, “25 Gbps Low-Voltage Waveguide Si-Ge Avalanche Photodiode,” *Optica*, vol. 3, no. 8, pp. 793–798, 2016, https://doi.org/10.1364/OPTICA.3.000793.

[50] Y. Li, A. Louri, and A. Karanth, “A Silicon Photonic Multi-DNN Accelerator,” in *PACT*, 2023, pp. 238–249, https://doi.org/10.1109/PACT58117.2023.00028.

[51] Y. Li, A. Louri, and A. Karanth, “Scaling Deep-Learning Inference with Chiplet-Based Architecture and Photonic Interconnects,” in *DAC*, 2021, pp. 931–936, https://doi.org/10.1109/DAC18074.2021.9586311.

[52] X. Zhang *et al.*, “Fixed-Point Back-Propagation Training,” in *CVPR*, 2020, pp. 2330–2338, https://doi.org/10.1109/CVPR42600.2020.00240.

[53] I. Giannopoulos *et al.*, “8-Bit Precision In-Memory Multiplication with Projected Phase-Change Memory,” in *IEDM*, 2018, pp. 1–4, https://doi.org/10.1109/IEDM.2018.8614558.