TT-CIM: Tensor Train Decomposition for Neural Network in RRAM-Based Compute-in-Memory Systems

Fan-Hsuan Meng, Yuting Wu, *Student Member, IEEE*, Zhengya Zhang, *Senior Member, IEEE*, and Wei D. Lu, *Fellow, IEEE*

***Abstract*— Compute-in-Memory (CIM) implemented with Resistive-Random-Access-Memory (RRAM) crossbars is a promising approach for accelerating Convolutional Neural Network (CNN) computations. The growing size in the number of parameters in state-of-the-art CNN models, however, creates challenge for on-chip weight storage for CIM implementations, and CNN compression becomes a crucial topic of exploration. Tensor Train (TT) decomposition can be used to decompose a tensor into smaller ones with fewer parameters, at the cost of increased number of computations. In this work we propose a technique to minimize intermediate operations across the full convolution operation and improve hardware utilization to implement TT-CNNs in CIM systems. We first use an iterative decompose-and-fine-tune method to prepare TT-CNNs. We then propose an inter-convolutional-step reuse scheme to reduce the required operation count and post-mapping RRAM count for TT-CNN implementation in tiled-CIM architecture. We demonstrate that through proper mapping, pipelining, and reuse, effective compression ratio of 12 and 20 with 0.8% and 1.4% accuracy drop, respectively for WRN; and effective compression ratio of 6 and 11 with 0.9% and 1.2% accuracy drop for VGG8. We also show that around 30% higher hardware utilization than the original CNN format can be achieved using the proposed TT-CIM approaches.**

***Index Terms*— Compute-in-memory, deep neural network, convolutional neural network, neural network compression, tensor train decomposition.**

I. Introduction

DEEP neural networks (DNNs), specifically convolutional neural networks (CNNs) has become the most successful machine learning method in recent years [1], [2], [3], [4]. State-of-the-art DNN models are typically large in scale, containing tens or even hundreds of millions of parameters [1], [2], [5], resulting in expensive data movements between the memory units and the computation units in traditional von Neumann architectures. Compute-in-memory (CIM)-based DNN accelerators address this challenge by performing computation directly in the memory units that store the weights [6]. Typical CIM implementations adopt a tiled architecture design with all weights stored stationary across CIM-tiles on the chip to eliminate off-chip weight access cost during inference [11], [13].

However, in order to store all the weights on the chip, the required chip scale will be largely affected by the size of the targeted CNN model. Therefore, various DNN model compression techniques, such as quantization, or pruning, have been applied to CIM architectures to reduce model size [14], [15], [16], [17], [18], [19]. Among these compression methods, Tensor decomposition, such as tensor-train (TT) decomposition [21] has been gaining attention for DNN compression of weights due to its high compression ratio, defined as the ratio between the uncompressed parameter count and the compressed parameter count [20], [25], [26].

CNN inference can be viewed as series of vector-matrix-multiplication (VMM) operations. TT decomposition breaks each matrix into smaller ones, namely, TT-cores. The advantage of TT decomposed CNN (TT-CNN) is that it consists only of regular structured VMM operations with the TT-cores. Therefore, the implementation of TT-NN models could require no fundamental hardware change. However, despite the advantages, implementing TT-CNN in a practical tiled CIM architecture still faces several challenges.

1) *Increased multiply-and-accumulate (MAC) computations*: Performing VMM on a decomposed matrix (TT-VMM) would require more MAC operations than the uncompressed format. Therefore, the area efficiency (throughput/area) will be degraded. In addition, some TT-cores require significantly more MAC operations than others, resulting in the lowering of **effective compression ratio** after pipeline balancing by storing more copies of these slower cores.
2) *Complicated pipeline design*: TT decomposition introduces more steps of VMMs and producing more intermediate products (IPs) which need to be carefully buffered and consumed in CIM systems. How to efficiently pipeline the intermediate results without

Manuscript received 4 September 2023; revised 20 October 2023 and 3 December 2023; accepted 12 December 2023. Date of publication 29 December 2023; date of current version 28 February 2024. This work was supported in part by the National Science Foundation (NSF) under Award CCF-1900675 and in part by Semiconductor Research Corporation (SRC) and Defense Advanced Research Projects Agency (DARPA) through the Applications Driving Architectures (ADA) Research Center. This article was recommended by Associate Editor C. Wang. *(Corresponding author: Wei D. Lu.)*

The authors are with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: fanhsuan@umich.edu; zhengya@umich.edu; wluee@umich.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2023.3344550.

Digital Object Identifier 10.1109/TCSI.2023.3344550

increasing the overall latency and SRAM buffer size require careful consideration.

3) *Difficulty of efficient parameter mapping*: The TT-cores decomposed from the original matrix can vary significantly in size and shape. Some cores are small with very high aspect ratios while others have very low aspect ratios, making it challenging to map TT-CNN models with high hardware utilization on regular tiled CIM systems.

State-of-the-art CNNs are largely composed of convolutional (CONV) layers, therefore, in this work, we focus mostly on optimizing implementation of TT decomposed CONV (TT-CONV) layers on traditional fully weight stationary tiled CIM hardware. We address the challenges mentioned previously and summarize our contribution as follows.

1) *Inter-CONV-step reuse*: Previous work viewed steps in a CONV operation as independent VMM operations and minimize the required computation only within a TT-VMM. In this work, we consider possible IP reuse opportunities between CONV steps, discussed in section III-A After adopting the inter-CONV-step reuse, the required number of MAC could be even lower than the uncompressed CNN. It also results in higher effective compression ratio than no reuse TT-CNN by speeding up slow TT-cores with the reuse.
2) *Pipeline design for TT-CNN in tiled CIM*: We demonstrated pipeline balancing, scheduling and IP buffering design of the TT-CNN with inter-CONV-step reuse, in section III-B and III-C We show that despite the increased computational stages in TT-CNN, the latency and required SRAM buffer size between TT-cores are comparable with the hardware or uncompressed CNN.
3) *Partial crossbar read for mapping utilization*: We proposed two partial read schemes to map smaller matrices with either very high or low aspect ratios to increase weight memory (i.e., RRAM crossbar) utilization, resulting in a higher effective compression ratio in section V-A.1.

The organization of the paper is as follows: In section II, we discuss the background including compute-in-memory (CIM) and tensor train decomposition for CNNs. Our proposed inter-convolutional-step reuse scheme for CIM hardware is presented in section III. The experiment setup and the results are shown in sections IV and V, and further discussions and conclusions are presented in sections VI and VII, respectively.

II. Background

A. Compute-in-Memory

Compute-in-memory is a promising approach to compute VMM directly in the memory unit storing the weight matrix [6], [7], [8], [9]. CIM computes VMM in a fully parallel fashion to provide high throughput and high energy efficiency. RRAM is especially attractive for CIM due to its nonvolatility, analog storage, compact size and low read energy. An RRAM-CIM mapping example is shown in Fig. 1(a): the positive and negative values are encoded by different conductance levels and are mapped separately on the crossbar. To perform a VMM, an input vector, encoded as input pulse trains, is applied to the rows of the memory array. The product between an input (pulses) and a weight value (conductance) results in a current that is accumulated on the bitlines (BLs), representing the VMM output.

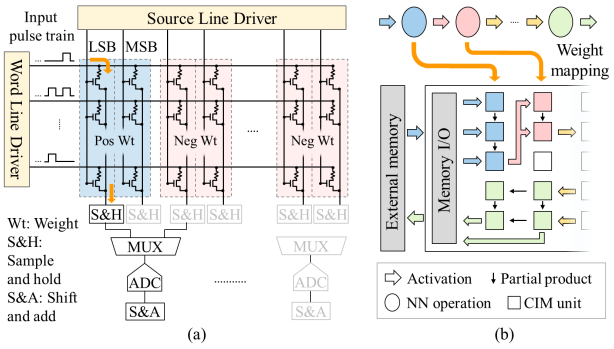


Fig. 1. (a) Basic principle of CIM VMM with positive and negative values encoded by different conductance levels and mapped separately on the crossbar. Input vector of a VMM is applied to the wordlines (WLs) of the crossbar as input pulse trains, with current accumulated on the bitlines as the VMM outputs. (b) Illustration of the spatial mapping of DNN on tiled CIM architecture.

RRAM-based CIM hardware usually follows a tiled architecture [9], [13], as illustrated in Fig. 1(b), with memory tiles storing the weights of the DNN. Weights are kept stationary on the memory tiles and the input activations are streamed in and through the tiles, producing output activations to feed the downstream tiles. Since all the weights are stored on chip and the activations are produced and consumed in a pipelined fashion between the tiles, external memory access for weights and activations are completely eliminated. Despite the performance and efficiency advantage of this entirely weight stationary approach, the on-chip memory size limits the maximum DNN size that can be supported.

Tensor decomposition methods, such as TT, allows large models to fit in smaller memories and retains regular data structure so that CIM can still be effectively applied. Therefore, tensor decomposition is particularly relevant to CIM hardware as a promising approach to allow CIM hardware to run large DNNs while achieving a high efficiency.

B. TT Decomposition

To represent arrays of different dimensionality, we follow the annotation in [20] and represent one-dimensional (1D) arrays, *vectors*, with bold lower case letters (e.g. $\boldsymbol{a}$); two-dimensional (2D) arrays, *matrices*, with bold upper case letters (e.g. $\boldsymbol{A}$); and d-dimensional ($d > 2$) arrays, *tensors*, with calligraphic bold upper case letters (e.g. $\boldsymbol{\mathcal{A}}$). An element in a vector, a matrix and a tensor are denoted as ordinary lower case letters (e.g. $a(i)$), ordinary upper case letters (e.g. $A(i, j)$), and ordinary calligraphic upper case letters (e.g. $\mathcal{A}(i_1, ..., i_d)$), respectively.

In TT decomposition, a d-dimensional tensor, $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{n_1 \times n_2 \times ... \times n_d}$ (where $n_1, n_2, \ldots, n_d$ represent the size of each dimension) is decomposed to a series of d tensor cores, or TT-cores, $\boldsymbol{\mathcal{G}_k} \in \mathbb{R}^{r_{k-1} \times n_k \times r_k}$, $k \in [1, d]$, where r_k is the compression rank or TT-rank. Generally, larger r_k leads to lower compression ratio and lower approximation error, and

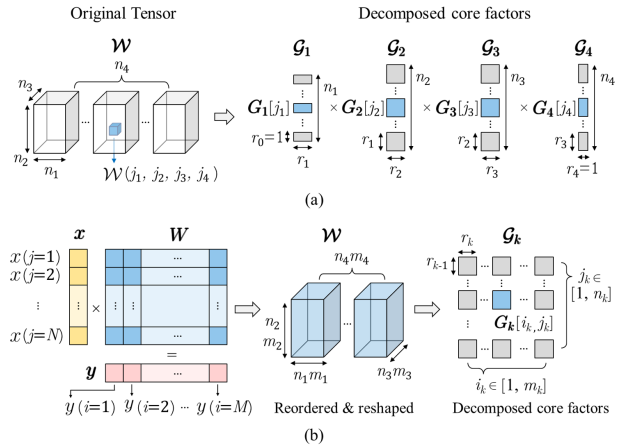


Fig. 2. (a) A 4-dimensional tensor approximated with four TT-cores using tensor train decomposition. Each rectangular represents a matrix G_i. (b) Reshape, reorder and TT-decomposition of a matrix for TT-VMM.

vice versa for lower r_k. Each 3D tensor core $\boldsymbol{\mathcal{G}_k} \in \mathbb{R}^{r_{k-1} \times n_k \times r_k}$ can be viewed as a stack of n_k number of $r_{k-1} \times r_k$ matrices. These matrices are called factors and are denoted $\boldsymbol{G_k}[i]$, $i \in [1, n_k]$. To satisfy the boundary condition, $r_0 = r_d = 1$ must be imposed.

A TT decomposition of a d-dimensional array is referred to as a rank-d TT decomposition. Take a rank-d TT decomposition of tensor $\boldsymbol{\mathcal{W}}$ as an example, after applying TT decomposition, $\boldsymbol{\mathcal{W}}$ is decomposed to 4 tensor cores, $\boldsymbol{\mathcal{G}_1}, \boldsymbol{\mathcal{G}_2}, \boldsymbol{\mathcal{G}_3}$, and $\boldsymbol{\mathcal{G}_4}$. An element in the original tensor, $\mathcal{W}(j_1, j_2, j_3, j_4)$, where $j_1 \in [1, n_1], \ldots, j_d \in [1, n_d]$, can be reconstructed by the following equation:

$$\mathcal{W}(j_1, j_2, j_3, j_4) \approx \boldsymbol{G_1}[j_1] \times \boldsymbol{G_2}[j_2] \times \boldsymbol{G_3}[j_3] \times \boldsymbol{G_4}[j_4]. \tag{1}$$

An example of the TT decomposition and the reconstruction of a tensor element is illustrated in Fig. 2(a). An element of the original tensor is reconstructed by the product of 4 factors, one from each tensor core.

C. TT-VMM

When VMM is to be computed with a TT-decomposed matrix, it is more computational effective to perform the VMM directly on the compressed format instead of reconstructing the matrix back to its original form, then perform VMM. In order to facilitate TT-VMM, the matrix, $\boldsymbol{W} \in \mathbb{R}^{N \times M}$ should be reshaped and decomposed with both M and N factorized into d parameters. Consider a VMM:

$$\boldsymbol{y} = \boldsymbol{x} \times \boldsymbol{W}. \tag{2}$$

We refer to vector $\boldsymbol{x}$ as the input vector and N as the input dimension; vector $\boldsymbol{y}$ as the output vector and M as the output dimension.

Before TT-decomposition, the original matrix $\boldsymbol{W}$ is reshaped and reordered into $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{n_1 m_1 \times n_2 m_2 \times \ldots \times n_d m_d}$ ($n_1 \times n_2 \times \ldots \times n_d = N$, $m_1 \times m_2 \times \ldots \times m_d = M$), and each value in the reshaped tensor is represented as $\boldsymbol{\mathcal{W}}([i_1, j_1], [i_2, j_2], \ldots, [i_d, j_d])$. After applying a rank-$d$ TT-decomposition, the tensor can be decomposed into d 4D cores $\boldsymbol{\mathcal{G}_k} \in \mathbb{R}^{r_{k-1} \times m_k \times n_k \times r_k}, k = 1, \ldots, d$. Each core can also be viewed as a 2D array of core factors, $\boldsymbol{G}_k[i_k, j_k]$, where i_k and j_k correlates to the relative position in the input dimension and the output dimension respectively. The TT-VMM of (2) on the decomposed matrix will then be:

$$\mathcal{Y}(i_1, i_2, \ldots, i_d) \approx \sum_{j_1, \ldots, j_d} \mathcal{X}(j_1, j_2, \ldots, j_d) \times \boldsymbol{G}_1[i_1, j_1] \times \boldsymbol{G}_2[i_2, j_2] \times \ldots \times \boldsymbol{G}_d[i_d, j_d], \tag{3}$$

where $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_d}$ as reshaped $\boldsymbol{y}$ vector; and $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{m_1 \times m_2 \times \ldots \times m_d}$ as reshaped $\boldsymbol{x}$ vector. A visualization of the process is shown in Fig. 2(b).

To illustrate the computation of TT-VMM, we use a rank-4 TT-VMM example as shown in Fig. 3. We use a series of four TT-cores, with one blue factor in each TT-core to represent one matrix value, as shown in Fig. 3. For example, Fig. 3(a-i) represents $\mathcal{W}([1, 1], [1, 1], [1, 1], [1, 1])$ and Fig. 3(c-iii) represents $\mathcal{W}([m_1, n_1], [m_2, n_2], [m_3, n_3], [m_4, n_4])$. The yellow blocks and the red blocks represent values in input vector $\boldsymbol{x}$ and output vector $\boldsymbol{y}$, respectively. To compute value $y(i_1, i_2, i_3, i_4)$, all the multiplication result between $\boldsymbol{x}$ and $\mathcal{W}([i_1, j_1], [i_2, j_2], [i_3, j_3], [i_4, j_4])$, $j_1 \in [1, n_1]$, $j_2 \in [1, n_2]$, $j_3 \in [1, n_3]$, $j_4 \in [1, n_4]$, should be accumulated. For example, $y(i = 1)$ in Fig. 3 equals (a-i) + (b-i) + ... + (c-i). By analyzing the full VMM computation, it can be observed that there are repetitive computations that could be eliminated. Two main ways could be used to reduce the computation count, first is **reuse** of intermediate products (IPs) between TT-cores, second is **contraction** of IPs between TT-cores.

1) TT-VMM With Reuse: There are duplicate computations in the TT-VMM, for example, for value Fig. 3(a-i) and (a-ii), the computations are:

$$\begin{aligned} &W(i = 1, j = 1) \times x(1) \\ &\quad = \boldsymbol{G_1}[1, 1] \times \boldsymbol{G_2}[1, 1] \\ &\qquad \times \boldsymbol{G_3}[1, 1] \times \boldsymbol{G_4}[1, 1] \times x(j = 1) \end{aligned} \tag{4}$$

and

$$\begin{aligned} &W(i = 2, j = 1) \times x(1) \\ &\quad = \boldsymbol{G_1}[2, 1] \times \boldsymbol{G_2}[1, 1] \\ &\qquad \times \boldsymbol{G_3}[1, 1] \times \boldsymbol{G_4}[1, 1] \times x(j = 1). \end{aligned} \tag{5}$$

It can be observed, the IP of $\boldsymbol{G_2}[1, 1] \times \boldsymbol{G_3}[1, 1] \times \boldsymbol{G_4}[1, 1] \times x(j = 1)$ can be reused in both computations. This type of reuse can be achieved across different input and output dimensions and has been demonstrated in [25].

2) TT-VMM With Contraction: Another type of duplicate computation is like the one between Fig. 3(a-i) and (b-i), the two computations can be expressed as:

$$\begin{aligned} &x(j = 1) \times W(i = 1, j = 1) \\ &\quad = x(j = 1) \times \boldsymbol{G_1}[1, 1] \\ &\qquad \times \boldsymbol{G_2}[1, 1] \times \boldsymbol{G_3}[1, 1] \times \boldsymbol{G_4}[1, 1] \end{aligned} \tag{6}$$

and

$$\begin{aligned} &x(j = 2) \times W(i = 1, j = 1) \\ &\quad = x(j = 2) \times \boldsymbol{G_1}[1, 2] \\ &\qquad \times \boldsymbol{G_2}[1, 1] \times \boldsymbol{G_3}[1, 1] \times \boldsymbol{G_4}[1, 1]. \end{aligned} \tag{7}$$

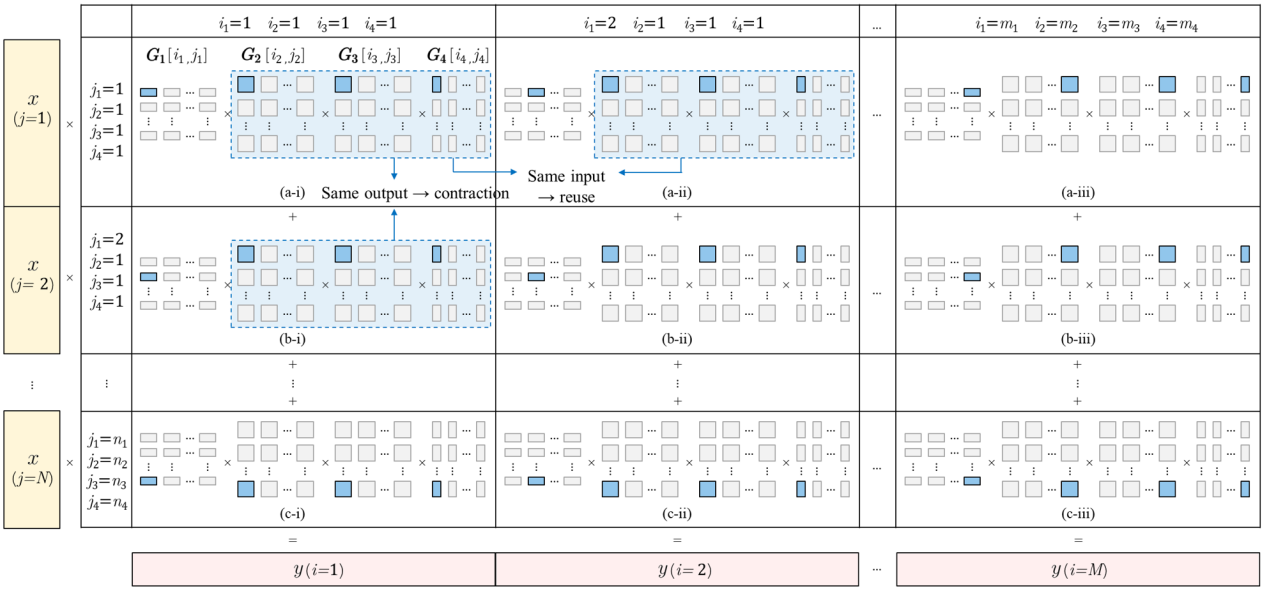


Fig. 3. Illustration of computations included in a TT-VMM. Yellow and red blocks represent the input and output vectors respectively. Blue blocks represent the factors used to reconstruct a specific weight value.

It can be observed that the computations after the first TT-core are the same, and since these two values will both contribute to $y(j=1)$, the IPs after the first TT-core could be accumulated immediately and duplicated computation could be avoided.

The computation can then be viewed as a **series of d-stage VMMs**, one for each TT-core. To illustrate this process, take a rank-3 TT-VMM in Fig. 4(a) as example, the process consists of 3 VMM stages, at each stage, the matrix is a large matrix combined from all the factors lined up in the order of $m_k \times n_k$, denoted as $\boldsymbol{G_{cbk}} \in \mathbb{R}^{r_k m_k \times r_{k-1} n_k}, k = 1, 2, 3$. A factor in the TT-core will then be a slice of $\boldsymbol{G_{cbk}}$ as follows:

$$\boldsymbol{G_{cbk}}[i_k : i_k + r_k][j_k : j_k + r_{k-1}] = \boldsymbol{G_k}[i_k, j_k], \tag{8}$$

In the first VMM stage, input vector will be $\boldsymbol{x}[j_3][j_2][1 : n_1]$, and it will take $n_2 \times n_3$ VMMs to finish computing the entire input space, producing the intermediate product of $\boldsymbol{\mathcal{V}_1} \in \mathbb{R}^{m_1 \times n_2 \times n_3}$. It can be observed that in the process, the output of the VMM contract in n_1 and expand in m_1.

A similar process continues in the next stage of operation corresponding to TT-core 2, where the IP $\boldsymbol{\mathcal{V}_1}$ is reshaped into $n_2 \times m_1 \times n_3$ to align with the input dimension n_2 of the second VMM stage, and it will take $m_1 \times n_3$ VMMs in total to process the full input IP. In this stage, the IP contract in n_2 and expand m_2, therefore the output IP will be $\boldsymbol{\mathcal{V}_2} \in \mathbb{R}^{m_1 \times m_2 \times n_3}$. Same process applies to the last stage for TT-core 3 and the end result will be the final output of the TT-VMM with the size of size $M = m_1 \times m_2 \times m_3$. This type of computation is inherently a better fit for CIM hardware because the contraction combines cores into larger sized matrix and will result in better crossbar mapping utilization. The same principle has been demonstrated in [26] with SRAM.

D. TT-Decomposition for CNNs

CNNs are largely composed of CONV and FC layers. Inference over a FC layer is effectively a vector-matrix-multiplication of size $N \times M$ (i.e., with matrix of N rows and M columns), where N being the size of the input activation and M being the size of the output activation. The matrix is the trainable weights of the FC layer. Prior work has successfully demonstrated compressing FC layers in DNNs [20].

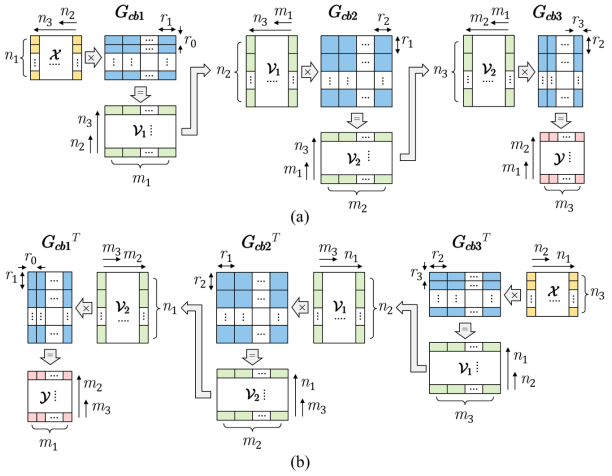


Fig. 4. Contraction process of VMM operated on a rank-3 TT-decomposed tensor in (a) forward and (b) backward computational direction.

The inference over a CONV layer can be viewed as a **series of VMMs** as the filter sweeps across the full feature map. We refer to each of these VMM as a **CONV step** in following discussions. Consider a convolutional weight of kernel size $l \times l$, with C number of channel and S number of filters. Each VMM has input dimension $N = l^2 C$ and output dimension $M = S$. The input vector is segmented and reshaped from the full input feature map, and the matrix is the trainable weights of the CONV layer. A CNN with TT-decomposed weights is referred to as a TT-CNN. Prior work has demonstrated rank-4 TT-decomposition where the input dimension is factorized as $n_1 \times n_2 \times n_3 \times n_4, n_1 = l^2, C = n_2 n_3 n_4$ can achieve high compression ratio at low accuracy drop for CNN compression [22], [23], [24].

III. Proposed TT-CIM Scheme in Tiled CIM Structures

Prior works on TT-CNN implementation only optimize the computation of TT-CONVs by identifying reuse opportunities within one CONV step. In this section, we proposed a computation scheme for contractional TT-VMM which includes **inter-CONV-step** IP reuse to achieve complete reuse in the scope of the full CONV operation. We then demonstrate the implementation of such computation in traditional tiled CIM architectures, mentioned in II-A without specific hardware changes.

A. Contractional TT-CONV With Inter-CONV-Step IP Reuse

There are two ways to compute contractional TT-VMM, forward and backward. The forward computation is discussed previously in Fig. 4(a); however, due to the property of multiplication of transposed matrices:

$$x \times \boldsymbol{A} \times \boldsymbol{B} = x \times \boldsymbol{B}^T \times \boldsymbol{A}^T, \tag{9}$$

the computation could also be computed backward, as shown in Fig. 4(b), where all the $\boldsymbol{G_{cb}}$s are transposed and the input and output dimension order is reversed. Different computation directions would affect the total number of computations required to finish the full TT-VMM, in addition, it could also affect reuse opportunities between each TT-VMM in the full TT-CONV operation.

By choosing the computational direction with contractional TT-VMM that results in **kernel-space-last** computational sequence, i.e., the TT-cores that are computed later are factorized in the filter kernel row and column dimension, opportunities to reuse IPs across different CONV steps exists. Usually, kernel dimensions (row and column) are factorized in TT-core 3 or 4 [22], [23], [24]. Therefore, to achieve kernel-space-last computational, Contractional TT-VMM should be in backward sequence. The reuse opportunities are explained in an example shown below.

Consider a TT-CONV example of input feature map (IFM) and output feature map (OFM) size of 5×5 , filter kernel size of 3×3 and input depth of 2, with filters factorized in the input dimension with the order of filter row, filter column and then filter depth (corresponding to TT-core 1 to 3) for the TT decomposition. Fig. 5(a) to (d) shows parts of the computations involved in four different VMMs in the full TT-CONV operation, contributing to $OFM(2, 1, 1)$, $(3, 1, 1)$, $(2, 2, 1)$ and $OFM(3, 2, 1)$ respectively. Blue blocks indicate the weight value in the filter and the corresponding factors for said weight; light yellow blocks represent values in the input feature map which will contribute to one output value; dark yellow blocks represent the actual input used for the CONV step. Each multiplication sequence illustrates the computation of one filter value multiplied by one IFM value. For example, in Fig. 5(a), filter value $F(2, 2, 1)$ is multiples with $IFM(2, 1, 1)$ and is a part of $OFM(2, 1, 1)$. Previous computation reduction only exists within each VMM step, while the inter-step IP reuse for contractional TT-VMM can be achieved between CONV-steps (contributing in different outputs) when they use the same input pixel. The reuse

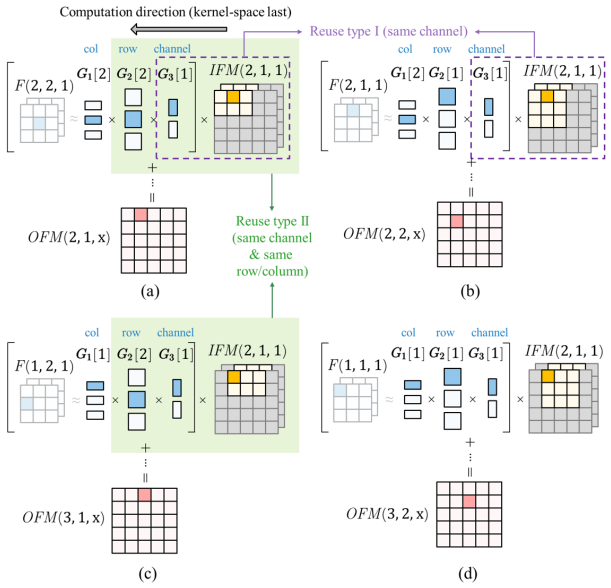


Fig. 5. Example steps of a TT-CONV. Step (a)-(d) all share the same sets of input and contribute to different output values.

opportunity exist before the IP accumulated in the kernel dimension with a mismatch. We would use the example in Fig. 5 to further illustrate this point.

1) Reuse Type I - Reuse of Channel Direction Contraction: Compared to the operation in Fig. 5(a) and (b), the two CONV steps result in different outputs but use the same inputs. Although the two illustrated operations use different weights, their TT operations are the same up till the TT-cores with kernel space factorization. Same with Fig. 5(c) and (d). Therefore, the IPs between TT-core 2 and TT-core 3 can be reused between TT-CONV steps.

2) Reuse Type II - Reuse of Channel & Row/Col Direction Contraction: Another type of reuse opportunity is between (a) & (c), and (b) & (d); by applying the same concept, IPs between TT-core 2 and 1 can be reused. Compared to reuse type I, the computation of an extra core with row factorization can be included before the IPs are reused. This is because, in the two operations, the weight value used is of the same row. The core reused could also be the column core if the factorization order is reversed.

In conclusion, because of the dataflow hardware architecture required for fully weight stationary CIM, TT-VMM using tensor contraction must be implemented. However, when computing TT-CONV, an extra inter-CONV-step reuse can be implemented to possibly reduce the number of operations required to below the original VMM. This reuse is fundamentally similar to the reuse implemented in TIE [25] but was added on top of contractional-TT-VMM and only applied for IPs sharing the same input values in a convolutional operation.

B. TT-CNN TT-Core Throughput Balancing in Tiled CIM

When mapping a traditional CNN to RRAM-based CIM, balancing the layers is very important. The feature map sizes for deeper layers are often smaller than the shallow layers due to downsampling. For example, consider a popular model, Wide ResNet (WRN) 16×8 [28] trained on CIFAR-100 [29]. The input size of the dataset is 32×32 . It goes through three

downsampling, and the feature map shrinks from 32 to 16, to 8, and finally to 1 after a global pooling layer, before the final fully connected layer. So, the number of VMM cycles required to process one input image will be $32 \times 32, 16 \times 16, 8 \times 8$. The pipeline can be balanced by storing copies of weights to speed up the computation in the earlier layers. The goal of a pipeline design is to find the trade-off between hardware utilization and hardware size. If enough copies are stored to achieve maximum hardware utilization, the overall memory size would be overwhelmingly large; on the other hand, if not enough copies are stored for slower layers, the hardware utilization will be low for the faster layers, as they wait for previous layers to finish their computations. As shown in Fig. 6(a), a reasonable approach is to match all stages to the throughput of a medium-sized layer so that only some deep layers are underutilized. We refer to this as a mapping sweet spot.

In TT-CNN, pipeline balancing becomes more complicated because four factors contribute to the number of VMMs each core needs to finish to process one input frame of the NN model, factorization scheme, input feature map size, stride of the CONV, and the level of IP reuse in each TT-core. Consider a rank-4 input factorization scheme where the kernel row and column dimensions are factorized in TT-core 3 and 4 respectively.

The first TT-core has $n_2n_3n_4$ VMMs for each CONV step. Since no kernel row or column factorization has been included, the IPs of TT-core 1 can be reused l^2 times. Therefore, the number of VMMs on $\boldsymbol{G_{cb}}$ required to finish one input frame will be,

$$\frac{W \times H \times n_2n_3n_4}{l^2}, \tag{10}$$

where W and H are the input feature map width and height.

For TT-core 2, the IP has contracted in n_1 and expanded in m_1, thus the number of VMMs for each CONV step will be $m_1n_3n_4$. Since no kernel row or column factorization has been included in neither TT-core 1 nor 2, the IPs of TT-core 2 can be reused l^2 times as well. The total number of VMMs required to finish one input frame will be,

$$\frac{W \times H \times m_1n_3n_4}{l^2} \tag{11}$$

In TT-core 3, the row dimension in the kernel space is factorized but not the column dimension, therefore the IPs could only be reused l times. In addition, striding in the row dimension should be taken into account, therefore, the number of VMMs required will be,

$$\frac{W \times H \times m_1m_2n_4}{l \times stride} \tag{12}$$

The last TT-core has no reuse opportunity available, thus the total VMM count will be,

$$\frac{W \times H \times m_1m_2m_3}{stride} \tag{13}$$

When implementing TT-CNN on RRAM-based CIM, all the cores across all layers need to have their respective number of copies to achieve better performance. An example of a TT-decomposed WRN is shown in Fig. 6(b). As shown, most TT-cores are balanced to the same computation throughput except for a few very fast TT-cores.

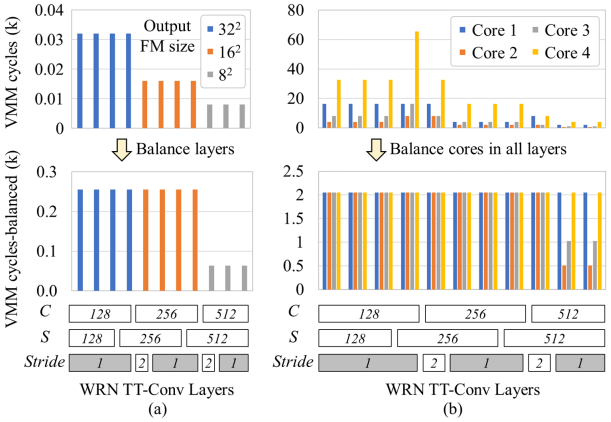


Fig. 6. Pipeline stage balancing for (a) WRN layers and (b) TT-WRN cores.

C. TT-Core Pipelining With Inter-CONV-Step Reuse

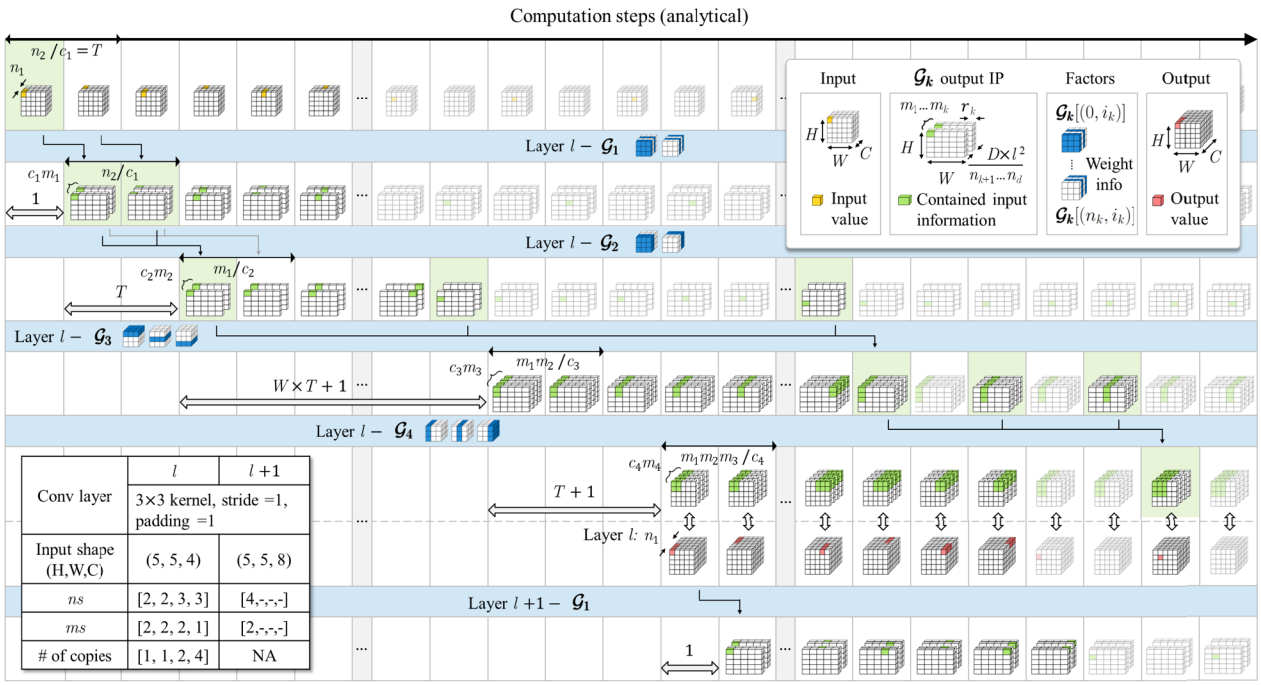
In traditional CNN, in the case of a $l \times l$ padded CONV layer with a row-by-row computational order, a layer can start as soon as there are $(l-2)W + (l-1)$ pixels (full IFM depth) available [11]. Consider all the TT-cores are balanced to produce one pixel (all channels included) in the output feature map every T_{org} cycles, then for 3×3 convolutional operations, the layer-to-layer latency will be:

$$(W+2) \times T_{org}. \tag{14}$$

Compared to traditional CNN, core-to-core pipelining in TT-CNN is more complicated. To design the proper pipelining process, it is important to understate what information each of the IPs carries in each of the corresponding TT-CONV stages. In every TT-core, the IP is contracted in one input dimension and expanded in one output dimension, therefore, stage-by-stage, the IP keeps accumulating information of the input activation until it has all the input dimension accumulated and becomes the final output activation. To further illustrate this process let's use an example in Fig. 7. Information on the CONV layers is provided in the inset table in Fig. 7. For a kernel-space-last computation order, the complete tensor space of the IPs will be $\boldsymbol{\mathcal{V}_1} \in \mathbb{R}^{W \times H \times n_2 \times m_1}$, $\boldsymbol{\mathcal{V}_2} \in \mathbb{R}^{W \times H \times 1 \times (m_1m_2)}$ and $\boldsymbol{\mathcal{V}_3} \in \mathbb{R}^{W \times \frac{H}{stride} \times 1 \times (m_1m_2m_3)}$ for TT-core 1, 2 and 3, respectively; and the final core's output will be the layer's output activation, $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{\frac{W}{stride} \times \frac{H}{stride} \times M}$.

Consider all the TT-cores are balanced to produce one pixel (all channels included) in the output feature map every T cycle. For the first TT-core to have enough data to start computing, n_1 data in the depth dimension needs to be ready. Since this is the first layer, consider these data are ready from the start, therefore the latency is 1. At each VMM cycle, the core produces c_1m_1 IPs where c_1 is the number of copies stored for the first TT-core.

The second core requires n_2c_2 IPs to be ready to start (c_2 is the number of copies stored for TT-core 2), and since the TT-cores are balanced to the same pixel-wise throughput T, the latency for this core to start will be T. The IPs produced by TT-core 2 would have the input depth dimension completely



	l	$l+1$
Conv layer	3×3 kernel, stride =1, padding =1	
Input shape (H,W,C)	(5, 5, 4)	(5, 5, 8)
ns	[2, 2, 3, 3]	[4,-,-,-]
ms	[2, 2, 2, 1]	[2,-,-,-]
# of copies	[1, 1, 2, 4]	NA

Fig. 7. Rank-4 TT-CONV pipelining with layer information in the inset table. The different colored cubes represent the tensors involved in the pipelining stages, demonstrating when each stage can start and what should be buffered between stages.

accumulated, ready to be contracted in the kernel space in the following cores.

The kernel-row dimension is accumulated in the third TT-core, and the inputs are streamed row-by-row, therefore, the TT-core will have to wait for $\mathcal{V}_2(1,1,1,1)$ and $\mathcal{V}_2(1,2,1,1)$ to be ready before it can start. It means it requires the previous TT-core to finish a whole row first. therefore, the latency for TT-core 3 is $W\times T+1$. In the last core, the input information is accumulated in the kernel dimension, the IPs required to start the computation is $\mathcal{V}_3(1,1,1,1)$ and $\mathcal{V}_3(2,1,1,1)$, making the latency for this core to start, $T+1$.

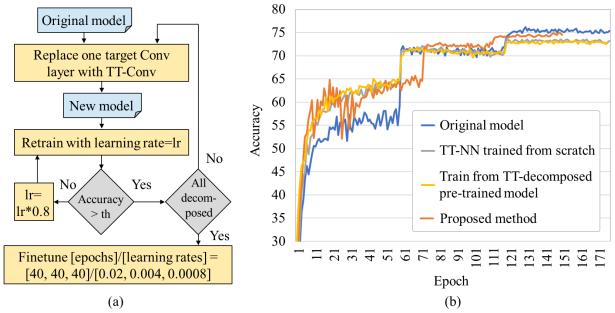


Fig. 8. (a) Iterative decompose and fine-tune method for TT-CNN and (b) preparation of TT-WRN with 3 different methods.

IV. Experiment

A. Training Method

To achieve a high compression ratio with minimum accuracy drop, we proposed an iterative decompose and fine-tune method to train the TT-CNN model. The training method is shown in Fig. 8(a) for WRN. The process starts with an untrained original WRN model and in each iteration, replace one traditional CONV layer with TT-CONV and fine-tune until a certain level of accuracy is met. After all the targeted layers are replaced with TT layers, the whole model is further trained to increase accuracy. We also tried training the TT-DNN from scratch directly and training from a decomposed pre-trained DNN. In Fig. 8(b), we show that this method achieves higher accuracy on CIFAR-100 [29] WRN-16 × 8 than the other two methods with fewer training epochs.

B. Factorization Scheme

The choice of ns and ms when factorizing a CONV layer would affect the performance of the TT-CNN. We follow prior works and choose rank-4 TT-decomposition and pick the TT-core ranks (r_1, r_2, r_3) with values from 16 to 32 [20], [21], [22], [23] for our experiment.

The **effective compression ratio** of the TT-CNN when implemented in tiled CIM would be different from the **intrinsic compression ratio** - which is the reduction in the number of parameters - because copies of parameters have to be stored to balance the pipeline. We select a few sets of factorization parameters (ns and ms) following a few principles: Firstly, factorization parameters are chosen to be close to each other [22]; secondly, to reduce computation count, we keep larger input factorization parameters in the earlier cores, so more IPs get contracted earlier in a forward computational sequence, on the contrary, the larger output factorization parameters are placed in later TT-cores; Lastly, we make sure the kernel space dimensions are factorized in later TT-cores for input factorization to maximize reuse.

We compare three types of input factorization schemes shown in Table I for the input dimension. The four depth sizes, 64, 128, 256, 512, are chosen because they are commonly used in state-of-the-art CNNs. Scheme A prioritizes keeping earlier factorization parameters larger; scheme B prioritizes keeping

TABLE I
DECOMPOSITION SCHEMES FOR TT-CNN INPUT DIMENSION

Scheme ID	Depth	n_1	n_2	n_3	n_4
A, B	64	8	4	2×3	3
	128	8	8	2×3	3
A	256	16	8	2×3	3
	512	16	16	2×3	3
B	256	8	8	4×3	3
	512	8	8	8	3×3
C	64	4	4	4	3×3
	128	8	4	4	3×3
	256	8	8	4	3×3
	512	8	8	8	3×3

TABLE II
DECOMPOSITION SCHEME FOR TT-CNN OUTPUT DIMENSION

Scheme ID	Number of Filters	m_1	m_2	m_3	m_4
I	64	2	2	4	4
	128	2	4	4	4
	256	4	4	4	4
	512	4	4	4	8
II	Scheme I reversed				

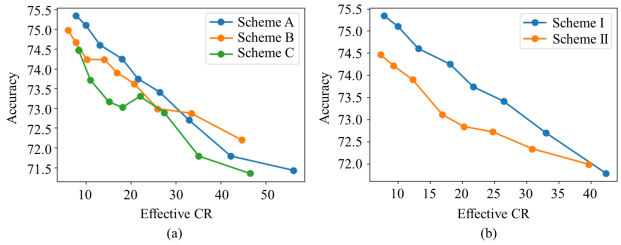


Fig. 9. WRN accuracy under different compression ratio for (a) different input factorization schemes and (b) different output factorization schemes.

the parameters closer to each other; and scheme C prioritizes maximizing reuse by factorizing the kernel space in the last TT-core only.

Accuracy comparison of the three schemes using output scheme I in II shown in Fig. 9(a) shows that scheme A achieves the highest effective compression ratio at low accuracy drop.

Factorization of the output dimension is relatively simple. We use scheme A for input factorization and compare performance between output scheme I and II. Fig. 9(b) shows that scheme I gives better performance.

V. RESULT

To evaluate the performance of the design points proposed in this work, we use a simulator, using parameters extracted from the sources as follows: The crossbar array properties are from [30]; the performance of the digital components is evaluated using parameters extracted from Verilog simulation with TSMC 22 nm process. The ADC parameters are extracted from an in-house low-power ADC, designed specifically for NN acceleration [31]. The system operates at 500M Hz frequency with the ADC and RRAM read operating at a different frequency of 50 MHz.

The simulator also includes a Pytorch-based [32] front-end layer for NN decomposition and CIM mapping estimation based on targeted throughput. Nonideality is included by adding random values in the forward pass of the Pytorch model. We experimented on WRN and a small VGG model VGG8 [1]. We were able to achieve an effective compression ratio of 12 (r=16) and 20 (r=24) with 0.8% and 1.4% accuracy drop respectively for WRN; and an effective compression ratio of 6 (r=16) and 11 (r=24) with 0.9% and 1.2% accuracy drop for VGG8.

A. TT-CNN Mapping on Tiled CIM

Tiled CIM architecture requires a properly designed system to handle data movement between CIM tiles [9], [11], [12]. Each CIM tile has an SRAM buffer shared by multiple CIM units to store the input/output activations required for the CIM units. Each CIM units has a crossbar array and peripheral circuits.

Our CIM unit follows the spec in [30] and consists of a 64×64 crossbar array with 2-bit RRAM-cells and 8 ADCs. For 8-bit serial inputs, each CIM unit will take 8×8 cycles to finish one VMM. 16 CIM units share one SRAM buffer with 128 read width. Positive and negative weights are mapped on the same crossbar, therefore, an 8-bit weight will require 8 RRAM cells to store, making the base VMM size 64×8. The mapping of NN on tiled architecture involves mapping the weights of the NN on the RRAM crossbars and allocating enough buffering storage for intermediate activations.

1) Weight and Factor Mapping: Mapping of factors is more challenging because the factors in the first and last TT-core are often small, with the first and last TT-cores having high and low aspect ratios, respectively, resulting in lower mapping utilization. We propose two types of partial read in the CIM units to address this challenge. The first type is the partial column read illustrated in Fig. 10(a). In the CIM process, after a crossbar read, the output in different bit-lines will be latched with a sample and hold (S&H) unit and converted in turns by shared ADCs. Therefore, since the bottleneck of the process is the ADC and not the crossbar read, we activate parts of the sourceline (SL) to achieve partial crossbar read. A different type of partial crossbar read is illustrated in Fig. 10(b). Compared to 8 1-bit serial inputs, by applying input to only half the rows with 4 2-bit serial inputs, 1 VMM can finish within half the time for one partial read, therefore the total time to finish the full VMM remains the same.

To summarize, Fig. 10(c) shows regular full-crossbar-read VMM, where after applying 1 input bit to the crossbar, the ADCs take 8 cycles to finish the conversion, and the same process is repeated 8 times. In Fig. 10(d), after each read, the ADCs only need 4 cycles to finish and are repeated 8 times for one partial read. In Fig. 10(e), after each read, the ADC works for 8 cycles but only repeats 4 times for one partial read. The two partial read schemes require negligible hardware overhead. By implementing partial read, the total RRAM size needed to map WRN can be reduced by 25.9%.

2) Inter-Core SRAM Buffering: In tiled CIM design, the size of the SRAM buffers needs to be chosen to be large enough to buffer the input needed by the CIM units that access the buffer. In traditional NN, the minimum inter-layer buffering size will be $((l-1)W+l) \times C$. In TT-CNN, the core that requires most IP buffering is the core that accumulates in the kernel-row dimension, e.g., the third TT-Core in Fig. 7. The required buffering size is now $((l-1))W+1) \times m_1 m_2 r_2$ times the channel size factorization in n_3 (i.e., n_3/l), which

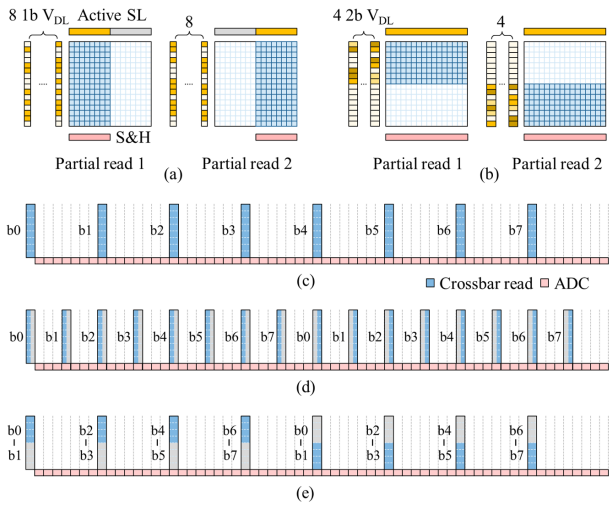


Fig. 10. Illustration of (a) partial column read and (b) partial row read of crossbar arrays. Crossbar read for different input bits and ADC conversion pipeline for (c) full crossbar read, (b) partial column read, and (e) partial row read.

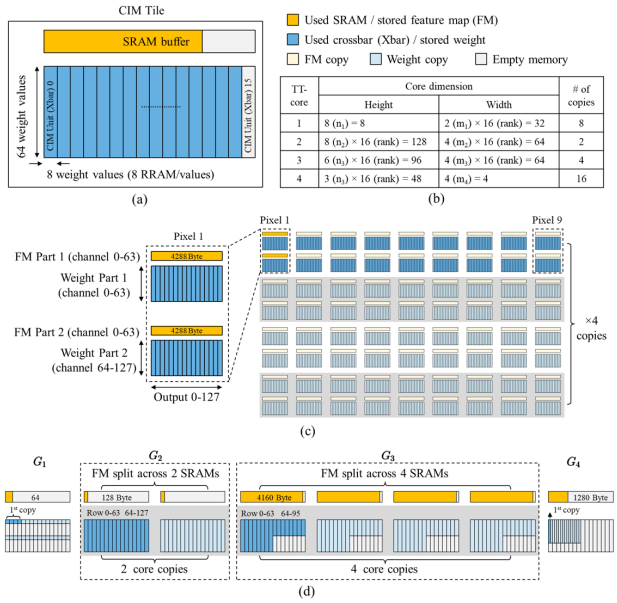


Fig. 11. (a) Illustration of a single CIM tile with 16 CIM units sharing one SRAM buffer. (b) Dimension of the WRN layer in TT format with rank of 16 through in 4 TT-cores. (c) Mapping of the uncompressed WRN layer with depth of 128 and output size of 128. (d) Mapping of the corresponding TT-decomposed layer.

is typically 2-3 times larger than the buffering requirement of the original CONV layer.

However, when mapping a layer on tiled architecture, it is common for a layer to cross multiple CIM tiles. In an original CONV operation, these tiles that work together to produce the set of output usually operate on a highly overlapped input space, meaning copies of activations need to be buffered across the tiles. However, for TT-CONV, the tiles that work together to produce the same layer output operates on different input space. A way to view this is to consider TT-core 3 in the previous example, which requires 4 copies of factors to match the target throughput, the entire input space is of size $WH \times m_1 m_2 r_2$, where the workload can be split into $m_1 m_2$ independent input space.

Mapping examples of the third convolutional layer in WRN with depth of 128 and 128 output channels in its uncompressed and TT-decomposed format are provided in Fig. 11. The minimum IFM buffering required is $(2 \times 32 + 3) \times 128 = 8576(Bytes)$ with 8 byte activations, split over 2 SRAM buffers, therefore, each SRAM buffer has to be at least 4288 Bytes. The largest buffering size required for the TT-cores is 16640 Bytes. In order to speed up the core to match targeted throughput, 4 sets of core values have to be stored and the activations buffered can be split into 4 SRAM buffers, therefore, each on is 4160 Bytes and is comparable to the requirement of the uncompressed layer.

B. Performance

When mapping CNN or TT-CNN on tiled CIM, the trade-off between total area size and the required number of VMM cycles for each frame is non-linear. Fig. 12(a) shows the required CIM-unit count at different throughputs for WRN and the duty rate distribution of the CIM-units. When the layers are not balanced to match a certain throughput, i.e., no additional weight copies are stored, the frame rate is low, and many CIM-units are underutilized. However, to double the frame rate, the total number of CIM-units only needs to increase by 6.8 % for uncompressed WRN. This is because only a few small layers (or cores) are slow, causing stalls in the pipeline. The area increment required to increase the frame rate increases at a higher frame rate and will eventually be directly correlated once the duty rate for all CIM-units reaches 100%.

The required CIM-unit count and average duty rate at different frame rates for TT-WRN (r=16) without and with inter-CONV-step reuse are shown in Fig. 12(b) and Fig. 12(c) respectively. The CIM-unit count is normalized with the unbalanced original model. The same trend is observed in the original CNN mapping. At a lower frame rate, exponential increment in frame rate can be achieved with little area increment: 4.8% for TT-WRN (no reuse), and 0.2% for TT-WRN (with reuse). In addition, in comparison with the original model, high hardware utilization can be achieved at a lower area overhead. For example, to achieve an 80% duty rate, uncompressed WRN requires 75% additional RRAM for mapping, TT-WRN(no reuse) requires 29% and TT-WRN(with reuse) requires 59%.

A direct comparison of throughput vs post-mapping RRAM capacity between the original CNN and TT-CNN in tiled CIM is shown in Fig. 13(a) and (b) for WRN and VGG8, respectively. The hollow dots indicated the smallest CIM architecture size required to map the model. It can be observed that the smallest size for the mapped uncompressed WRN is 8, and 4 times larger than the TT-WRN (with reuse) models under the same throughput for $r = 16$ and $r = 24$, respectively; For VGG8, the numbers are 15, and 8 times larger for $r = 16$ and $r = 24$, respectively. The TT-CNN sweet spot for achieving high hardware utilization with minimal required RRAM are the points right before the linear curve in Fig. 13. The corresponding RRAM capacities required for TT-WRN with $r = 16$ and $r = 24$ are 0.69 MB and 1.4 MB, respectively; the required capacity for TT-VGG8 with $r = 16$ and $r = 24$ are 0.6 MB, 1.2 MB, respectively. An RRAM-based CIM work

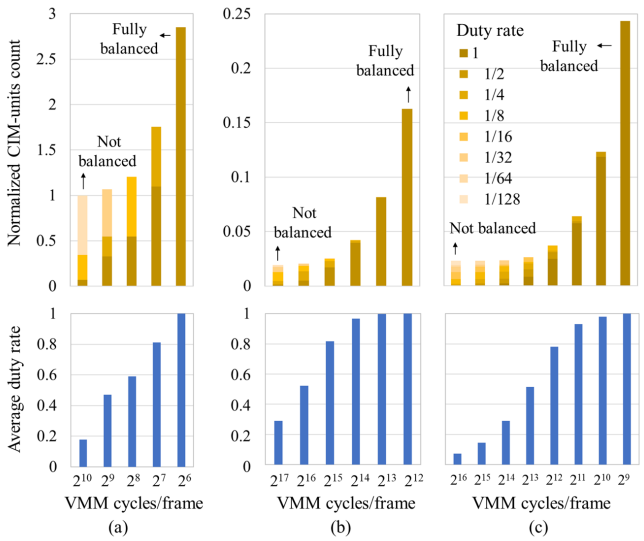


Fig. 12. (a) Required CIM-unit count at different frame rate for WRN, and the duty rate distribution of the CIM-units. Required CIM-unit count and average duty rate at different frame rate for TT-WRN (r=16) (b) without and (c) with inter-CONV-step reuse.

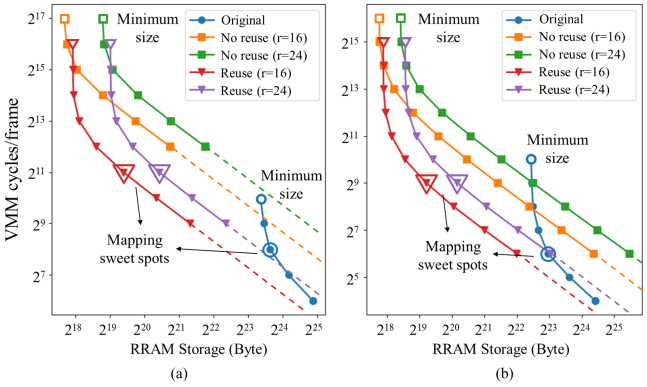


Fig. 13. Required number of VMM cycles vs required RRAM capacity for (a) WRN and (b) VGG8.

based on TSMC 22nm technology provides 1 MB RRAM at a die area of 18 mm^2 [10], comparable to the size required to map TT-WRN and TT-VGG8. The corresponding uncompressed WRN and VGG8 require 13 MB and 11.8 MB RRAM at the mapping sweet spot. Projecting from the reported area from [10], the required RRAM storage will be 234 mm^2 and 210.6 mm^2, respectively, which is unrealistically large.

Additional performance metrics are provided in Fig. 14 for TT-WRN and TT-VGG8 implemented using the proposed TT-CNN approach at two rank values, normalized against results from implementing the uncompressed models. The main benefit of TT-CNN is to allow an NN model to be mapped on a much smaller chip compared to its uncompressed counter, as can be seen in Fig. 14(a). The power required to run the models is also significantly reduced, as shown in Fig. 14(b). Since the proposed inter-CONV-step reuse method also reduces the required computation for running the model, the per area frame rate increases for TT-CNNs, and the latency is reduced, as shown in Fig. 14(c) and (d).

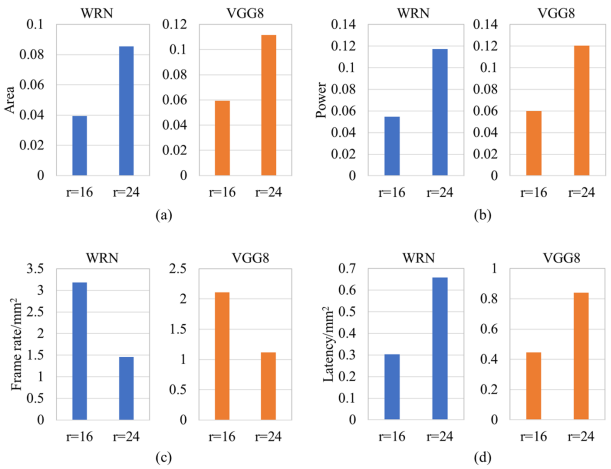


Fig. 14. Performance evaluation results of (a) area, (b) power, (c) frame rate per area and (d) latency per area, for implementing WRN and VGG8 using the proposed TT-CNN approach. The results are normalized against those from uncompressed models.

VI. Discussion

A. Non-Idealities in RRAM Devices

RRAM device non-idealities are known to induce error in CIM-based computing systems [33], [34]. Methods to minimize the error including optimizing RRAM programming methods, and variation-aware training are proposed in prior works [33], [34], [35], [36]. We conducted tests on WRN with quantization-aware and variation-aware training to evaluate the effect of device variations on TT-CIM vs. uncompressed models. The weights are quantized to 8-bit and the standard deviation of the normal distributed variations is set to be $0.05\times$ of the quantized weight range. We picked the best result out of 5 training processes (trained from scratch) for both uncompressed WRN and TT-WRN. The uncompressed model suffers a 1.41% accuracy drop while the TT-decomposed model suffers a comparable 1.99% drop. We hypothesize that though breaking one VMM into multiple VMMs could potentially lead to higher errors due to increased computing steps, the smaller matrix size in each TT core on the other hand helps reduce the accumulated error during CIM-based VMM. Additionally, the smaller matrix size helps contain the quantized weight range in a lower level since the variation caused during weight programming is proportional to the quantization range.

B. Comparison With Prior Work

A number of prior works related to hardware acceleration on compressed DNNs have been proposed. Reference [27] was one of the earliest works reported on TT-DNNs. The work presented a three-dimensional multilayer CMOS-RRAM architecture for the implementation of TT-decomposed fully connected layers. Reference [25] develops a computation-efficient inference scheme for TT-format DNN that aims to eliminate redundant computation within each VMM. The results were demonstrated on CNNs and RNNs in addition to fully connected layers. Reference [26] demonstrated an SRAM-based CIM chip for TT-DNN inference with the contractional

TABLE III
PERFORMANCE COMPARISON

Work		ISSCC '22 [10]	VLSI '23 [31]	TIE [25]	ISSCC '21 [26]	This work	
						84 tile	176 tile
Architecture		RRAM-CIM	RRAM-CIM	TT digital	TT SRAM-CIM	TT RRAM-CIM	
Node		22nm	65nm	28nm	28nm	22nm	
Weight capacity	Original	1M	8K	N/A (Not CIM)	128K	0.66M	1.4M
	Compressed	N/A	N/A		294K	10.8[1] -11.0[2] M	
TOPs/W	Original	21.6	20.7	N/A	N/A	10.1	10.1
	Compressed	N/A	N/A	71.9[3]	5.99[4]	23.1[1] , 21.0[2]	10.7[1] , 10.4[2]
TOPs	Original	0.14	0.057	N/A	N/A	1.08	2.25
	Compressed	N/A	N/A	2.86[3]	0.256[4]	2.46[1] , 1.96[2]	2.35[1] , 1.89[2]

WideResNet[1], VGG8[2], VGG16 CONV layers [3], ResNet20[4]

TT-VMM discussed in II-C. With TT decomposition, all off-chip weight accesses are eliminated despite the small chip size. In comparison, this work proposed an extra level of reuse on top of matrix contraction and demonstrated the mapping and organization of implemented TT-CNN on a traditional fully weight stationary tiled-based CIM architecture.

Table III summarizes the performance of this work and related works, including two RRAM-CIM architectures, one with a relatively large RRAM capacity of 1 MB [10] and another one employing the ADCs used in this work for performance evaluation [31], and two TT-CNN related works [25] and [26]. For this work, we considered 2 configurations, a smaller one to map the models with rank=16 with 84 tiles, and a larger one with 176 tiles to map models of rank=24. With the TT scheme presented in this work, fully weight stationary CIM architectures can support DNN models with sizes up to 16x more than the original RRAM capacity, enabling the CIM system to eliminate the need for off-chip weight access while supporting medium-to-large DNN models. Unlike many DNN compression techniques, TT-CIM requires no hardware specifications and can be supported by traditional tiled CIM. In addition, the TOPs/W and TOPs performance are improved with the proposed reuse scheme. In comparison to digital TT-DNN accelerators, TT-CIM enables fully weight stationary CIM by eliminating off-chip DRAM access and thus preserves the energy and throughput benefits of FWS CIM. In addition, with the mapping and pipelining schemes presented, no hardware overhead is required to support TT-CIM.

VII. CONCLUSION

Tensor train decomposition is a promising technique for compressing CNN models and maintaining the computational structure to deal with the challenge of large on-chip weight storage requirements in fully weight stationary CIM. Prior work related to TT-CNN optimizes computation within each CONV step and views each CONV step as independent VMMs. In the proposed TT-CIM theme, we view a TT-CONV as a whole and optimized reuse across CONV steps. To efficiently implement TT-CNN in tiled CIM, more stages of pipelining need to be implemented, requiring optimization of the balancing, scheduling, and buffering of the pipeline. Combining mapping techniques to achieve higher RRAM utilization, we show the proposed TT-CIM can achieve a high effective compression ratio reducing the required RRAM capacity drastically in fully weight stationary CIM systems compared to the original CNN model.

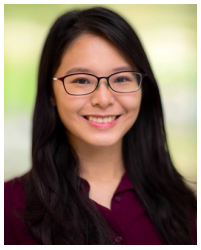
ACKNOWLEDGMENT

The authors would like to thank Ziyu Wang, Yongmo Park, and Sangmin Yoo for insightful discussions.

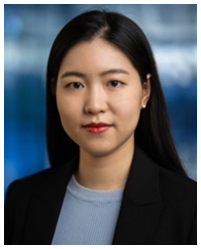
REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[3] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.
[4] J. Gu et al., "Recent advances in convolutional neural networks," *Pattern Recognit.*, vol. 77, pp. 354–377, May 2018.
[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.
[6] M. A. Zidan, J. P. Strachan, and W. D. Lu, "The future of electronics based on memristive systems," *Nature Electron.*, vol. 1, no. 1, pp. 22–29, Jan. 2018.
[7] A. Sebastian, M. Le Gallo, and E. Eleftheriou, "Computational phase-change memory: Beyond von Neumann computing," *J. Phys. D, Appl. Phys.*, vol. 52, no. 44, Oct. 2019, Art. no. 443002.
[8] P. Yao et al., "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, no. 7792, pp. 641–646, Jan. 2020.
[9] X. Wang et al., "TAICHI: A tiled architecture for in-memory computing and heterogeneous integration," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 2, pp. 559–563, Feb. 2022.
[10] J.-M. Hung et al., "An 8-Mb DC-current-free binary-to-8b precision ReRAM nonvolatile computing-in-memory macro using time-space-readout with 1286.4-21.6TOPS/W for edge-AI devices," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 65, Feb. 2022, pp. 1–3.
[11] A. Shafiee et al., "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 14–26, 2016.
[12] X. Liu et al., "RENO: A high-efficient reconfigurable neuromorphic computing accelerator design," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2015, pp. 1–6.
[13] Q. Wang, X. Wang, S. H. Lee, F.-H. Meng, and W. D. Lu, "A deep neural network accelerator based on tiled RRAM architecture," in *IEDM Tech. Dig.*, Dec. 2019, pp. 14.4.1–14.4.4.
[14] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.
[15] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.

[16] C. Chu et al., "PIM-prune: Fine-grain DCNN pruning for crossbar-based process-in-memory architecture," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.
[17] J. Meng, L. Yang, X. Peng, S. Yu, D. Fan, and J.-S. Seo, "Structured pruning of RRAM crossbars for efficient in-memory computing acceleration of deep neural networks," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 5, pp. 1576–1580, May 2021.
[18] F.-H. Meng, X. Wang, Z. Wang, E. Y. Lee, and W. D. Lu, "Exploring compute-in-memory architecture granularity for structured pruning of neural networks," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 12, no. 4, pp. 858–866, Dec. 2022.
[19] L. Liang et al., "Crossbar-aware neural network pruning," *IEEE Access*, vol. 6, pp. 58324–58337, 2018.
[20] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 442–450.
[21] I. V. Oseledets, "Tensor-train decomposition," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, Jan. 2011.
[22] T. Garipov, D. Podoprikhin, A. Novikov, and D. Vetrov, "Ultimate tensorization: Compressing convolutional and FC layers alike," 2016, *arXiv:1611.03214*.
[23] D. Wang, G. Zhao, G. Li, L. Deng, and Y. Wu, "Compressing 3DCNNs based on tensor train decomposition," *Neural Netw.*, vol. 131, pp. 215–230, Nov. 2020.
[24] D. Liu, L. T. Yang, P. Wang, R. Zhao, and Q. Zhang, "TT-TSVD: A multi-modal tensor train decomposition with its application in convolutional neural networks for smart healthcare," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 18, no. 1s, pp. 1–17, Feb. 2022.
[25] C. Deng, F. Sun, X. Qian, J. Lin, Z. Wang, and B. Yuan, "TIE: Energy-efficient tensor train-based inference engine for deep neural network," in *Proc. ACM/IEEE 46th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2019, pp. 264–277.
[26] R. Guo et al., "A 5.99-to-691.1 TOPS/W tensor-train in-memory-computing processor using bit-level-sparsity-based optimization and variable-precision quantization," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 64, Feb. 2021, pp. 242–244.
[27] H. Huang, L. Ni, K. Wang, Y. Wang, and H. Yu, "A highly parallel and energy efficient three-dimensional multilayer CMOS-RRAM accelerator for tensorized neural network," *IEEE Trans. Nanotechnol.*, vol. 17, no. 4, pp. 645–656, Jul. 2018.
[28] S. Zagoruyko and N. Komodakis, "Wide residual networks," 2016, *arXiv:1605.07146*.
[29] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," M.S. thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2009.
[30] F. Cai et al., "A fully integrated system-on-chip design with scalable resistive random-access memory tile design for analog in-memory computing," *Adv. Intell. Syst.*, vol. 4, no. 8, Aug. 2022, Art. no. 2200014.
[31] J. M. Correll et al., "An 8-bit 20.7 TOPS/W multi-level cell ReRAM-based compute engine," in *Proc. IEEE Symp. VLSI Technol. Circuits (VLSI Technol. Circuits)*, Jun. 2022, pp. 264–265.
[32] A. Paszke et al., "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates, 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
[33] C.-C. Chang et al., "Device quantization policy in variation-aware in-memory computing design," *Sci. Rep.*, vol. 12, no. 1, pp. 1–12, Jan. 2022.
[34] G. Krishnan et al., "Robust RRAM-based in-memory computing in light of model stability," in *Proc. IEEE Int. Rel. Phys. Symp. (IRPS)*, Mar. 2021, pp. 1–5.
[35] Y. Wu et al., "Bulk-switching memristor-based compute-in-memory module for deep neural network training," 2023, *arXiv:2305.14547*.
[36] K.-D. Suh et al., "A 3.3 V 32 Mb NAND flash memory with incremental step pulse programming scheme," *IEEE J. Solid-State Circuits*, vol. 30, no. 11, pp. 1149–1156, Nov. 1995.



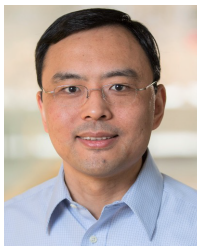
Fan-Hsuan Meng received the B.S. degree in electrical engineering and the M.S. degree in electronics engineering from National Tsing Hua University, Hsinchu, Republic of China, in 2014 and 2016, respectively. She is currently pursuing the Ph.D. degree with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA. She was an Engineer in process integration for 7 nm FinFET devices with TSMC, Hsinchu, from 2016 to 2017. Her research interests include memristive devices, and its application for neuromorphic computing. She is also working on system level optimization for in-memory computing-based neural network accelerators.



Yuting Wu (Student Member, IEEE) received the B.S. degree in microelectronics from Peking University, Beijing, China, in 2017, and the Ph.D. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2023. She is currently working on developing machine learning models for edge devices with Ambarella Inc., Santa Clara, USA. Her research interests include efficient machine learning models and hardware accelerator.

Zhengya Zhang (Senior Member, IEEE) received the B.A.Sc. degree in computer engineering from the University of Waterloo in 2003 and the M.S. and Ph.D. degrees in electrical engineering from the University of California at Berkeley (UC Berkeley) in 2005 and 2009, respectively.

He has been a Faculty Member with the University of Michigan, Ann Arbor, since 2009, where he is currently a Professor with the Department of Electrical Engineering and Computer Science. His research interests include low-power and high-performance VLSI circuits and systems for computing, communications, and signal processing. He was a recipient of the University of Michigan College of Engineering Neil Van Eenam Memorial Award in 2019, the Intel Early Career Faculty Award in 2013, the National Science Foundation CAREER Award in 2011, and the David J. Sakrison Memorial Prize from UC Berkeley in 2009. He has been serving on the Technical Program Committee of the IEEE Custom Integrated Circuits Conference (CICC) since 2019. He served as an Associate Editor for IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS from 2015 to 2022, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I: REGULAR PAPERS from 2013 to 2015, and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II: EXPRESS BRIEFS from 2014 to 2015. He served on the Technical Program Committee of the IEEE VLSI Symposium on Technology and Circuits from 2019 to 2022. He is an IEEE Solid-State Circuits Society Distinguished Lecturer from 2023 to 2024.



Wei D. Lu (Fellow, IEEE) received the B.S. degree in physics from Tsinghua University, Beijing, China, in 1996, and the Ph.D. degree in physics from Rice University, Houston, TX, USA, in 2003. From 2003 to 2005, he was a Post-Doctoral Research Fellow with Harvard University, Cambridge, MA, USA. He joined the Faculty of the University of Michigan in 2005. He is currently a Professor with the Electrical Engineering and Computer Science Department, University of Michigan. His research interests include resistive-random access memory (RRAM), memristor-based logic circuits, neuromorphic computing systems, aggressively scaled transistor devices, and electrical transport in low dimensional systems. He was a recipient of the NSF CAREER Award.