Photonic Reconfigurable Accelerators for Efficient Inference of CNNs with Mixed-Sized Tensors

Sairam Sri Vatsavai, *Student Member, IEEE,* Ishan G Thakkar, *Member, IEEE*

***Abstract*—Photonic Microring Resonator (MRR) based hardware accelerators have been shown to provide disruptive speedup and energy-efficiency improvements for processing deep Convolutional Neural Networks (CNNs). However, previous MRR-based CNN accelerators fail to provide efficient adaptability for CNNs with mixed-sized tensors. One example of such CNNs is depthwise separable CNNs. Performing inferences of CNNs with mixed-sized tensors on such inflexible accelerators often leads to low hardware utilization, which diminishes the achievable performance and energy efficiency from the accelerators. In this paper, we present a novel way of introducing reconfigurability in the MRR-based CNN accelerators, to enable dynamic maximization of the size compatibility between the accelerator hardware components and the CNN tensors that are processed using the hardware components. We classify the state-of-the-art MRR-based CNN accelerators from prior works into two categories, based on the layout and relative placements of the utilized hardware components in the accelerators. We then use our method to introduce reconfigurability in accelerators from these two classes, to consequently improve their parallelism, flexibility of efficiently mapping tensors of different sizes, speed and overall energy efficiency. We evaluate our reconfigurable accelerators against three prior works for the area proportionate outlook (equal hardware area for all accelerators). Our evaluation for the inference of four modern CNNs indicates that our designed reconfigurable CNN accelerators provide improvements of up to 1.8× in Frames-Per-Second (FPS) and up to 1.5× in FPS/W, compared to an MRR-based accelerator from prior work.**

***Index Terms*—Deep Learning, Accelerator, Reconfigurability, Silicon Photonics**

I. Introduction

Convolutional Neural Networks (CNNs) have shown record-breaking performance for implementing various real-world artificial intelligence tasks such as image recognition, language translation, and autonomous driving [1]–[4]. The ever-increasing complexity of CNNs has pushed for highly customized CNN hardware accelerators [5]. Among them, silicon-photonic CNN accelerators have shown great promise to provide unparalleled throughput, ultra-low latency, and high energy efficiency [6]–[11]. Typically, a silicon-photonic CNN accelerator consists of multiple Tensor Product Cores (TPCs) that perform multiple tensor products in parallel. Several TPC-based photonic CNN accelerators have been proposed in prior works based on various silicon-photonic devices, such as Mach Zehnder Interferometer (MZI) (e.g., [12], [13], [14]) and Microring Resonator (MRR) (e.g., [10], [11], [15], [16]).

Among these TPC-based photonic CNN accelerators from prior work, the MRR-enabled TPC-based accelerators (e.g., [10], [11], [15]–[17]) have shown disruptive performance and energy efficiencies for processing CNN tensor products, due to the MRRs' compact footprint, low dynamic power consumption, and compatibility with cascaded Dense-Wavelength-Division-Multiplexing (DWDM). The MRR-enabled TPCs of these accelerators transform CNN tensor products into Vector Dot Products (VDPs) by decomposing the input tensors into vectors (1D tensors). The VDP operations are performed on the individual VDP Elements (VDPEs), which are the main MRR-enabled hardware components in a TPC. Multiple VDPEs in a TPC can perform multiple VDP operations in parallel. The results of these VDP operations can be summed together (when needed) using a partial-sum (*psum*) reduction network, which can be employed outside of the TPCs as part of the post-processing components of the CNN accelerator. The functioning of the TPCs and their constituent VDPEs in the ultra-high-speed photonic domain results in disruptive throughput for processing tensors.

However, the existing MRR-enabled TPC-based accelerators are not efficient in processing modern CNNs with mixed-sized tensors, such as Xception [18] and MobileNetV1 [19]. This is because these modern CNNs utilize depthwise separable convolutions in addition to the standard convolutions. Depthwise separable convolutions in a CNN employ reduced sized tensors compared to the standard convolutions, to reduce the overall computational load of processing the CNN. For instance, MobileNetV1 shows 8-9× reduction in its computational load with only 1% accuracy drop [19]. But the existing MRR-enabled TPCs and their constituent VDPEs have fixed sizes. Therefore, mapping the processing of the modern CNNs with mixed-sized tensors [20] on such fixed-sized TPCs often leads to a low hardware utilization in the TPCs. This in turn diminishes the achievable performance and energy efficiency from such fixed-sized TPCs-based accelerators. This is because the low hardware utilization incurs non-amortizable area and static power overheads while also idling away the opportunity for increasing the processing throughput.

To address this shortcoming, in this paper, we present a novel way of introducing reconfigurability in the MRR-enabled TPCs-based CNN accelerators, to enable efficient support for both depthwise separable convolutions and standard convolutions. To enable this reconfigurability, we invent reconfigurable VDPEs that employ MRR-based comb switches to allow re-aggregation of the CNN vectors (decomposed 1D tensors) to consequently enable dynamic resizing of the produced VDP results. Our evaluations show that our invented reconfigurable VDPEs can (1) substantially improve the hardware utilization, (2) enhance the flexibility of processing CNN tensors of various sizes, (3) improve the opportunities for

Manuscript received April 07, 2022; revised June 11, 2022; accepted July 05, 2022. This article was presented in the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES) 2022 and appears as part of the ESWEEK-TCAD special issue.

parallel tensor processing, and (4) significantly enhance the energy efficiency, for CNN inference acceleration.

Our key contributions in this paper are summarized below:

- We review several state-of-the-art MRR-enabled TPC-based CNN accelerators from prior work and then classify the circuit-level TPC organizations used in these accelerators into two categories, namely AMM and MAM (Section III-A);
- We perform the scalability analysis of TPCs of AMM and MAM categories to capture the inter-dependence of the maximum achievable TPC size, bit precision, and operating data rate (Section III-B);
- We map the processing of four state-of-the-art CNNs with mixed-sized tensors on the TPCs of AMM and MAM categories to evaluate the hardware utilization of the TPCs, to consequently establish the need for reconfigurability in the AMM and MAM-styled TPCs (Section IV);
- We invent a novel reconfigurable structure for VDPEs, and utilize these VDPEs to modify the AMM and MAM TPCs, to render these TPCs with the capabilities of dynamic re-aggregation of vectors for adaptive resizing of the processed VDPs (Section V);
- We evaluate the performance of our designed reconfigurable MAM and AMM TPC organizations for the inference of four modern CNNs in terms of Frames-Per-Second (FPS), FPS/W, and compare it with three different AMM- and MAM-styled CNN accelerators from prior work, with area proportionate outlook for which we set the area of all our evaluated accelerators to be equal (Section VI).

II. Preliminaries

A. CNNs with Mixed-Sized Tensors

It has been established that the efficiency of executing CNNs can be improved by drastically reducing the computation, communication, and memory requirements of the CNNs. To achieve this, there has been a growing trend of employing compressed, mixed-sized tensors in CNNs. For example, to compress the size of the utilized tensors, the Xception CNN model [18] introduced depthwise separable convolutions. Typically, a Depthwise Separable Convolution (DSC) breaks up a Standard Convolution (SC) into a Depthwise Convolution (DC) and a subsequent Pointwise Convolution (PC). Fig. 1 demonstrates how a Standard Convolution (SC), Depthwise Convolution (DC), and Pointwise Convolution (PC) work. An SC performs the channel-wise and spatial-wise tensor product computation in a single step by applying a single 3D kernel (convolutional filter) tensor to all the channels of the input tensor. In contrast, a DSC splits the tensor product computation into two steps. In the first step, the constituent DC applies a dedicated 2D kernel tensor per channel of the input tensor, and the channel-wise outputs are stacked to produce a single intermediate tensor. Then, in the second step, the subsequent PC is used to create a linear combination of all the channels of the intermediate tensor by applying a 1D kernel tensor for each spatial point of the intermediate tensor, to consequently produce the final output tensor. This can be better understood by referring to Fig. 1, as explained next.

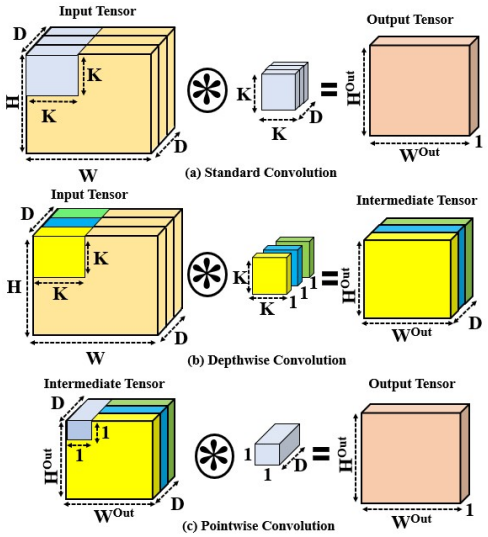

Fig. 1: Illustration of various types of convolutions.

1) Standard Convolutions (SCs): From Fig. 1(a), in SC, the input tensor is $H \times W \times D$ in size, where H is the height, W is the width and D is the number of channels (i.e., the depth). The kernel tensor has dimensions $K \times K \times D$, which is convolved over the input tensor to generate a single output tensor with dimensions $H^{Out} \times W^{Out} \times 1$. If there are F such kernel tensors convolved over the input tensor, then the output tensor will have the dimensions $H^{Out} \times W^{Out} \times F$ (Fig. 1(a) shows F=1). If we define the tensor as a multidimensional array of points, then each point in the output tensor is obtained by performing a tensor product (sum of point-wise products) of the $K \times K \times D$ kernel tensor and the corresponding $K \times K \times D$ part of the input tensor (part of the input tensor highlighted in gray in Fig. 1(a)). In this case, the total number of weight points across all of the F kernel tensors can be given by:

$$W_{SC} = K \times K \times D \times F \tag{1}$$

This W_{SC} defines part of the memory and communication costs of generating the output tensor using SCs. Similarly, the total number of point-wise multiplication operations O_{SC} (a point-wise multiplication between an input tensor point and a weight tensor point produces one point-wise product), required to generate all points in the output tensor, can be given by:

$$O_{SC} = H^{Out} \times W^{Out} \times K \times K \times D \times F \tag{2}$$

This O_{SC} defines the computational cost of generating the output tensor using SCs.

2) Depthwise Separable Convolutions (DSCs): As mentioned earlier, each DSC is typically split into a DC and a subsequent PC. From Fig. 1(b), in DC, each channel of the input tensor has a corresponding 2D kernel tensor with dimensions $K \times K \times 1$. Since there are D channels in the input tensor, there are D such kernel tensors. Convolving these channel-wise kernel tensors across the spatial dimensions of their respective input channels generate a total of D channels of the intermediate tensor with each channel being of dimensions $H^{Out} \times W^{Out} \times 1$. Subsequently, in PC (Fig. 1(c)),

the intermediate tensor is convolved with a pointwise kernel tensor of dimensions $1\times1\times D$ to generate one output tensor of dimensions $H^{Out}\times W^{Out}\times 1$. If there are F such point-wise kernel tensors, then the output tensor will have the dimensions $H^{Out}\times W^{Out}\times F$ (Fig. 1(c) shows F=1). In this case, the total number of weight points across the DC and PC steps, considering F point-wise kernel tensors in the PC step, can be given by:

$$W_{DSC} = K\times K\times D + D\times F \tag{3}$$

Similarly, generating the final output tensor would require the total number of point-wise multiplication operations in the DCs, PCs, and DSCs to be O_{DC}, O_{PC}, and O_{DSC}, respectively, which can be given by:

$$O_{DC} = H^{Out}\times W^{Out}\times K\times K\times D \tag{4}$$

$$O_{PC} = H^{Out}\times W^{Out}\times D\times F \tag{5}$$

$$O_{DSC} = O_{DC} + O_{PC} \tag{6}$$

Thus, for generating the final output tensor, the use of DSCs (DCs+PCs) results in a reduction in the required number of weight points and point-wise multiplication operations. This reduction can be given by the reduction factors R_W and R_O:

$$R_W = \frac{W_{DSC}}{W_{SC}} = \frac{1}{F} + \frac{1}{K^2} \tag{7}$$

$$R_O = \frac{O_{DSC}}{O_{SC}} = \frac{1}{F} + \frac{1}{K^2} \tag{8}$$

It can be inferred that the use DSCs can reduce the memory+communication and computing costs of generating the output tensor by R_W and R_O. Because of this advantage, several state-of-the-art CNN models, such as EfficientNet [21], ShuffleNetV2 [22] and MobileNetV2 [23], adopt DSCs.

B. Accelerating CNN Tensor Products

Modern CNNs employ a large number of input and kernel tensors across all their constituent standard convolutional, depthwise seperable convolutional and fully-connected layers. These CNN tensors are often irregular in size. For example, ResNet50 [24] (EfficientNetB7 [21]) CNN employs kernel tensors of at least 12 (26) different sizes (see the sizes in Table III). Accelerating the inference of such CNNs having mixed-sized tensors requires efficient hardware implementations of the Products (a.k.a. General Matrix Multiplications (GEMMs)) of their input and kernel (weight) tensors. To make the hardware implementations of such CNN Tensor Products feasible, the involved input and kernel tensors are typically decomposed into vectors (1D tensors). These vectors are referred to as Decomposed Input Vectors (DIVs) and Decomposed Kernel Vectors (DKVs), henceforth. Decomposing into DIVs and DKVs in turn transforms the Tensor Products into decomposed vector dot products (VDPs), which can be efficiently performed on VDP hardware accelerators [9]. When implemented on hardware, these decomposed VDPs produce partial results (*psums*), which can then be post-processed to achieve the final Tensor Product results using the *psum* reduction networks employed in the hardware accelerators [9].

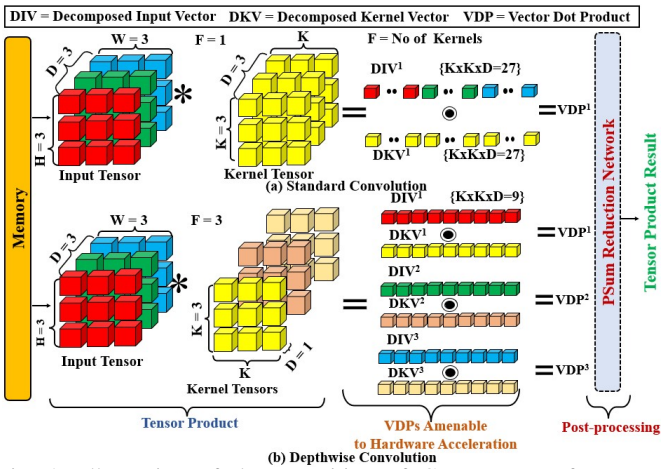

Fig. 2: Illustration of decomposition of CNN tensors for tensor product acceleration, for (a) standard convolution (SC), and (b) depthwise convolution (DC).

To conceive this process of Tensor Product acceleration, consider Fig. 2, which illustrates tensor products between the input and kernel tensors, for a SC (Fig. 2(a)) and a DC (Fig. 2(b)). In Fig 2(a), the tensor product between an input tensor of dimension (H, W, D) = (3, 3, 3) and a single kernel tensor (F = 1) of dimension (K, K, D) = (3, 3, 3) is illustrated. To make this tensor product amenable to hardware acceleration, these input and kernel tensors are respectively flattened into a DIV (DIV^1) and a DKV (DKV^1) of 27 points each (corresponding to K×K×D = 3×3×3). Consequently, the tensor product is converted into a VDP operation between the DIV^1 and DKV^1, which produces VDP^1 as the result. Producing this VDP^1 can be simplified into 27 point-wise multiplication operations to produce 27 point-wise products, which are then summed together using 27 accumulation operations. This makes producing VDP^1 amenable to acceleration on any computing hardware that can support multiple parallel multiplications and accumulations (e.g., [25], [26], [27], [28], [29]). If the number of supported, in-parallel multiplications and accumulations in the employed accelerator hardware is less than the size of the VDP operation (27 in our example), the result VDP^1 has to be decomposed into multiple *psum* results. These *psum* results are then summed together using the *psum* reduction network (Fig. 2), to produce the final tensor product result, at the cost of extra latency. Note that for the example illustrated in Fig. 2, we assume that the accelerator hardware size matches with the VDP operation size.

In contrast, as discussed earlier, performing a DC operation on the same input tensor of dimension (H, W, D)=(3, 3, 3) requires 3 channel-wise kernel tensors (F=3) of dimension (K, K, D) = (3, 3, 1) each. The need to use 3 channel-wise kernel tensors necessitates that a total of 3 tensor products are obtained. For that, as shown in Fig. 2(b), the 3 channels of the input tensor are flattened into 3 DIVs (DIV^1, DIV^2, DIV^3), and the 3 channel-wise kernel tensors are flattened into 3 DKVs (DKV^1, DKV^2, DKV^3), of 9 points each (corresponding to K×K×D = 3×3×1). Performing VDP operations

between respective DIVs and DKVs produces 3 VDP results (VDP^1, VDP^2, VDP^3). Thus, compared to an SC operation, a DC operation renders smaller sized DIVs and DKVs. This in turn reduces the parallelism requirement per VDP result in the employed accelerator hardware, because implementing the VDP operations between the reduced sized DIVs and DKVs requires the hardware support for a less number of in-parallel multiplications and accumulations per VDP result (only 9 in our example of DC). If the employed accelerator hardware supports more parallelism than necessary for implementing a DC operation, it can lead to lower hardware utilization efficiency. Similar to the SC and DC operations, it is also common to convert a PC operation into multiple VDP operations to make it amenable to hardware acceleration. *In summary*, to make processing of CNN tensor products amenable to hardware-based acceleration, this process of tensor flattening and decomposition remains consistent for the input and kernel tensors of arbitrary sizes (for sizes other than considered in Fig. 2 as well), across all types of convolutional and fully-connected CNN layers.

C. Related Work on Photonic CNN Accelerators

To accelerate machine learning tasks with low latency and low energy consumption, prior works have proposed various accelerators based on Photonic Integrated Circuits (PICs) (e.g., [8]–[11], [14], [15], [30], [31]). Among these, the CNN accelerators employ PIC-based TPCs to perform CNN tensor products. Some accelerators implement digital TPCs (e.g., [32], [33]), whereas some others employ analog TPCs (e.g., [7]–[9], [17]). Different TPC implementations employ MRRs (e.g., [9]–[11], [16], [34]) or MZIs (e.g., [12], [14], [30]) or both (e.g., [32]). The analog TPCs can be further classified as incoherent (e.g., [8], [9], [11]) or coherent (e.g., [12], [35]–[40]). To set and update the values of the individual input and kernel tensors used for tensor products, the incoherent TPCs utilize the analog optical signal power, whereas the coherent TPCs utilize the electrical field amplitude and phase. The coherent TPCs achieve low inference latency, but they suffer from control complexity, high area overhead, low scalability, low flexibility, high encoding noise, and phase error accumulation issues [41]. In contrast, the incoherent TPCs based accelerators achieve better scalability and lower footprint, because they use MRR-based compact PICs [9], unlike the coherent TPCs that use MZI-based bulky PICs.

Various state-of-the-art photonic CNN accelerators are well discussed in a survey paper [42]. Because of the inherent advantages of MRR-enabled incoherent TPCs, there is impetus to design more energy-efficient and scalable CNN accelerators employing MRR-enabled incoherent TPCs. However, the CNN accelerators from prior works have mainly focused on designing their constituent TPCs only for the standard convolutional layers of CNNs. Prior works have paid very little attention to accelerate the processing of the depthwise separable convolutional layers of CNNs. In this paper, we contribute towards making the MRR-enabled incoherent TPCs more efficient by enabling them to dynamically adapt to process the standard convolutional and depthwise separable convolutional layers of CNNs.

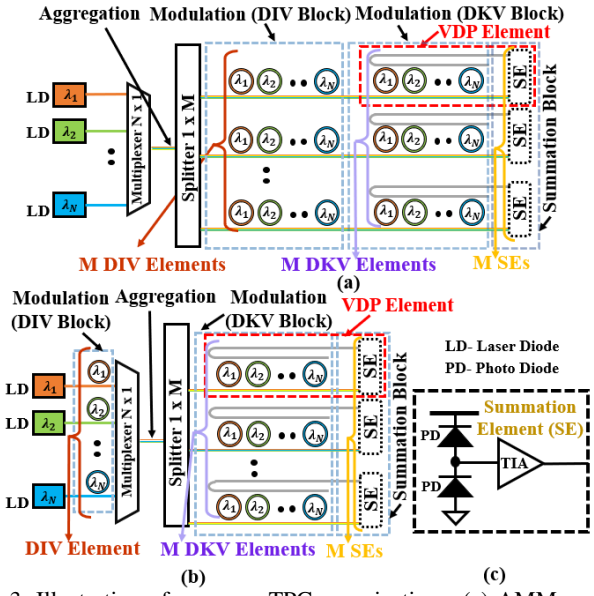

Fig. 3: Illustration of common TPC organizations. (a) AMM organization, (b) MAM organization, and (c) Summation Element.

III. Classification and Scalability Analysis of TPC Organizations

A. Classification

Most of the photonic MRR-enabled analog, incoherent CNN accelerators from prior works employ multiple analog TPCs that work in parallel. Typically, an analog TPC implements the decomposed VDP operations of a tensor product (Fig. 2; Section II-B). For that, the TPC typically employs a total of five blocks (Fig. 3(a)): *(i)* a laser block that employs N Laser Diodes (LDs) to generate N optical wavelength channels; *(ii)* an aggregation block that aggregates the generated optical wavelength channels into a single photonic waveguide through DWDM (using an $N\times1$ multiplexer) and then splits the optical power of these N wavelength channels equally into M separate waveguides (using a $1\times M$ splitter); *(iii)* a modulation block, also referred to as DIV block, that employs M arrays of MRRs (one array per waveguide, with each array having N MRRs; each array referred to as DIV element) to imprint M DIVs of N points each onto the $N\times M$ wavelength channels by modulating the analog power amplitudes of the wavelength channels; *(iv)* another modulation block, referred to as DKV block, that employs another M arrays of MRRs (one array per waveguide, with each array having N MRRs; each array referred to as DKV element) to further modulate the $N\times M$ wavelength channels with DKVs, so that the analog power amplitudes of the individual wavelength channels then represent the point-wise products of the utilized DKVs and DIVs; and *(v)* a Summation Block (SB) that employs a total of M Summation Elements (SEs), with each SE having two balanced Photodiodes (PDs) upon which the point-wise-product-modulated N wavelength channels are incident to produce the output current that is proportional to the VDP result (i.e., the sum of the N input point-wise products). The laser block and SB are typically positioned at the two ends of the TPC, with the aggregation,

TABLE I: List of abbreviations and their full forms used in this paper. Definition and values of various parameters (obtained from [43]) used in Eq. 9, Eq. 10, and Eq. 11 for the scalability analysis of AMM and MAM TPCs.

Abbreviations	Full form	Parameter	Definition	Value
TPC	Tensor Processing Core	P_{Laser}	Laser Power Intensity	10 dBm
DSC	Depthwise Separable Convolution	R	PD Responsivity	1.2 A/W
SC	Standard Convolution	R_L	Load Resistance	50 Ω
PC	Pointwise Convolution	I_d	Dark Current	35 nA
DC	Depthwise Convolution	T	Absolute Temperature	300 K
DKV	Decomposed Kernel Vector	DR	Data Rate	10 GS/s
DIV	Decomposed Input Vector	RIN	Relative Intensity Noise	-140 dB/Hz
VDP	Vector Dot Product	η_{WPE}	Wall Plug Efficiency	0.1
S	Size of DKV	IL_{SMF}(dB)	Single Mode Fiber Insertion Loss	0
DKV_S	DKV of size S	IL_{EC}(dB)	Fiber to Chip Coupling Insertion Loss	1.6
F_S	Set of DKVs with size S	IL_{WG}(dB/mm)	Silicon Waveguide Insertion Loss	0.3
DR	Data rate	$EL_{Splitter}$(dB)	Splitter Insertion Loss	0.01
VDPE	Vector Dot Product Element	IL_{MRM}(dB)	Microring Modulator (MRM) Insertion Loss	4
N	Size of VDPE	OBL_{MRM}(dB)	Out of Band Loss MRM	0.01
M	Number of VDPEs per TPC Unit	IL_{MRR}(dB)	Microring Resonator (MRR) Insertion Loss	0.01
SE	Summation Element	$IL_{penalty}$(dB) (MAM)	Network Penalty	4.8
CS	Comb Switch	$IL_{penalty}$(dB) (AMM)	Network Penalty	5.8
y	Number of Comb Switches	d_{MRR}	Gap between two adjacent MRRs	20 μm
x	Re-aggregating or Filtering wavelength count	$d_{element}$ (MAM)	Thermal isolation spacing between DIV and DKV elements	0 μm
L	Set of re-aggregated wavelengths	$d_{element}$ (AMM)		100 μm

modulation (DIV), and modulation (DKV) blocks placed in between them.

Based on the order in which these intermediate blocks (aggregation, modulation (DIV), modulation (DKV) blocks) are positioned between the laser block and SB, we classify the MRR-based TPC organizations from prior work as MAM (Modulation, Aggregation, Modulation) [9] or AMM (Aggregation, Modulation, Modulation) [15]. Fig. 3 illustrates MAM and AMM TPC organizations. From Fig. 3(a), the AMM TPC organization positions the aggregation block first, and then the DIV block followed by the DKV block. In contrast, the MAM TPC in Fig. 3(b) positions the DIV block first, and then positions the aggregation block followed by the DKV block. Note that the MAM-styled DIV block is structurally different from the AMM-styled DIV block. The MAM-styled DIV block employs only one MRR per waveguide, and as a result, it can imprint only 1 DIV of *N* points onto the *N* wavelength channels. This 1 DIV is shared among all DKVs in the MAM TPC, whereas each DKV can have a different DIV corresponding to it in the AMM TPC.

Moreover, note that each DKV element in both MAM and AMM TPCs have two waveguides (shown but not labelled in the figures). First, the drop waveguide, coupling with the MRRs at the top. Second, the through waveguide, coupling with the MRRs at the bottom. The wavelengths that carry negatively-signed pointwise products have more guided optical power in the drop waveguide, whereas the wavelengths that carry positively-signed pointwise products have more guided optical power in the through waveguide. The drop waveguide makes its guided power incident upon the top-side PD in the corresponding SE, whereas the through waveguide makes its guided power incident upon the bottom-side PD. This enables a signed accumulation of the pointwise products carried by the guided wavelengths, because the top-side and bottom-side PDs in the SE are balanced [15]. In both the AMM and MAM TPC organizations, we refer to the combination of a DKV element and the corresponding SE as VDP element (VDPE). A VDPE size (i.e., *N*) should match the DKV size for efficient, low-overhead implementation of the VDP operation.

B. Scalability Analysis

Prior works [43] and [33] have shown that the scalability of photonic accelerator architectures, in terms of the achievable VDPE size *N*, decreases with the increase in the required bit precision. However, these prior works lack in two ways. First, they do not capture the impact of the utilized data rate on the inter-dependence between *N* and bit precision. Second, they do not provide the scalability analysis for AMM TPC architectures (they only analyze MAM TPCs). To address these shortcomings, we extend the methodology from [43] to perform the scalability analysis of the AMM and MAM TPCs to capture the inter-dependence of VDPE size *N*, number of VDPEs per TPC *M*, bit precision, and data rate. From [43], it is known that the bit precision of an MRR-based VDPE depends on the output photodetector sensitivity (P_{PD-opt}) and data rate (DR) [43]. In this paper, we evaluate the required P_{PD-opt} for various bit precision values and DRs by solving Eq. 9 [43]. We sweep for DRs of 1, 3, 5, and 10 GS/s (Giga-Symbols per sec), and sweep for bit precision values of 1-bit to 8-bit, and consider *M* = *N* for our analysis. The value of *N* is obtained by solving Eq. 11 [43], along with P_{PD-opt} for a given bit precision and DR obtained from Eq. 9 and Eq. 10 [43].

$$n_{i/p} = \frac{1}{6.02}\left[20log_{10}(\frac{R\times P_{PD-opt}}{\beta\sqrt{\frac{DR}{\sqrt{2}}}} - 1.76\right] \quad (9)$$

$$\beta = \sqrt{2q(RP_{PD-opt}+I_d)+\frac{4kT}{R_L}+R^2P_{PD-opt}^2 RIN} \quad (10)$$

$$P_{Laser} = \frac{10^{\frac{\eta_{WG}(dB)[N(d_{MRR})+d_{element}]}{10}}M}{\eta_{SMF}\eta_{EC}IL_{i/p-MRM}} \times \frac{P_{PD-opt}}{\eta_{WPE}IL_{MRR}} \times \frac{1}{(OBL_{MRM})^{N-1}(EL_{splitter})^{log_2 M}} \times \frac{1}{(OBL_{MRR})^{N-1}(IL_{penalty})} \quad (11)$$

Table I reports the definitions of the parameters and their values from Eq. 9, Eq. 10, and Eq. 11, as used for our analysis. In Table I, $P_{penalty}$ represents the impairments due to extinction ratio, crosstalk, inter-symbol interference (ISI), and laser relative intensity noise (RIN). We perform this analysis for both AMM and MAM TPC architectures. As discussed in section III-A, AMM and MAM TPCs differ in the placement of the DKV and DIV elements. In MAM TPCs, all DKV elements share a single DIV element, whereas in AMM TPCs all DKV elements have their individual DIV elements. Therefore, to avoid thermal crosstalk in AMM TPCs, the DKV elements need to be placed sufficiently farther from their respective DIV elements [44], which in turn increases the required $d_{element}$ for AMM TPCs (Table I), thereby increasing the optical lengths of the waveguides in AMM TPCs. Due to the longer waveguides, the $IL_{penalty}$ increases for AMM TPCs compared to MAM TPCs (Table I). This in turn affects the achievable VDPE size *N*, DR, and bit precision for AMM TPCs, as confirmed by our below-presented analysis results.

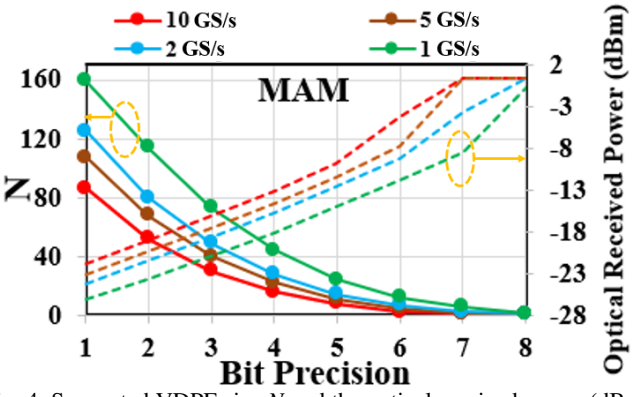

Fig. 4: Supported VDPE size *N* and the optical received power (dBm) for bit precision ={1, 2, 3, 4, 5, 6, 7, 8}bits at data rates (DRs) = {1, 3, 5, 10}GS/s, for MAM TPCs.

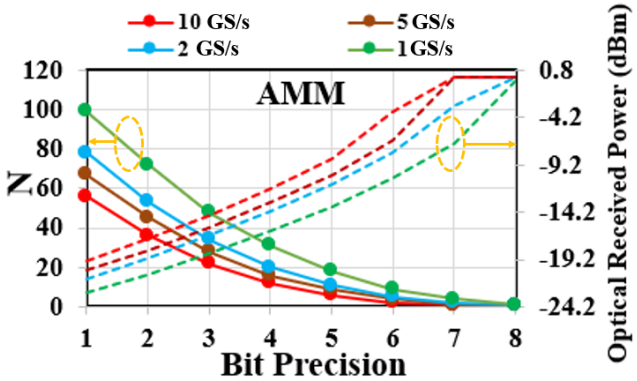

Fig. 5: Supported VDPE size *N* and the optical received power (dBm) for bit precision ={1, 2, 3, 4, 5, 6, 7, 8}bits at data rates (DRs) = {1, 3, 5, 10}GS/s, for AMM TPCs.

Fig. 4 and Fig. 5 show the inter-dependence of VDPE size *N*, bit precision, and DR for MAM and AMM TPCs respectively. From Fig. 4, for MAM TPCs, the maximum VDPE size *N* significantly decreases from *N* = *159* for 1-bit precision to *N* = *1* for 8-bit precision. Similarly, from Fig. 5, for AMM TPCs, the supported *N* drops from *N* = *99* for 1-bit precision to *N* = *1* for 8-bit precision. This trend of decreasing *N* with increasing bit precision is in line with the similar trend observed in recent work Albireo [33], although note that the accelerator architecture of Albireo is different from our considered AMM and MAM architectures. Moreover, we can also observe from Fig. 4 and Fig. 5 that the supported *N* also varies with DR. With the increase in DR, the supported *N* decreases in both MAM and AMM TPCs. For instance, MAM TPCs' supported *N* for 4-bit precision drops from *N* = *44* at 1 GS/s to *N* = *16* at 10 GS/s. In case of AMM TPCs, at 4-bit precision, the supported *N* drops from *N* = *31* at 1 GS/s to *N* = *12* at 10 GS/s. For given bit precision and DR values, AMM TPCs support lower *N* compared to MAM TPCs. This is because, as discussed earlier, AMM TPCs incur higher $IL_{penalty}$. From our analysis, it can be inferred that AMM and MAM TPCs cannot support any *N* for 8-bit precision; therefore, we advocate that AMM and MAM TPCs should be utilized to achieve 4-bit precision at the highest, to select such a power-of-two precision value below the unattainable 8-bit precision that can support a tangible *N* value. Our obtained values of *N* for different DRs are given in Table II.

TABLE II: VDPE size (*N*) at 4-bit precision across various DRs for different TPC architectures.

TPC Architectures	DR (GS/s)			
	1	3	5	10
RMAM	43	27	22	16
RAMM	31	20	16	12
MAM (HOLYLIGHT [9])	44	28	22	16
AMM (DEAPCNN [15])	31	20	16	12

IV. Need for Reconfigurability in TPCs

A feasible acceleration of CNN tensor products on MRR-enabled incoherent, analog TPCs mandates that a CNN kernel tensor of shape (K, K, D) is decomposed/flattened into a 1D DKV of size *S* = K×K×D. From Fig. 2, the corresponding DIV should also be of size *S*. In modern CNNs, the value *S* corresponding to various kernel tensors vary drastically. Table III provides kernel tensor shapes and corresponding DKV sizes (*S*) for EfficientNetB7 [21]. We selected EfficientNetB7 as an example of CNNs with a large number of DCs, PCs, and SCs. From Table III, DCs have a small set of DKV sizes ($S \in \{9, 25\}$). In contrast, in Table III, PCs have a wide range of *S* values, from as small as 8 to as large as 3840. As discussed earlier (Section II-B), for the processing of PCs and DCs using TPCs to produce the final tensor products, their corresponding DKVs are mapped onto the constituent VDPEs of the TPCs so that the tensor products are converted into VDP operations. Therefore, depending on how the size *S* of a DKV compares with the size *N* of a VDPE, the following three scenarios arise for the mapping of the DKV onto the VDPE to produce the final VDP result:

- **Scenario 1, *S*=*N***: For this case, all the DKV points have one-to-one mapping with all the MRRs of the VDPE, and there are no idle MRRs in the VDPE. The VDP result will be the final tensor product result.
- **Scenario 2, *S*>*N***: In this case, a single VDPE cannot produce the final tensor product result, as it cannot process the entire DKV. Therefore, the DKV needs to be further decomposed into a total *P* partial DKVs requiring a total of *P* = *Ceil*(*S*/*N*) VDPEs to produce the final tensor

product result. All P VDPEs produce partial VDP results called *psum*s, and during post-processing, a reduction network accumulates these *psum*s to generate the final tensor product result. Although this accumulation of *psum*s incurs additional latency, which can be efficiently hidden by employing a pipelined design of the *psum* reduction network with non-blocking bandwidth [45]. One drawback, however, is that if S is not an integer multiple of N, then this scenario leads to some unutilized MRRs in the VDPEs across P partial VDP operations.

- **Scenario 3, $S<N$**: In this case, the result of the single VDP operation provides the final tensor product result, but some MRRs in the VDPE remain unutilized. The count of unutilized MRRs depends on the size difference between the DKV and VDPE (i.e., between S and N).

For the last two scenarios (for which $S \neq N$), the unutilized MRRs cause underutilization in the VDPEs. This hampers the performance and efficiency of processing modern CNNs with mixed-sized tensors. This is because the unutilized MRRs incur area and static power overheads while also idling away the opportunity for increasing the processing throughput. Therefore, how well N matches with S plays a crucial role in determining the performance and efficiency of a TPC. To that end, we reason that dynamic flexibility in the supported value of N in the VDPEs is required to efficiently support processing of various sizes of DCs, PCs, and SCs of modern CNNs.

TABLE III: Kernel tensor shapes (K, K, D), the total number of such kernels (F) and corresponding DKV sizes (S) for EfficientNet_B7 [21], as an example CNN with a large number of DSCs. (FC=Fully Connected Layer and other abbreviations are defined in Table I). The K, D, F values were extracted from Keras Applications [46].

Model	Convolution	Tensor Shape (K, K, D)	F	S
EfficientNet_B7	DC	(3, 3, 1)	25024	9
	DC	(5, 5, 1)	45216	25
	PC	(1, 1, 8)	288	8
	PC	(1, 1, 12)	2016	12
	PC	(1, 1, 16)	64	16
	PC	(1, 1, 20)	3360	20
	PC	(1, 1, 32)	312	32
	PC	(1, 1, 40)	9600	40
	PC	(1, 1, 48)	2016	48
	PC	(1, 1, 56)	13440	56
	PC	(1, 1, 64)	48	64
	PC	(1, 1, 80)	3360	80
	PC	(1, 1, 96)	29952	96
	PC	(1, 1, 160)	21120	160
	PC	(1, 1, 192)	56	192
	PC	(1, 1, 224)	13440	224
	PC	(1, 1, 288)	452	288
	PC	(1, 1, 384)	29952	384
	PC	(1, 1, 480)	780	480
	PC	(1, 1, 640)	14080	640
	PC	(1, 1, 960)	2064	960
	PC	(1, 1, 1344)	2960	1344
	PC	(1, 1, 2304)	6496	2304
	PC	(1, 1, 3840)	2400	3840
	SC	(3, 3, 3)	64	27
	FC	(2560, 1, 1)	1	2560

A. Perils of Fixed VDPE Size (N) in TPCs from Prior Work

To validate our reasoning presented just before this subsection, we evaluated the hardware utilization values for various TPC architectures (with fixed N) from prior works. For example, MAM (HOLYLIGHT [9]), AMM (DEAPCNN [15]) have N=43 and N=31, respectively. For these TPCs, Fig. 6 shows hardware (MRR) utilization per-VDPE in terms of the ratio (in %) of the utilized VDPE area over the total (utilized+idle) area. The figure also shows the utilization for our proposed Reconfigurable MAM (RMAM) and Reconfigurable AMM (RAMM) TPCs. But we introduce and discuss our RAMM and RMAM TPCs in Section V, so please look past their excellent utilization results for now. From Fig. 6, MAM (HOLYLIGHT [9]), and AMM (DEAPCNN [15]) yield significantly low per-VDPE utilization (as low as 8%). This is because of the huge mismatch between N and S (i.e., $S<N$) while processing DCs and PCs. Such low VDPE utilization can hamper performance and efficiency of the TPCs, as discussed earlier.

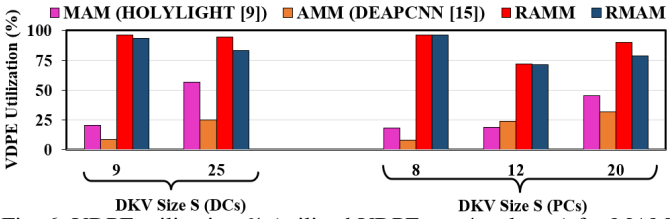

Fig. 6: VDPE utilization % (utilized VDPE area/total area) for MAM (HOLYLIGHT [9],N=44), AMM (DEAPCNN [15],N=31), RAMM (N=31), and RMAM (N=43) at DR=1GS/s and 4-bit precision for various DKV sizes corresponding to DCs and PCs.

These results motivate the need to design a VDPE that can dynamically adapt to various DKV sizes. Therefore, we invented a novel reconfigurable VDPE design, which is described in the next section.

V. MRR-based Reconfigurable TPC Architectures

In this section, we present a novel reconfigurable VDPE which serves as the backbone of our proposed MRR-based reconfigurable TPC architectures. This reconfigurable VDPE adds the following two desirable attributes to the AMM and MAM types of TPCs. *First*, it introduces the flexibility to the TPCs so that the processing of the DKVs of various sizes can be efficiently mapped onto them, regardless of the fixed VDPE size N for the TPCs. *Second*, it introduces opportunities for increasing the processing parallelism and MRR utilization efficiency. This reconfigurable VDPE can directly replace the VDPEs of the MAM and AMM TPCs (Fig. 3), to convert them into reconfigurable MAM (we refer to it as RMAM, henceforth) and reconfigurable AMM (we refer to it as RAMM, henceforth) TPCs. The structure and operation of our invented reconfigurable VDPE are discussed in the following subsections.

A. Reconfigurable VDPE: Structure and Layout

Fig. 7 illustrates our proposed reconfigurable VDPE. It consists of a DKV element, which is an array of N modulation MRRs that can imprint N pointwise products onto the incoming N wavelength channels, similar to the DKV elements of the MAM and AMM TPCs from Fig. 3. This DKV element is followed by a group of a total of y pairs of MRR comb switches (CSs). These y CS pairs can re-aggregate the incoming N pointwise-product-modulated wavelength channels

(incoming from the DKV element) into a total of y distinct sets, with each set L (Fig. 7) having a total of x distinct wavelength channels. In other words, each CS pair filters a comb (set L; Fig. 7) of x distinct wavelength channels from the incoming N wavelength channels. Each CS pair is able to do this because of its innate spectral response that allows it to be in resonance with x distinct wavelengths simultaneously (more on the design of CSs in Section V.C). Each CS pair then sends its corresponding x wavelength channels to its dedicated summation element (SE), which performs a signed accumulation of the data carried on the x wavelength channels to produce a VDP result. To enable the signed accumulation, similar to the SEs of the AMM and MAM TPCs (Fig. 3), the SEs corresponding to the CS pairs also employ balanced PDs. Thus, the group of y CS pairs per reconfigurable VDPE enables y VDP results of size x each to be produced in parallel.

Note that here x, y, and N are integers, and their relation is given by this equation: $\boldsymbol{y = N > 2x \ ? \ floor(N/x) \ : \ 0}$. Henceforth, we refer to x as re-aggregation size. Also note that for the group of CS pairs to produce y in-parallel VDP results, each CS pair in the group has to be switched ON by electro-optically tuning its spectral resonance passbands to align with its corresponding set of x wavelength channels [47]. In contrast, it is also possible to switch OFF a CS pair by tuning its resonance passbands out of alignment with the corresponding x wavelength channels. When the CS pairs are switched OFF, all of the N incoming wavelength channels are allowed to pass by the CS pairs to be eventually accumulated at the summation element SE^N. Thus, depending on whether the CS pairs are ON or OFF, the reconfigurable VDPE can operate in two different modes. These modes are further explained in the next section.

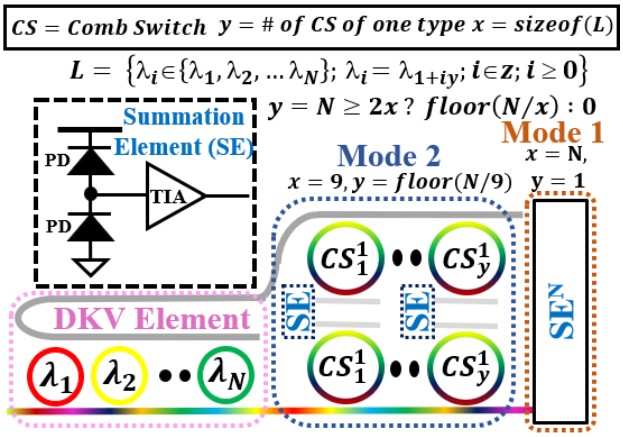

Fig. 7: Schematic of our invented reconfigurable VDPE, employing a DKV element and one group of comb switch (CS) pairs corresponding to the reconfiguration Mode 2.

B. Reconfigurable VDPE: Operation

The reconfigurable VDPEs of our RMAM and RAMM TPC architectures support two operational modes, i.e., Mode 1 and Mode 2 (Fig. 7). Mode 1 is the non-reconfiguration mode, in which all the CS pairs of a reconfigurable VDPE are switched OFF, so that the reconfigurable VDPE operates like a regular VDPE to produce a VDP result of size N. In contrast, Mode 2 is the reconfiguration mode, in which all the CS pairs of

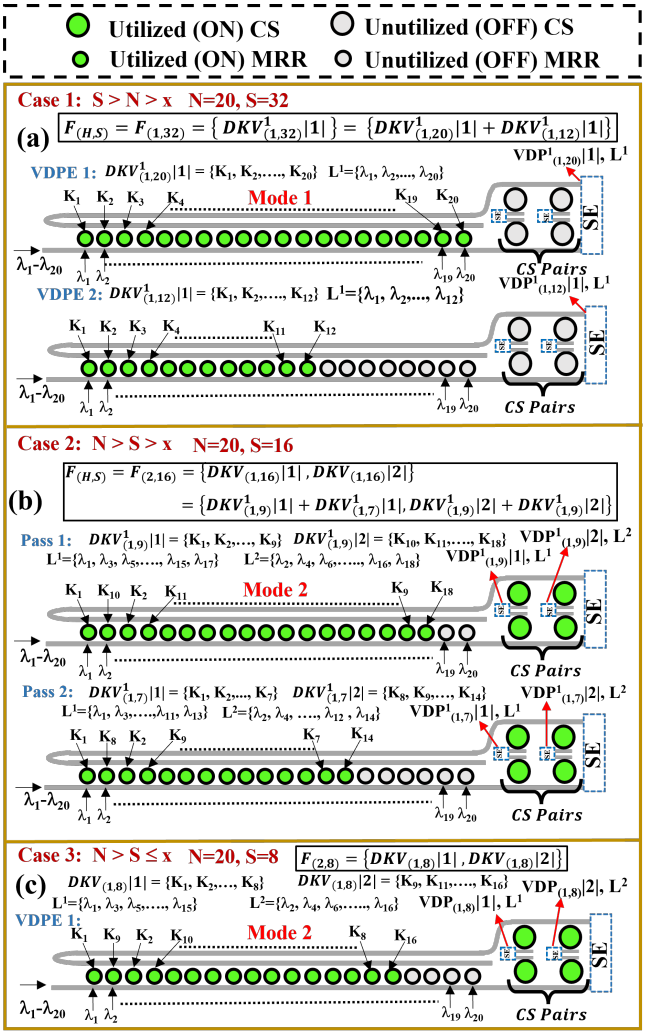

Fig. 8: Example operation of our reconfigurable VDPE for various cases depending on the S and N values, for x=9. Here, ON and OFF CS pairs, respectively, represent Mode 2 and Mode 1 of operation.

a reconfigurable VDPE are switched ON so that a total of y in-parallel VDP results can be produced with each result being of size x. We determine the re-aggregation size x to be 9 because the DKV size s=9 is the most common, frequently used, smallest DKV size across various CNNs (Table III). We reason that determining the value of x based on the most common, smallest DKV size maximizes the opportunities for increasing processing parallelism and utilization efficiency.

Since the reconfigurable VDPEs simply replace the regular VDPEs of the MAM and AMM TPCs, an RAMM/RMAM TPC is analogous in structure to an AMM/MAM TPC. From Fig. 3 in Section III, an AMM/MAM TPC contains a VDPE block (containing multiple VDPEs), which is basically the structure that remains in the TPC if we mask off the DIV element of the TPC. Similarly, an RAMM/RMAM TPC would also typically contain a block of M reconfigurable VDPEs. Since each reconfigurable VDPE is of size N, such reconfigurable VDPE block would basically be of $M \times N$ dimensions.

From Section II and III, the input kernel tensors are flattened into DKVs of size S and then mapped onto the VDPE block of a TPC for processing. In that vein, to further explain our

method of mapping for RAMM/RMAM TPCs, typically, a matrix $F_{(H,S)}$, which has *H* rows with each row containing a DKV of size *S*, is mapped onto one or multiple $M \times N$ sized reconfigurable VDPE blocks. This matrix $F_{(H,S)}$ can be written as a set of a total of *H* $1 \times S$ sized DKVs, i.e., $F_{(H,S)}$ = $\{DKV_{(1,S)}|1|, \{DKV_{(1,S)}|2|, ..., DKV_{(1,S)}|H|\}$. Each individual DKV in this set is basically a flattened kernel tensor. When matrix $F_{(H,S)}$ is mapped onto reconfigurable VDPE blocks, this set of DKVs is mapped onto the individual reconfigurable VDPEs of the blocks. After this mapping, the individual reconfigurable VDPEs operate in either Mode 1 or Mode 2, depending on how the size *S* of the DKVs compares with the size *N* of the reconfigurable VDPEs, given that the VDPEs have the re-aggregation size *x=9*.

We advocate for selecting the most appropriate mapping and mode of operation (from Mode 1 or Mode 2) that can maximize the MRR utilization and processing throughput of the RAMM/RMAM TPCs. We identify three cases for the relation among *N*, *S*, and *x=9* that drive the selection of the appropriate mapping and mode of operation. The mappings and modes of operation for these three cases are illustrated in Fig. 8. For Fig. 8, we selected the example RAMM TPC architecture from Table II with *N=20* for DR = 3 GS/s. These illustrations are further explained below.

Case 1, *S>N>x*: For this case, the reconfigurable VDPEs operate in Mode 1. In this case, before mapping the DKV matrix $F_{(H,S)}$ on the reconfigurable VDPEs, matrix $F_{(H,S)}$ is divided into multiple slices along the dimension *S*. Since *S* can be written as *S=b×N+c*, the matrix $F_{(H,S)}$ is divided into a total of *b+1* slices, with *b* slices of size (*H*, *N*) each and one slice of size (*H*, *c*). Hence, $F_{(H,S)}$ can be written as $\{F^1_{(H,N)}+..+F^b_{(H,N)}+F^1_{(H,c)}\}$, where $F^1_{(H,N)}$,..., $F^b_{(H,N)}$, and $F^1_{(H,c)}$ are the slices of matrix $F_{(H,S)}$. Here, '+' represents the concatenation operator. Since matrix $F_{(H,S)}$ can be written as $\{DKV_{(1,S)}|1|, .., DKV_{(1,S)}|H|\}$ (as discussed earlier), slicing of $F_{(H,S)}$ in turn means slicing of all the DKV components of $F_{(H,S)}$. Consequently, every component DKV $DKV_{(1,S)}|H|$ of $F_{(H,S)}$ is also divided into *b+1* slices along the dimension *S*. Hence, $DKV_{(1,S)}|H|$ can be written as $\{DKV^1_{(1,N)}|H|+..+DKV^b_{(1,N)}|H|+DKV^1_{(1,c)}|H|\}$. Similarly, each DKV of matrix $F_{(H,S)}$ can be re-written as the concatenation of DKV slices. After the matrix $F_{(H,S)}$ has been sliced, each individual slice of every DKV of $F_{(H,S)}$ is mapped onto one reconfigurable VDPE running in Mode 1 operation. Each VDPE generates a partial VDP result corresponding to the mapped DKV slice. The partial VDP results from all the DKV slices that originated from a single original DKV (a component of $F_{(H,S)}$) are then accumulated at the *psum* reduction network (not shown in the figure), to generate the final VDP result. Case 1 is shown in Fig. 8(a), where input matrix $F_{(H,S)}$ with *H=1* and *S=32* is sliced based on *b=1*, *c=12*. The consequently generated individual DKV slices are then mapped on to reconfigurable VDPE1 and VDPE2 operating in Mode 1. The two VDPEs produce two partial VDP results $VDP^1_{(1,20)}|1|$ and $VDP^1_{(1,12)}|1|$. These partial VDP results are then summed together.

Case 2, *N>S>x*: For this case, the reconfigurable VDPEs operate in Mode 2. In this case, before mapping the DKV matrix $F_{(H,S)}$ onto the reconfigurable VDPEs, matrix $F_{(H,S)}$ is divided into multiple slices along the dimension *S*. Since *S* can be written as *S=b×x+c* (the values *b* and *c* here are often different from Case 1), the matrix $F_{(H,S)}$ is divided into a total of *b+1* slices, with *b* slices of size (*H*, *x*) each and one slice of size (*H*, *c*). Hence, $F_{(H,S)}$ can be written as $\{F^1_{(H,x)}+..+F^b_{(H,x)}+F^1_{(H,c)}\}$, where $F^1_{(H,x)}$,..., $F^b_{(H,x)}$, and $F^1_{(H,c)}$ are the slices of matrix $F_{(H,S)}$. Here, '+' represents the concatenation operator. Since matrix $F_{(H,S)}$ can be written as $\{DKV_{(1,S)}|1|, .., DKV_{(1,S)}|H|\}$ (as discussed earlier), slicing of $F_{(H,S)}$ in turn means slicing of all the DKV components of $F_{(H,S)}$. Consequently, every component $DKV_{(1,S)}|H|$ of $F_{(H,S)}$ is also divided into *b+1* slices along the dimension *S*. Hence, $DKV_{(1,S)}|H|$ can be re-written as $\{DKV^1_{(1,x)}|H|+..+DKV^b_{(1,x)}|H|+DKV^1_{(1,c)}|H|\}$. After the matrix $F_{(H,S)}$ has been sliced, the total of *H* DKV slices that belong to every matrix slice $F^1_{(H,x)}$ are mapped onto a total of *(H/y)* different reconfigurable VDPEs, where each reconfigurable VDPE processes a total of *y* slices ($y = N \geq 2x?floor(N/9) : 0$). Hence, each reconfigurable VDPE in this case produces a total of *y* VDP results in parallel, which basically renders a $ytimes$ throughput improvement for each reconfigurable VDPE, compared to Mode 1 of operation. In this case, once a given matrix slice has been mapped onto the total of *(H/y)* VDPEs, the partial VDP results are produced for all DIVs from the input CNN layer that correspond to the mapped matrix slice in a stationary weight dataflow, before the next matrix slice is mapped for a new pass of all the DIVs. The partial VDP results from all the DKV slices that originated from a single original DKV (a component of $F_{(H,S)}$) are then accumulated at the *psum* reduction network, to generate the final VDP result. Case 2 is shown in Fig. 8(b), where input matrix $F_{(H,S)}$ with *H=2* and *S=16* is sliced based on *b=1*, *c=7*. The consequently generated individual DKV slices are then mapped on to a single VDPE operating in Mode 2 in two passes. Pass 1 generates $VDP^1_{(1,9)}|1|$ and $VDP^1_{(1,9)}|2|$, Whereas Pass 2 generates $VDP^1_{(1,7)}|1|$ and $VDP^1_{(1,7)}|2|$. The partial VDP results from Pass 1 are then summed together with respective partial VDP results from Pass 2 to generate two final VDP results.

Case 3, *N>S≤x*: For this case, the reconfigurable VDPEs operate in Mode 2. The input matrix $F_{(H,S)}$ is directly mapped onto the individual reconfiguration VDPEs, after writing it in terms of DKVs as $F_{(H,S)} = \{DKV_{(1,S)}|1|, .., DKV_{(1,S)}|H|\}$. Here, H DKVs corresponding to matrix $F_{(H,S)}$ are mapped onto a total of (*H/y*) reconfigurable VDPEs operating in Mode 2, where each VDPE processes a total of *y* DKVs in parallel. Case 3 is shown in Fig. 8(c), where input matrix $F_{(H,S)}$ with *H=2* and *S=8* is mapped on to a reconfigurable VDPE operating in Mode 2. The VDPE processes *y=2* DKVs, generating *y=2* VDP final results $VDP_{(1,8)}|1|$ and $VDP_{(1,8)}|2|$.

Discussion: Compared to Mode 1, Mode 2 operation of the reconfigurable VDPE increases the hardware/MRR utilization, while simultaneously increasing the processing throughput by up to $y\times$. This throughput improvement makes it tolerable to have the extra area overhead of additional *y* CS pairs per

reconfigurable VDPE for Mode 2 operation. It can be argued that $y\times$ improvement in throughput can also be achieved by employing $y\times$ more VDPEs (without CS pairs) in Mode 1 operation. However, employing $y\times$ more VDPEs in Mode 1 operation incurs the area overhead of additional $N\times y$ MRRs (as each VDPE has N MRRs), whereas employing y CS pairs in one reconfigurable VDPE incurs the area overhead equivalent to additional $6\times y$ MRRs (as the area of 1 CS pair = area of 6 MRRs). Since for all TPCs from Table II, $6\times y$ is less than $N\times y$, Mode 2 operation turns out to be highly efficient than Mode 1 operation. As for modern CNNs, more than 40% of the DKVs (Table III) belong to Case 2 ($N>S>x$) and Case 3 ($N>S\leq x$) above, the substantially improved efficiency for Mode 2 operation also translates in up to 78.2% and 54.71% higher VDPE utilization, respectively, for our RAMM and RMAM TPCs in Fig. 6, compared to their respective baseline AMM (DEAPCNN) and MAM (HOLYLIGHT) TPCs.

C. Design of MRR Comb Switches

Compared to the modulation MRRs of the DIV and DKV elements (Fig. 3), our utilized MRR-based CSs (Fig. 7) typically have a larger ring radius [47]. From Section V.A, a CS has the capability of filtering a comb of x wavelength channels from the incoming N wavelength channels. To achieve this capability, the periodic resonance passbands of the CS need to overlap with this comb of x distinct wavelength channels. Generally, the resonance passband of a modulation MRR periodically repeats at the spectral distance known as Free Spectral Range (FSR), and two adjacent wavelengths in the incoming comb of N wavelengths have a channel spacing of Δ (Eq. 12).

$$\Delta = \frac{FSR}{N+1} \tag{12}$$

Therefore, to filter x distinct wavelengths from these N incoming wavelengths separated by Δ, the FSR of the CS needs to be equal to CS_{FSR} (Eq. 13).

$$CS_{FSR} = \frac{N*\Delta}{x} \tag{13}$$

Since N varies for our RAMM and RMAM TPCs across different DRs (Table II), the required FSR for the CSs would also vary across different DRs. The FSR for a CS can be defined by appropriately defining its radius. Therefore, we designed the CSs with desired FSRs required for our RAMM and RMAM TPCs for various DRs by appropriately defining the radius of a primitive MRR. For that, we used the photonics foundry-validated MODE and INTERCONNECT tools from Ansys/Lumerical [48]. Table IV lists the design parameters of our designed CSs. From Table IV, a CS can incur insertion losses. This can impact the achievable N for a TPC. Our scalability analysis in Section III-B accounts for this fact.

D. System Level Implementation

Fig. 9 illustrates the system level implementation of our accelerators. It consists of a global memory for storing CNN parameters, a pre-processing and mapping unit for decomposing the tensors into DIVs/DKVs and mapping them onto the DIV/DKV elements. It has a mesh of tiles connected to routers and this mesh network facilitates parameter communication among tiles. Each tile consists of 4 RMAM/RAMM TPCs interconnected (via H-tree network) with output buffer, activation and pooling units. Due to their analog nature, the RMAM/RAMM TPCs require DACs and ADCs as well. In addition, each tile also contains *psum* reduction network to enable summing up of the intermediate *psum*s. Depending on the type of convolution operations being processed (SC/DC/PC) and their tensor sizes (Table III), the mapping unit sends control signals to the individual RAMM/RMAM TPCs to reconfigure their operational modes (Mode 1/Mode 2).

TABLE IV: Design parameters of various comb switch (CS) designs used in our RMAM and RAMM TPCs for various DRs.

Data Rate (DR) (GS/s)	**1**	**3**	**5**
RAMM TPC			
N	31	20	16
CS_{FSR}	4.83nm	5 nm	NA
Radius	18.17 μm	17.5 μm	NA
No of CS Pairs	3	2	0
Insertion Loss (dB)	0.029	0.028	0
RMAM TPC			
N	**43**	**28**	**22**
CS_{FSR}	4.65 nm	5.35nm	4.54 nm
Radius	18.98 μm	16.2 μm	19.49 μm
No of CS Pairs	4	3	2
Insertion Loss (dB)	0.029	0.026	0.031

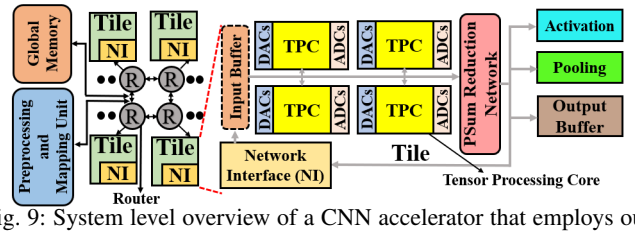

Fig. 9: System level overview of a CNN accelerator that employs our RMAM/RAMM TPCs.

VI. Evaluation

A. Simulation Setup

To evaluate our designed RAMM and RMAM accelerator architectures, we simulated the inference of various depthwise separable convolutions based CNNs such as EfficientNetB7 [21], Xception [18], NASNetMobile [49], and ShuffleNetV2 [22] with input batch size of 1. These CNNs comprehensively cover wide variations seen in CNNs in terms of tensor sizes. We developed a custom, transaction-level, cycle-true python-based simulator to model CNN inference on MRR-based TPC accelerators with weight-stationary dataflow.

TABLE V: ADC area and power overheads.

ADC	**Area (mm^2)**	**Power**
1 GS/s [50]	0.002	2.55 mw
3 GS/s [51]	0.021	11mw
5 GS/s [52]	0.103	29 mw

We compared our RAMM and RMAM accelerator architectures with the baseline AMM (DEAPCNN [15]), MAM (HOLYLIGHT [9]) and the latest variant of AMM design (CROSSLIGHT [11]). We evaluate these accelerators at 4-bit precision and across different DRs such as 1 GS/s, 3 GS/s, and 5 GS/s. The N values corresponding to different DRs are taken from Table II for our system level analysis. Table V and

Table VI give the parameters used for evaluating the overheads of the peripherals. We consider each laser diode to emit input optical power of 10 mW (10 dBm) (Table I) [15]. Multiplexer and splitter parameters are taken from [9], and other VDP element parameters are listed in Table VII.

We performed area proportionate (AP) analysis, for which, we altered the VDPE count of each accelerator so that the accelerators' area matched with the area of the RMAM accelerator with VDPE count of 512. Table VIII reports our obtained area-proportionate VDPE counts for all our considered accelerators. We evaluate the metrics such as Frames Per Second (FPS) and FPS/W (energy efficiency) for our considered accelerator architectures.

TABLE VI: Accelerator Peripherals Parameters [9].

	Power(mW)	Area(mm^2)	Latency
DAC [53]	30	0.034	0.78ns
Reduction Network	0.05	0.03E-3	3.125ns
Activation Unit	0.52	0.6E-3	0.78ns
IO Interface	140.18	24.4E-3	0.78ns
Pooling Unit	0.4	0.24E-3	3.125ns
eDRAM	41.1	166E-3	1.56ns
Bus	7	9E-3	5 cycles
Router	42	0.151	2 cycles

TABLE VII: VDP Element Parameters [11].

DKV/DIV MRR Q-factor	8000	
DKV/DIV MRR FWHM	0.2 nm	
Sensitivity of PD	-20 dBm	
	Power (mW)	**Latency**
EO Tuning	80 μW/FSR	20 ns
TO Tuning	27.5 mW/FSR	4 μs
TIA	7.2 mW	0.15 μs
Photodetector	2.8 mW	5.8 ps

TABLE VIII: VDPE counts of various accelerators.

Accelerators	**DR (GS/s)**		
	1	**3**	**5**
RMAM	512	512	512
RAMM	587	576	567
MAM (HOLYLIGHT [9])	568	562	547
AMM (DEAPCNN [15])	656	629	620

B. Evaluation Results

Fig.10 shows the FPS results for various accelerators at different DRs, normalized to RMAM at 1 GS/s. Our RMAM accelerator on gmean outperforms MAM (HOLYLIGHT), AMM (DEAPCNN), and CROSSLIGHT for all DRs, such as 1 GS/s, 3 GS/s, and 5 GS/s. At 1 GS/s, RMAM achieves 1.8×, 17.1×, and 65× better FPS than MAM (HOLYLIGHT), AMM (DEAPCNN), and CROSSLIGHT, respectively, on gmean across the CNNs. From Section III-B, the *N* values decrease with increase in DR, which leads to lower throughput with increase in DR for various accelerators. For instance, RMAM's FPS drops by 5.3× and 8× at 3 GS/s and 5 GS/s respectively, compared to its FPS at 1 GS/s. Therefore, compared to the 3-GS/s (5-GS/s) variants of MAM (HOLYLIGHT), AMM (DEAPCNN), and CROSSLIGHT, respectively, our 1-GS/s RMAM variant achieves 8.3× (10.2×), 52.57× (79.8×), and 86× (106×) better FPS. These FPS benefits are because of our RMAM variants' higher throughput and more efficient processing through reconfiguration (Mode 2 operation). Our other accelerator RAMM, at 1 GS/s, achieves 1.54× and 5.8× better FPS compared to AMM (DEAPCNN) and CROSSLIGHT, respectively. Even at higher DRs, RAMM performs better than AMM (DEAPCNN) and CROSSLIGHT. However, at 5 GS/s, because of low *N*, reconfiguration is not supported in RAMM due to the condition $y = N \geq 2x?floor(N/9) : 0$. Therefore RAMM is exactly identical to AMM (DEAPCNN) at 5 GS/s. Overall, our RMAM accelerator gives better FPS compared to other accelerators across all DRs.

Fig. 11 shows the FPS/W (energy efficiency) results for various accelerators across different DRs, normalized to RMAM at 1 GS/s. Our RMAM and RAMM accelerators also achieve better FPS/W compared to the other accelerators. At 1 GS/s, RMAM achieves 1.5×, 27.2×, and 171× better FPS/W than MAM (HOLYLIGHT), AMM (DEAPCNN), and CROSSLIGHT, respectively, on gmean across various CNNs. Similar to the FPS results, when compared to the 3-GS/s (5-GS/s) variants of MAM (HOLYLIGHT), AMM (DEAPCNN), and CROSSLIGHT, our 1-GS/s RMAM variant achieves 4.2× (4×), 46.4× (29.6×), and 80.7× (54×) better FPS/W, respectively. These energy efficiency benefits are also because of the improved VDPE utilization achieved from the superior reconfigurability of our RMAM accelerator. The improved VDPE utilization better amortizes the static power consumption of the constituent hardware components (e.g., MRRs, CSs, ADCs, DACs) to yield better energy efficiency. Our other accelerator RAMM, at 1 GS/s, achieves 1.5× and 9.7× better FPS/W compared to AMM (DEAPCNN) and CROSSLIGHT, respectively. Even at higher DRs, RAMM performs better than AMM (DEAPCNN) and CROSSLIGHT. However, at 5 GS/s, RAMM and AMM (DEAPCNN) have equal energy efficiency. Overall, our RMAM provides better energy efficiency compared to the other accelerators across different DRs.

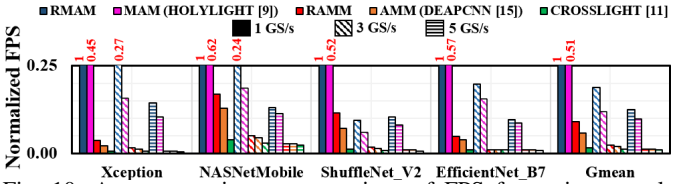

Fig. 10: Area proportionate comparison of FPS for various accelerators across different CNNs and data rates (DRs). Results are normalized with respect to RMAM at 1 GS/s.

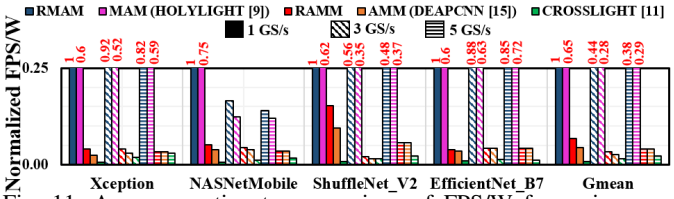

Fig. 11: Area proportionate comparison of FPS/W for various accelerators across different CNNs and data rates (DRs). Results are normalized with respect to RMAM at 1 GS/s.

VII. Conclusion

In this paper, we presented the use of our invented reconfigurable VDPEs as a novel way of introducing flexibility in the photonic MRR-based CNN accelerators. Our reconfigurable VDPEs employ a set of comb switches to enable dynamic maximization of the size compatibility between the VDPEs and the CNN tensors that are processed using the VDPEs. We

then used our reconfigurable VDPEs to enhance the MRR-based CNN accelerators of the AMM and MAM categories. Consequently, we derived different variants of Reconfigurable MAM (RMAM) and Reconfigurable AMM (RAMM) accelerators that operate at different data rates. We evaluated different variants of our RMAM and RAMM accelerators against three prior works, for the inference of four modern CNNs with mixed-sized tensors. Our evaluation indicates that our RMAM and RAMM accelerators are significantly better at striking a balance between the hardware utilization and CNN processing latency, which in turn provides them with substantial improvements in FPS and FPS/W, with equal area consumption, compared to the photonic MRR-based accelerators from prior works. These results promote the use of our RMAM and RAMM accelerators for efficient processing of future CNNs having a wide variety in their employed tensor sizes.

ACKNOWLEDGMENTS

We thank the anonymous reviewers whose valuable feedback helped us improve this paper. We would also like to acknowledge the National Science Foundation (NSF) as this research was supported by NSF under grant CNS-2139167.

REFERENCES

[1] Yann *et al.*, "Deep learning," *Nature*, May 2015.
[2] M. Popel *et al.*, "Transforming machine translation: a deep learning system reaches news translation quality comparable to human professionals," *Nature Communications*, Sep. 2020.
[3] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, Feb. 2015.
[4] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.
[5] L. Baischer *et al.*, "Learning on hardware: A tutorial on neural network accelerators and co-processors," *CoRR*, vol. abs/2104.09252, 2021.
[6] P. R. Prucnal *et al.*, *Neuromorphic Photonics*. CRC Press, May 2017.
[7] A. N. Tait *et al.*, "Broadcast and weight: An integrated network for scalable photonic spike processing," *J. Lightwave Technol.*, Nov 2014.
[8] Tait *et al.*, "Microring weight banks," *IJSTQE*, vol. 22, no. 6, pp. 312–325, 2016.
[9] W. Liu *et al.*, "Holylight a nanophotonic accelerator for deep learning in data centers," in *2019 DATE*, Mar. 2019.
[10] L. Yang *et al.*, "On-chip optical matrix-vector multiplier," in *Optics and Photonics for Information Processing*. SPIE, Sep. 2013.
[11] F. Sunny *et al.*, "Crosslight: A cross-layer optimized silicon photonic neural network accelerator," in *DAC*, 2021.
[12] Y. Shen *et al.*, "Deep learning with coherent nanophotonic circuits," *Nature Photonics*, Jun. 2017.
[13] C. Ramey, "Silicon photonics for artificial intelligence acceleration : Hotchips 32," *HCS*, 2020.
[14] H. Bagherian *et al.*, "On-chip optical convolutional neural networks," *CoRR*, 2018.
[15] V. Bangari *et al.*, "Digital electronics and analog photonics for convolutional neural networks (deap-cnns)," *JSTQE*, 2019.
[16] P. Y. Ma *et al.*, "Photonic independent component analysis using an on-chip microring weight bank," *Optics Express*, 2020.
[17] A. N. Tait *et al.*, "Neuromorphic photonic networks using silicon photonic weight banks," *Scientific reports*, 2017.
[18] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *CoRR*, 2016.
[19] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, 2017.
[20] S. Dave *et al.*, "Hardware acceleration of sparse and irregular tensor computations of ml models: A survey and insights," *Proceedings of the IEEE*, 2021.
[21] M. Tan *et al.*, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, 2019.
[22] X. Zhang *et al.*, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," *CoRR*, 2017.
[23] M. Sandler *et al.*, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, 2018.
[24] K. He *et al.*, "Deep residual learning for image recognition," 2015.
[25] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, 2014.
[26] Z. Jia *et al.*, "Dissecting the graphcore IPU architecture via microbenchmarking," *CoRR*, 2019.
[27] X. Si *et al.*, "A twin-8t sram computation-in-memory unit-macro for multibit cnn-based ai edge processors," *IJSSC*, 2020.
[28] Y. Yu *et al.*, "Opu: An fpga-based overlay processor for convolutional neural networks," *IVLSI Systems*, 2020.
[29] M. Miscuglio *et al.*, "Photonic tensor cores for machine learning," *Applied Physics Reviews*, sep 2020.
[30] H. Zhang *et al.*, "An optical neural chip for implementing complex-valued neural network," *Nature Communications*, 2021.
[31] S. S. Vatsavai *et al.*, "Silicon photonic microring based chip-scale accelerator for delayed feedback reservoir computing," in *VLSID*, 2021.
[32] K. Shiflett *et al.*, "Pixel: Photonic neural network accelerator," in *HPCA*, 2020.
[33] S. Kyle *et al.*, "Albireo: Energy-efficient acceleration of convolutional neural networks via silicon photonics," in *ISCA*, 2021.
[34] J. Feldmann *et al.*, "Parallel convolutional processing using an integrated photonic tensor core," *Nature*, 2021.
[35] R. Hamerly *et al.*, "Large-scale optical neural networks based on photoelectric multiplication," *Phys. Rev. X*, May 2019.
[36] Z. Zhao *et al.*, "Hardware-software co-design of slimmed optical neural networks," ser. ASPDAC '19, 2019.
[37] X. Xu *et al.*, "11 TOPS photonic convolutional accelerator for optical neural networks," *Nature*, Jan. 2021.
[38] X. Zhao *et al.*, "An integrated optical neural network chip based on mach-zehnder interferometers," 2018.
[39] X. Lin and Others, "All-optical machine learning using diffractive deep neural networks," *Science*, 2018.
[40] T. Zhou *et al.*, "Large-scale neuromorphic optoelectronic computing with a reconfigurable diffractive processing unit," *Nature Photonics*, 2021.
[41] Mourgias-Alexandris *et al.*, "Neuromorphic photonics with coherent linear neurons using dual-iq modulation cells," *JLT*, 2020.
[42] L. De Marinis *et al.*, "Photonic neural networks: A survey," *IEEE Access*, 2019.
[43] M. A. Al-Qadasi *et al.*, "Scaling up silicon photonic-based accelerators: Challenges and opportunities," *APL Photonics*, Feb. 2022.
[44] Q. Cheng *et al.*, "Silicon photonics codesign for deep learning," *Proceedings of the IEEE*, 2020.
[45] H. Kwon *et al.*, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.
[46] F. Chollet *et al.*, "Keras," https://keras.io/api/applications/, 2015.
[47] B. G. Lee *et al.*, "All-optical comb switch for multiwavelength message routing in silicon photonic networks," *IEEE Photonics Technology Letters*, 2008.
[48] INTERCONNECT and MODE, "Accurately simulate photonic components and circuits," Jul 2020. [Online]. Available: https://www.lumerical.com/products/
[49] B. Zoph *et al.*, "Learning transferable architectures for scalable image recognition," *CoRR*, 2017.
[50] D.-R. Oh *et al.*, "An 8b 1gs/s 2.55mw sar-flash adc with complementary dynamic amplifiers," in *IVLSIC*, 2020.
[51] Y.-S. Shu, "A 6b 3gs/s 11mw fully dynamic flash adc in 40nm cmos with reduced number of comparators," in *VLSIC*, 2012.
[52] M. Guo *et al.*, "A 29mw 5gs/s time-interleaved sar adc achieving 48.5db sndr with fully-digital timing-skew calibration based on digital-mixing," in *VLSIC*, 2019.
[53] F. N. U. Juanda *et al.*, "A 10-gs/s 4-bit single-core digital-to-analog converter for cognitive ultrawidebands," *TCS*, 2017.