BCIM: Efficient Implementation of Binary Neural Network Based on Computation in Memory

ABSTRACT

Applications of Binary Neural Networks (BNNs) are promising for embedded systems with hard constraints on computing power. Contrary to conventional neural networks with the floating-point datatype, BNNs use binarized weights and activations which additionally reduces memory requirements. Memristors, emerging non-volatile memory devices, show great potential as the target implementation platform for BNNs by integrating storage and compute units. The energy and performance improvements are mainly due to 1) accelerating matrix-matrix multiplication as the main kernel for BNNs, 2) diminishing memory bottleneck in von-Neumann architectures, 3) and bringing massive parallelization. However, the efficiency of this hardware highly depends on how the network is mapped and executed on these devices. In this paper, we propose an efficient implementation of XNOR-based BNN to maximize parallelization while using a simple sensing scheme to generate activation values. Besides, a new mapping is introduced to minimize the overhead of data communication between convolution layers mapped to different memristor crossbars. This comes with extensive analytical and simulation-based analysis to evaluate the implication of different design choices considering the accuracy of the network. The results show that our approach achieves up to 10× energy-saving and 100× improvement in latency compared to the state-of-the-art in-memory hardware design.

KEYWORDS

Memristor, computation-in-memory, Binary Neural Network.

ACM Reference Format:

. 2018. BCIM: Efficient Implementation of Binary Neural Network Based on Computation in Memory . In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Machine learning algorithms and specifically Deep Neural Networks (DNNs) have pushed the state-of-the-art designs and become prominent in a variety of applications, including, but not limited to language processing [1], object recognition [2], and image classification [3, 4]. Designing larger networks and the ability to train them with advanced algorithms was the main driver to enable performing complex applications for several years. Besides advanced algorithms, hardware implementation and its challenges play a major role in the deployment and development of DNN applications specifically for embedded systems. One of the main hardware challenges of NN is the large data set (weights) that has to be stored and performed computation on (memory wall) [5]. Neural networks usually use floating-point computation which requires large storage and many resources. As a response to this challenge, Binary Neural Networks (BNNs), where the weights and activation values are binarized, receives more attention from researchers [6]. A BNN reduces memory consumption and simplified computations which leads to a higher energy-efficient system. However, this efficiency highly depends on the implementation of the network considering the hard constraints of embedded systems. Therefore, considerable research is required to ensure the effectiveness of BNNs for state-of-the-art applications.

In recent years, many works have been published with a focus on the algorithmic optimizations of BNNs. Minimizing quantization error [7–9], improving the network loss function [10, 11], and reducing gradient error [12, 13] have been the main topics of interest. From the hardware perspective, besides using traditional systems (CPU, GPU, and FPGA) [14–16], memristor-based BNN accelerators are getting more attention due to reduced communication overhead, as the main bottleneck in traditional computing systems, by deploying computation in memory as a concept [17]. However, using memristors for signed numbers is challenging. From this perspective, existing works can be classified into hardware or algorithmic solutions. Positive and negative values can be mapped to different memristors [18–20]. Other approaches are considering one- [21] or two-column reference memristors [22] while the weights and activation are presented as unsigned numbers. In general, these approaches require more devices, increase design complexity, and reduce energy/performance-efficiency of the system. As an algorithmic solution, the signed multiply-and-accumulate which is the main operation in BNNs can be converted to XNOR operations [7]. Using this method, it was proposed to use memristors as an activation function [23], but this induces endurance, energy, and performance issues due to the excessive programming. To ensure the accuracy of XNOR operations against device variation, a new memristor crossbar structure based on differential sensing is used [24]. However, XNOR operations are forced to be performed sequentially due to the sensing mechanism. All these overheads drive researchers to explore efficient implementations of BNN specifically for embedded systems. This is inevitable considering advanced workloads demanding more energy and computing times.

This work advances the state-of-the-art by proposing an efficient implementation of BNNs. In this design, we mimic the functionality of ADC and the required following digital processing by a Sense Amplifier (SA) while it allows simultaneous row activation to maximize resource utilization on the crossbar and enhance the performance. Extensive analytical analysis and simulations are performed to ensure the accuracy of the design considering the

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

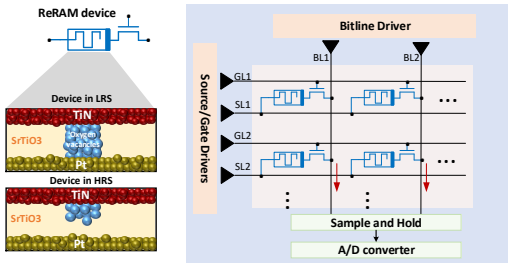

Figure 1: Memristor ReRAM device behavior in LRS and HRS mode as well as a crossbar structure

scenarios where the design behaves as an approximation. The effect of the number of references for the SA and the distance between the values of the references are studied. Furthermore, we minimize data communication between layers by proposing a novel mapping of the weights and activation values into the crossbar and its input buffer, respectively. Finally, we investigate the efficiency of our approach on different network structures in terms of accuracy, energy, and performance by developing our PyTorch-based simulation platform intended to be open-sourced. The platform can mimic the behavior of the crossbar and allows for more characteristics and non-idealities to be integrated and explored for different networks. In summary, this paper presents the following main contributions:

- We propose an energy-efficient and highly parallel implementation of XNOR-based BNNs where the functionality of ADC and the required post-processing after that is modeled by a SA.
- We perform extensive analytical as well as simulation-based analysis where the proposed implementation behaves as an approximation in order to comprehend the implication of SA on the accuracy of the design.
- We present an efficient mapping of the weights and activation values to improve data utilization and minimize the number of communication between network layers. The technique is general and can be applied to non-binary networks as well.

2 PRELIMINARY

In this section, two topics are covered. First, we provide background information about memristor devices and the operations supported in a crossbar array. Second, we explain briefly the basics of binary neural networks.

2.1 Memristor devices

Contrary to charge-based memories, memristor devices are categorized as non-volatile memory where data can be represented as a low resistive state (LRS) and a high resistive state (HRS) by application of appropriate voltage signals. Many technologies can be used to build these devices such as Resistive Random-Access Memory (ReRAM or RRAM) or Phase-Change Memory (PCM) [25, 26]. As an example, ReRAM consists of a metal-insulator-metal stack where the device is set and reset by changing the polarity of the programming voltage to form or dissolve the conducting filament (Figure 1). The resistance level indicates the logic value intended to be stored

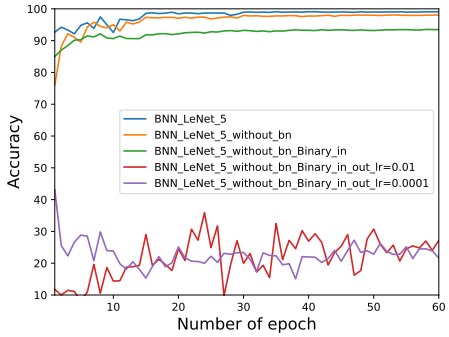

Figure 2: Accuracy of binarized LeNet5 network and the impact of input/output layer binarization as well batch normalization (bn) on accuracy loss

in the device. In order to read the device without disturbance, a small voltage should be applied and the current (voltage) through (across) the device should be sensed. Figure 1 depicts a 1T1R crossbar structure where three drivers are employed to program or read the devices. In the case of read or computational operations, the current passes through the bitline is sensed, and converted to digital domain using a SA or ADC. The main computational operations that can be performed on the crossbar include addition, logical operations, and Matrix-Matrix Multiplication (MMM). Besides the capabilities of co-locating computation and storage together, huge parallelism can be achieved within a single memory array (crossbar and its periphery) as well as at the inter-array level. These are the main drivers that attracts researchers to exploit this concept for different state-of-the-art applications [17].

2.2 Fundamentals of Binary Neural Networks

Nowadays, deeper neural networks have been developed to be able to perform advanced tasks such as complex classification or image segmentation. However, implementing these networks in embedded platforms with limited storage and computation units is challenging specifically in consideration of strict energy/performance constraints. Despite conventional neural networks with high precision datatypes, in BNNs, weights and activations are binarized to make the network extremely compact. Equation 1 shows a simple binarization rule that can be applied to both activations (input tensors) and weights where B_ω and B_I are the binarized weights and input tensors, respectively. The binarization not only saves on the storage usage, but also reduces the expensive multiply-accumulate operation to a simple addition.

$$B_\omega = \begin{cases} +1 & if\ \omega \geq 0 \\ -1 & if\ \omega < 0 \end{cases} \quad B_I = \begin{cases} +1 & if\ I \geq 0 \\ -1 & if\ I < 0 \end{cases} \tag{1}$$

Although binarization enhances the system's efficiency in terms of memory usage, energy, and performance, it usually comes at the cost of accuracy loss compared to its high-precision counterpart. Therefore, using proper methods and algorithms to preserve the accuracy of the network as high as possible is essential. Each iteration of training a network can be divided into three steps; forward pass, backward propagation, and parameter update. The weights during backward propagation and forward pass are binarized while

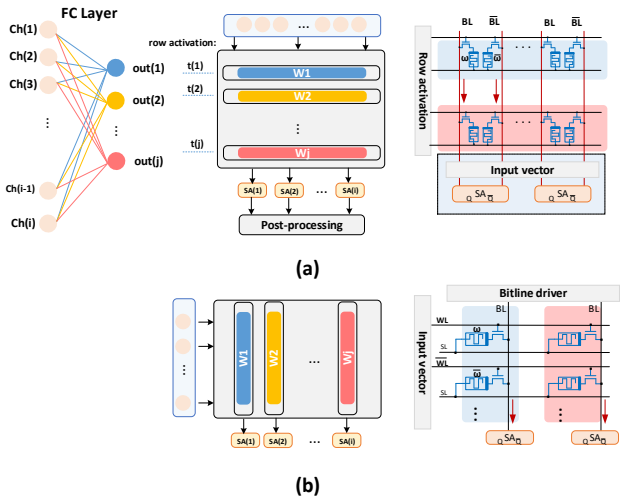

Figure 3: (a) BNN implementation using differential sensing and sequential XNOR operation [24] (b) proposed design where massive XNOR operations are performed in parallel

keeping high precision weights during parameter update is necessary. Since parameter changes obtained by gradient descent are tiny, binarization ignores these changes and the network cannot be trained [7, 27]. In addition, binarizing the input and output layer usually results in a huge accuracy loss. Figure 2 depicts the accuracy of the binarized LeNet5 network trained for the MNIST dataset. This clearly shows the impact of binarizing the input and output layers as well as batch normalization (bn) on the accuracy of the network.

3 MEMRISTOR-BASED ACCELERATORS FOR BNN

Memristor crossbar arrays are tailored to perform analog VMM with more significant energy efficiency compared to their digital counterpart (CPU/GPU) [28]. Although some memristor devices can potentially be programmed to multi-resistance levels, they have higher reliability, stability, and accuracy when fewer resistance levels are used. Hence, BNN-based applications where the main kernel is binarized VMM are the promising targets to be implemented using memristor devices. A small-scale demonstration of BNN on memristor devices is presented in [29] with focusing mainly on device variation and its effect on BNNs. A new methodology is proposed in [30] to make the design more tolerant against device variations to be able to activate more word-lines and perform more computation at the same time. Based on Equation 1, BNNs require signed representation, but negative numbers cannot be directly stored in memristors. Considering that, the existing BNN accelerators can be classified into two categories based on the solution that they employ.

- Hardware solutions: In order to deal with signed numbers, the weights and activation values can be converted each to two vectors containing only positive values [18]. The two vectors of the weights and activation values are downloaded to the corresponding memristors and crossbar's input ports. Subsequently, the four possible partial results are computed. This requires a high number of memristor devices which translates to low area and energy efficiency. In addition, more complex input drivers are required to provide current in both directions. A similar approach is mapping positive and negative weights into different crossbars [19, 20]. Besides, ADC is exploited to compute the partial result when a BNN layer size is larger than the crossbar size. However, using ADCs imposes significant energy and area overhead to the system. An interesting approach is using one- [21] or two-column reference memristors [22] while the weights and activations are presented as {0,1}. In this design, the current flowing through the reference column(s) has to be mirrored equal to the number of columns in the crossbar. This increases the design complexity and energy consumption of the system. In addition, when a layer size cannot fit into a crossbar, it gets critical to have a flexible referencing scheme to avoid accuracy loss. We discuss this more in Section 5.

- Algorithmic solutions: Binary multiply and accumulate operation can be replaced by **XNOR+popcount+post processing**. As a result, the weights and activations for BNN can be presented as unsigned {0,1} values. As a consequence and considering memristor crossbars, it makes the implementation simpler, but additional digital processing has to be done after the crossbar. Content-addressable memory (CAM) structure based on binary XNOR operation is used for BNN [23]. In this design, the activation function is implemented by a memristor where its state determines the input value for the next layer. However, this suffers from an extremely high number of device programming which causes challenges in terms of reliability, performance, and energy. An XNOR-based robust design to device imperfections is proposed using a differential sensing mechanism [24]. This design is closest to this work and is considered for our baseline. Figure 3(a) illustrates how a fully connected layer is mapped to a crossbar. Due to the structure of the crossbar and mapping of the weights, outputs are generated sequentially. Besides, additional digital processing is required to generate the final result.

Considering the limitations and challenges of existing works, a high- performance and energy-efficient design of BNN is highly demanded.

4 METHODOLOGY

In this section, first, we discuss the implementation of BNNs on memristor crossbars based on XNOR operation. Second, we explain how the crossbar's input buffer containing activation values is managed to minimize data transfer between crossbars.

4.1 Proposed BNN implementation

The multiply-accumulate operation between two signed binarized vectors can be replaced by *XNOR* and popcount operations [7]. To achieve that, first, the vectors are converted to unsigned where '-1' is represented as '0'. This is helpful since it simplifies the mapping of weights to the crossbar without concern for negative values. Second, by applying Equation 2, the final value is obtained where *A'* is the unsigned representation of vector *A*. In this equation, popcount() returns the number of ones in a bitstream and *'vector*

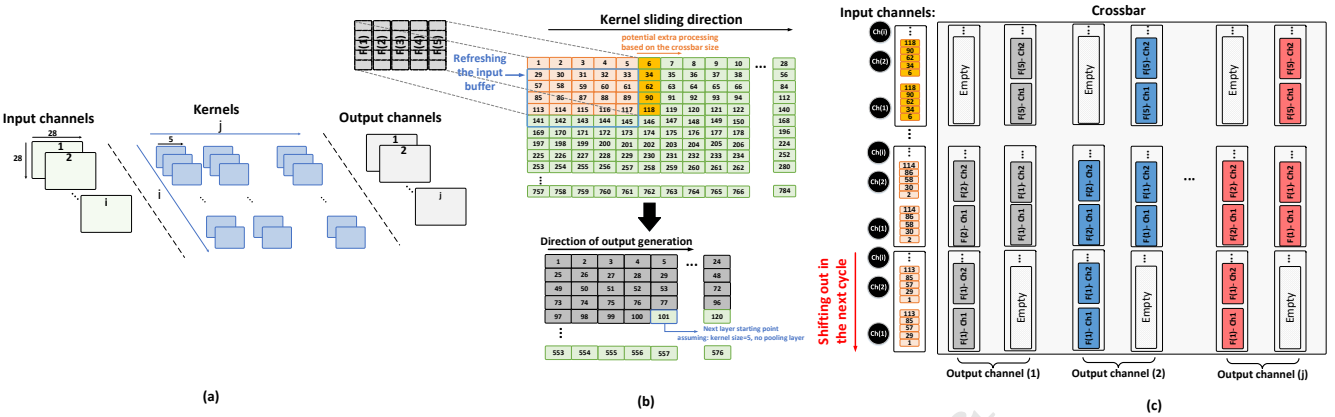

Figure 4: (a) Example of a CNN layer (b) details of a convolution operation with 5×5 and 28×28 kernel and input size (c) mapping of the activation values to the input buffer and kernels to the crossbar based on the proposed approach to minimize data transmission between layers by only streaming the newly computed activation values into the input buffer

size' is the length of the two vectors. In the following, an example is provided to have better clarification.

$$A * B = 2 * Popcount(A^{'} \odot B^{'}) - vector\ size \tag{2}$$

$A = [1, -1, -1, 1]\ B = [-1, 1, 1, 1] \Rightarrow A * B = -2$
$A^{'} = [1, 0, 0, 1]\ B^{'} = [0, 1, 1, 1] \Rightarrow A^{'} \odot B^{'} = [0, 0, 0, 1]\ A * B = 2 * Popcount(A^{'} \odot B^{'}) - vector\ size = -2$

By applying the above method to a fully connected layer, the process of generating the activation value for the next layer can be expressed as (similarly to a convolution layer):

$$out_m = Sign(2 * \sum_{k=1}^{I} (Ch_k \odot \omega_{k,m}) - vector\ size) \tag{3}$$

where Ch_k represents the activation value for the current layer, $\omega_{k,m}$ is the weight related to the k^{th} input and m^{th} output, and I is equal to the number of inputs of the current layer. Figure 3(b) depicts how the above equation is implemented on a crossbar. Despite the approach illustrates in Figure 3(a), the summation in the equation is performed in an analog way in the crossbar. In this mapping, each column corresponds to one output of the layer and they can perform the operations in parallel. However, in order to generate the final value, besides the sign function, other operations have to be performed. As to achieve that, it may require to place ADC to generate the actual value of this analog summation and perform other operations in the periphery of the crossbar accordingly. However, by changing the sign operation to a comparator where its reference is obtained from Equation 4, the output can be efficiently computed.

$$SA\ reference = vector\ size/2 \tag{4}$$

Based on this approach, first, we maximized the number of parallel activation values that can be computed for a BNN layer. Second, to avoid reducing this efficiency by using a high-resolution ADC, a simple analog comparator with a smart referencing value is deployed. This not only performs the sign operation, but also omits extra digital processing in the periphery.

4.2 Efficient data movement

Data movement between the BNN layers may influence the performance and energy of the system [31], but is often overlooked by the existing works. In this subsection, we focus on how the data should be transferred from one convolutional layer to the next to minimize the number of transactions and the size of a buffer placed between layers. This approach can be utilized for both binary and non-binary datatypes.

Figure 4(a) depicts an example of convolution layer where the kernel matrix is convolved into the *"i"* input channels to generate data for the *"j"* output channels. In this example, the input size for each channel and the kernel size are 28×28 and 5×5, respectively. Figure 4(b) illustrates the details of the convolution operation where each kernel slides on a corresponding input channel to produce the partial result. The kernels are programmed to the crossbar while the data of input channels corresponding to the current operating window (highlighted by light orange) are buffered and sent to the wordlines of the crossbar. When the operating window slides, the data has to be sent and reorganized in the buffer to be matched to the weights of the kernel programmed into the crossbar. However, bringing the whole data again for the next operating window is not an efficient way since most of it already exists in the input buffer of the crossbar from the previous operating window.

To provide better data utilization and reduce the number of transactions, Figure 4(c) demonstrates an efficient mapping of kernels in the crossbar as well as activation value in the input buffer. In this approach, the kernels and the input data within the operating window are sliced into columns. The same columns for different input channels are packed together and placed in the input buffer. The next columns are stacked on top of each other as highlighted by the light orange color in Figure 4(c). The kernels are also treated the same way. By doing that, when the operating window slides to the right (assuming stride is one), the left-most columns for all the input channels are shifted out and new data corresponding to the right-most columns are streamed into the buffer. There is no need to change the mapping of the kernels in the crossbar and they always reside in front of the right inputs. When the operating

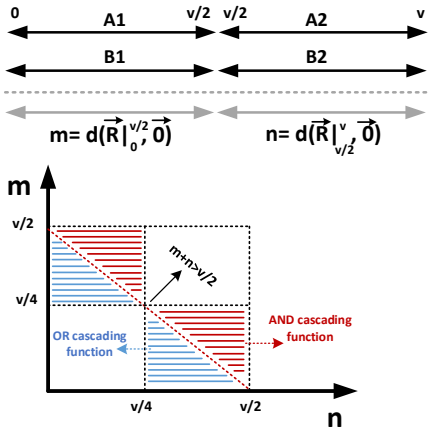

Figure 5: Illustration of the regions where logical *AND* and *OR* functions inject inaccuracy into the network

window reaches the last columns, it has to be shifted down and starts from the most left column again. Therefore, the input buffer is refreshed and filled with data highlighted by the blue window in 4(b). As a result, maximum data is utilized when the operating window slides while the input buffer can be implemented as simple as possible.

In order to maximize the performance, we can exploit parallelization and pipelining. In case the crossbar dimension is large enough, the computation for current and next operating windows can be performed in parallel. As illustrated in Figure 4(b) and (c), an extra column (highlighted by bright orange) required for the next operating window is placed into the input buffer of the crossbar. Besides, we have to consider another column in the crossbar to be able to generate the value for both operating windows simultaneously. It has to be taken into account that this extra input set should not contribute to the computation of the current window. Therefore, the memristors located in the first column and in front of this extra input set should be programmed to logic value '0'. It is worth mentioning that the kernels for other output channels are programmed to different columns of the crossbar to maximize parallelization. However, in case the crossbar has a lower number of columns, we need to deploy more crossbars to avoid an excessive number of reprogrammings. Besides parallelization, the same pipelining approach presented in [31] can be applied in this work. Depending on the kernel size of the next layer in the network, when enough elements are produced for the output channels of the current layer, the operation can be started for the next layer.

5 INTRA-LAYER ACCURACY ANALYSIS

In Section 4, the proposed implementation was presented where a single SA can generate the activation value for the next BNN layer (see Figure 3). However, if the weights that are supposed to be in a single column of a crossbar cannot fit into it, they have to be split and mapped to more columns. Therefore, the final activation value has to be calculated from the intermediate activation values obtained from different sets of columns. This is where inaccuracy is injected into the network with a certain probability. In the following, the ideal situation is formulated where the crossbar size is equal or greater than the vector size. $\overrightarrow{A}$ and $\overrightarrow{B}$ are the two binary vectors and $d(\overrightarrow{R}, \overrightarrow{0})$ is the hamming distance between vectors $\overrightarrow{R}$ and $\overrightarrow{0}$.

Vector size = v, *Crossbar size* = $C \geq v$
input 1: $\overrightarrow{A}$, input 2: $\overrightarrow{B}$
$\overrightarrow{R} = \overrightarrow{A} \odot \overrightarrow{B}$

$$out_{golden}(\overrightarrow{R}) = \begin{cases} 1 & \text{if} \quad d(\overrightarrow{R}, \overrightarrow{0}) > v/2 \\ 0 & \text{otherwise} \end{cases}$$

In case the crossbar size is not big enough, the formulation is changed as presented below. As an example, we assume the crossbar size is half of the vector size. Therefore, each vector has to be split into two parts and mapped to two columns of the crossbar.

Vector size = v, *Crossbar size*: $C = v/2$

input 1: $\overrightarrow{A}|_0^{v/2}, \overrightarrow{A}|_{v/2}^{v}$ where $\overrightarrow{A} = [\overrightarrow{A}|_0^{v/2}, \overrightarrow{A}|_{v/2}^{v}]$
input 2: $\overrightarrow{B}|_0^{v/2}, \overrightarrow{B}|_{v/2}^{v}$ where $\overrightarrow{B} = [\overrightarrow{B}|_0^{v/2}, \overrightarrow{B}|_{v/2}^{v}]$

$$\overrightarrow{R}|_0^{v/2} = \overrightarrow{A}|_0^{v/2} \odot \overrightarrow{B}|_0^{v/2} \quad \overrightarrow{R}|_{v/2}^{v} = \overrightarrow{A}|_{v/2}^{v} \odot \overrightarrow{B}|_{v/2}^{v}$$

$$out_{p1}(\overrightarrow{R}|_0^{v/2}) = \begin{cases} 1 & \text{if} \quad d(\overrightarrow{R}|_0^{v/2}, \overrightarrow{0}) > (v/2)/2 \\ 0 & \text{otherwise} \end{cases}$$

$$out_{p2}(\overrightarrow{R}|_{v/2}^{v}) = \begin{cases} 1 & \text{if} \quad d(\overrightarrow{R}|_{v/2}^{v}, \overrightarrow{0}) > (v/2)/2 \\ 0 & \text{otherwise} \end{cases}$$

Since we mapped the vector into two columns, two intermediate activation values (out_{p1}, out_{p2}) are obtained. The final value depends on the "**cascading function**". This function can be a simple logical *AND* or *OR* function.

$$out(\overrightarrow{R}|_{v/2}^{v}, \overrightarrow{R}|_0^{v/2}) = out_{p2}(\overrightarrow{R}|_{v/2}^{v}) \wedge out_{p1}(\overrightarrow{R}|_0^{v/2})$$

In the case of logical *AND* as an example, the following conditions show the scenarios where the output of cascading function differs from the golden output. This is also illustrated in Figure 5. The axes are the hamming distance obtained from the result of the first and second parts of the output vector. The red and blues regions indicate where the *AND* and *OR* functions generate inaccurate results. Following are the conditions where the output of *AND* cascading function differs from the golden output.

$out(\overrightarrow{R}|_{v/2}^{v}, \overrightarrow{R}|_0^{v/2}) \neq out_{golden}(\overrightarrow{R})$ if:

$$\begin{cases} d(\overrightarrow{R}|_0^{v/2}, \overrightarrow{0}) + d(\overrightarrow{R}|_{v/2}^{v}, \overrightarrow{0}) > v/2 \\ d(\overrightarrow{R}|_0^{v/2}, \overrightarrow{0}) < v/4 \end{cases}$$
$$\vee$$
$$\begin{cases} d(\overrightarrow{R}|_0^{v/2}, \overrightarrow{0}) + d(\overrightarrow{R}|_{v/2}^{v}, \overrightarrow{0}) > v/2 \\ d(\overrightarrow{R}|_{v/2}^{v}, \overrightarrow{0}) < v/4 \end{cases}$$

The number of input vectors that fall in these regions (blue or red in Figure 5) are calculated based on Equation 5. According to this equation, Figure 6 depicts the maximum accuracy loss for two cascading functions considering two boundary

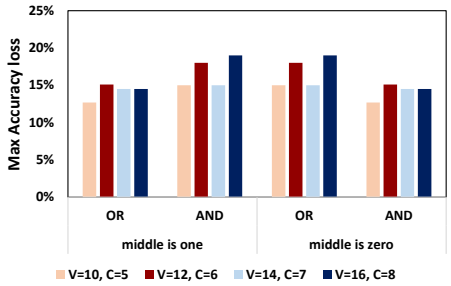

Figure 6: Maximum accuracy loss simulated for all possible input vectors for different vector sizes (V), crossbar size (C), and cascading functions

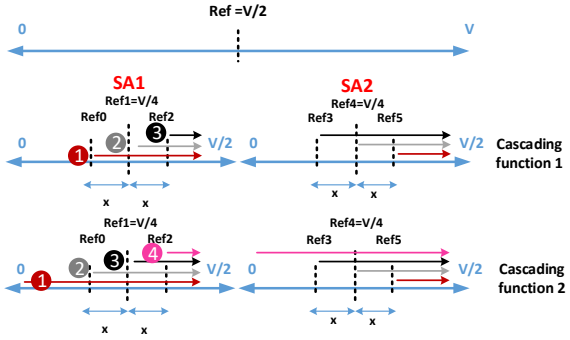

Figure 7: Illustration of two cascading functions where two auxiliary references added to the main reference

conditions. This is done by generating all the input sets to verify the Equation 5. We observe that the accuracy loss does not have considerable changes over vector sizes as the relative area associated to inaccurate region remains the same (Figure 5).

$$\forall (m,n) \in Solution\ Set: \\ \#(\overrightarrow{A},\overrightarrow{B}) = ({}_mC_{v/2} * 2^m * 2^{v/2-m}) * \\ ({}_nC_{v/2} * 2^n * 2^{v/2-n}) \quad (5)$$

To reduce the accuracy loss, more references can be considered. This leads to more intermediate results which provide us with more information as well as more flexibility to have more advanced cascading functions. However, we should take into account that keeps adding references increases the hardware complexity of SA. In the following, we investigate the implication of the number of references as well as their actual values on accuracy loss. Figure 7 presents an example where two auxiliary references are added to the main reference. In this scenario, three intermediate values are produced for each of the output vectors and the final activation value should be decided based on them. We illustrate two possible cascading functions in this figure. The first function

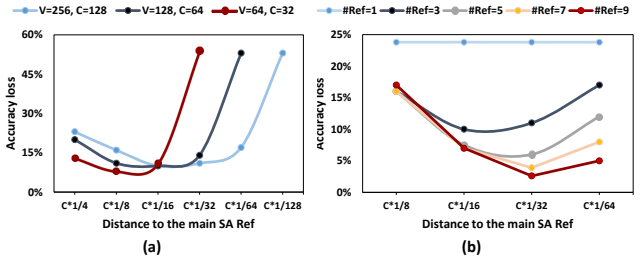

Figure 8: (a) Accuracy loss based on the distance of two auxiliary references to the main reference (b) effect of number of auxiliary references on accuracy

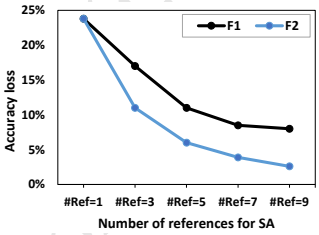

Figure 9: Impact of the two cascading functions (illustrated in Figure 7) on accuracy loss

comprises three conditions, where meeting each, can set the final activation value to one. These are based on the fact that the summation of two hamming distances obtained from two output vectors should be greater than half of the original vector size (Equation 4). This function always set the activation value to one accurately (true positive), while it misses to set it to one in some cases (false negative). Considering that, the second cascading function makes the conditions more relaxed. The probability of accuracy loss for these two functions is computed in the following.

$P_{Loss}(F1) =$
$\mathbf{P}([SA_{out2} > Ref5 \wedge SA_{out1} < Ref0] \wedge [SA_{out2} + SA_{out1} > v/2]) + \mathbf{P}([SA_{out2} < Ref3 \wedge SA_{out1} > Ref2] \wedge [SA_{out2} + SA_{out1} > v/2]) + \mathbf{P}([Ref4 < SA_{out2} < Ref5] \wedge [Ref0 < SA_{out1} < Ref1] \wedge [SA_{out1} + SA_{out2} > v/2]) + \mathbf{P}([Ref3 < SA_{out2} < Ref4] \wedge [Ref1 < SA_{out1} < Ref2] \wedge [SA_{out1} + SA_{out2} > v/2])$

$P_{Loss}(F2) =$
$\mathbf{P}([SA_{out2} > Ref5] \wedge [SA_{out2} + SA_{out1} < v/2]) + \mathbf{P}([SA_{out2} > Ref5] \wedge [SA_{out2} + SA_{out1} < v/2]) + \mathbf{P}([Ref4 > SA_{out2} > Ref3] \wedge [Ref2 > SA_{out1} > Ref1] \wedge [SA_{out2} + SA_{out1} < v/2]) + \mathbf{P}([Ref5 > SA_{out2} > Ref4] \wedge [Ref1 > SA_{out1} > Ref0] \wedge [SA_{out2} + SA_{out1} < v/2])$

An important parameter that has a remarkable impact on the accuracy loss is the distance of auxiliary references to the main reference ("x" in Figure 7). This is quite dependent on the distribution of data. Hence, the designer can analyze the network and based on that find the proper value for

Table 1: Typologies of the BNNs and their software accuracy

Name	Topology	Dataset	Accuracy
LeNet-5	5x5,6 - 2x2 Pool - 5x5,16 - 2x2 Pool - FC(120) - FC(84) - FC(10)	MNIST	%98
CNN-1	5x5,5 - 2x2 Pool - FC(720) - FC(70) - FC(10)	MNIST	%97
CNN-2	7x7,10 - 2x2 Pool - FC(1210) - FC(1210) - FC(10)	MNIST	%98
MLP-S	FC(784) - FC(500) - FC(250) - FC(10)	MNIST	%97
MLP-M	FC(784) - FC(1000) - FC(500) - FC(250) - FC(10)	MNIST	%98.2
MLP-L	FC(784) - FC(1500) - FC(1000) - FC(500) - FC(10)	MNIST	%98.4

the references where the accuracy loss is minimized. Figure 8(a) demonstrates the impact of this parameter for cascading function 2 assuming normal distribution. This is presented for different crossbar sizes ("c"). The distance to the main reference is shown relative to the crossbar size. The figure indicates the importance of the values for the references and how considerably they can change the accuracy loss. Another important parameter is the number of references. The implication on accuracy can be comprehended from Figure 8(b). It is observed that by keep adding more references, an improvement in accuracy is reduced while more complexity is added to the hardware. Finally, the impact of the cascading functions on accuracy is evaluated in Figure 9 over a different number of references. The same two methods presented in Figure 7 are also used for the situation where we have more than three references. The figure indicates that choosing a proper function can help the accuracy of the system remarkably.

6 EVALUATION

6.1 Simulation setup

Our simulation results are obtained by creating our PyTorch-based platform. This platform is able to evaluate the accuracy, energy, and latency of different networks containing binarized and non-binarized layers. The software is written in a modular way to flexibly change network structure as well as different circuit-level parameters. The system runs at a clock frequency of 1GHz. The data bus between the crossbars has 32-bit width. Based on the 32nm technology node, transferring data to store it in an input buffer consumes 5mW [32, 33]. The energy and latency number of the "Shift and Add" unit required for non-binarized layers taken from [33]. In all the simulations, the crossbar size is 512×512 [34]. We use an analytical model based on a small PCM prototype and extend the memory to the required size. The model is acquired from the results of the EU project MNEMOSENE [35]. Finally, the specification of ADC is taken from [36].

Our benchmark (MlBench) comprises 6 BNNs for machine learning applications. The structure of each network is listed in Table 1. LeNet-5, CNN-1, and CNN-2 are convolutional networks, and MLP-S/M/L are multilayer perceptrons (MLPs) with different network scales [37]. These networks are evaluated on the widely used MNIST database of handwritten digits. We compare our design with a recent work published in one of the leading journals in this field [24]. For this work, we instantiate the digital post-processing units (popcount) for every 16 columns of the crossbar instead of sequentially operating over all the columns (see Figure 3(a)). This diminishes the latency overhead of digital processing for the baseline.

6.2 Results

Accuracy analysis

Figure 10 depicts the accuracy loss using our proposed approach compared to the software implementation. The figure presents the results for all the benchmarks considering two different cascading functions (see Figure 7). Since the size of each layer in LeNet-5 network is smaller or in the range of crossbar size, no accuracy loss is observed. However, this is not the case for the rest of the networks. The results show the importance of cascading function. As calculated in 5, "F2" is superior than "F1" due to less noise injection per layer. However, the difference depends on the network structure and distribution of data.

Other important parameters which can have a remarkable impact on accuracy are the number of references and their distance from each other. We ran the simulation for *CNN1* and *CNN2* networks with 3 references. As expected and can be seen in Figure 10, the presence of two auxiliary references helps to generate less inaccuracy in the network. Besides, Figure 11 depicts the consequence of distance between main reference and auxiliary references ("x" in Figure 7). The distance is relative to the crossbar size ("C"). Placing the references far from or close to each other reduces their efficiency in eliminating the cases where inaccurate activation values are generated. Therefore, the designer should find the optimal value for the references.

Energy and latency analysis

Figure 12 depicts the energy improvement and the contribution of layers in total energy consumption considering two networks. The result shows that up to 10× improvement is achieved compared to the baseline. Energy improvement

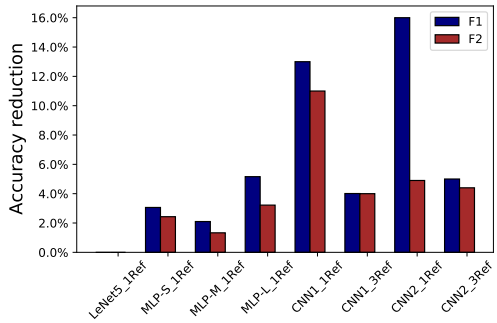

Figure 10: Accuracy reduction for different network structures due to the crossbar size limitation and breaking the vectors over more crossbars

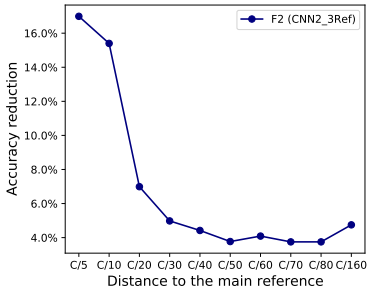

Figure 11: Impact of auxiliary references and their distance from the main reference on accuracy loss

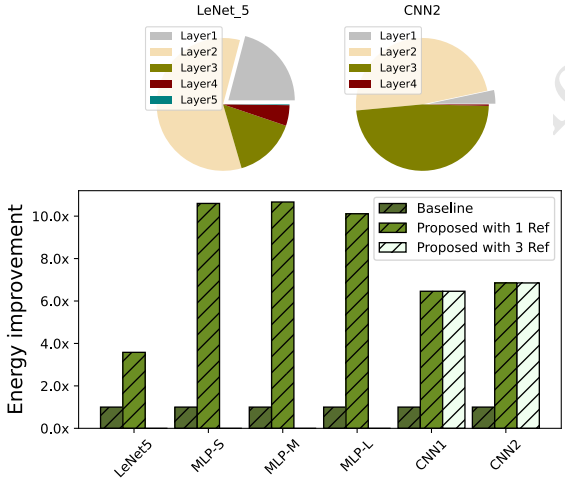

Figure 12: Energy improvement compared to the baseline and break down of energy for different layers of two networks

mainly is eventuated from less crossbar activation. In convolutional networks, since the first layer, which is not binarized, has the most contribution to the total energy, less improvement is obtained. In addition, a SA with three references requires three cycles to generate the output which

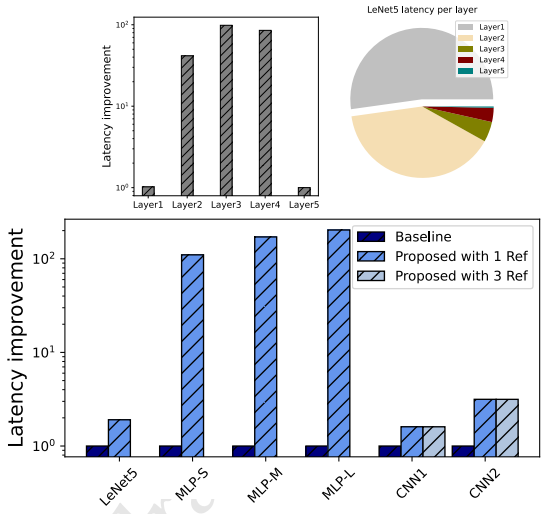

Figure 13: Latency improvement compared to the baseline and its break down for different layers of LeNet-5 network

leads to approximately three times more energy consumption. However, the impact on the total energy of the network is negligible due to the small contribution of SAs.

Figure 13 shows the latency improvement of the entire network. The remarkable improvement is obtained mainly due to computing each activation value in a non-sequential manner as well as computing the activation values among different output channels in parallel. Similar to the energy number, the improvement for convectional networks is less due to the large contribution of the first layer to the total latency of the network. Therefore, as a solution to reduce the overhead of this layer on the network, a designer may allocate more resources for this layer to compute the activation values for different operating windows in parallel. However, in our simulation, we consider as minimum as possible resources for each layer.

7 CONCLUSION AND FUTURE DIRECTION

This paper proposed a novel in-memory memristor-based design that substantially improves the performance and energy efficiency of BNN applications. The proposed XNOR-based BNN design, replace the functionality of ADC and post-processing with a SA while maximizing parallelization and resource utilization in the design with a novel mapping of weights and activation values in the crossbar and its input buffer. The design can outperform the baseline specifically in intermediate layers. On average this work is able to yield 8× and 60× higher energy and performance than the baseline. In our future work, we consider the impact of variability in references on the accuracy of the network. Besides, we evaluate the design for larger and more complex networks to comprehend the impact of inaccuracy injected into intermediate layers on the final accuracy of the networks.

REFERENCES

[1] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 2, pp. 604–624, 2020.

[2] A. Dhillon and G. K. Verma, "Convolutional neural network: a review of models, methodologies and applications to object detection," *Progress in Artificial Intelligence*, vol. 9, no. 2, pp. 85–112, 2020.

[3] W. Wang, Y. Yang, X. Wang, W. Wang, and J. Li, "Development of convolutional neural network and its application in image classification: a survey," *Optical Engineering*, vol. 58, no. 4, p. 040901, 2019.

[4] D. Lu and Q. Weng, "A survey of image classification methods and techniques for improving classification performance," *International journal of Remote sensing*, vol. 28, no. 5, pp. 823–870, 2007.

[5] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognition*, vol. 105, p. 107281, 2020.

[6] C. Yuan and S. S. Agaian, "A comprehensive review of binary neural network," *arXiv preprint arXiv:2110.06804*, 2021.

[7] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[8] Q. Hu, P. Wang, and J. Cheng, "From hashing to cnns: Training binary weight networks via hashing," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[9] J. Faraone, N. Fraser, M. Blott, and P. H. Leong, "Syq: Learning symmetric quantization for efficient deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4300–4309.

[10] B. Martinez, J. Yang, A. Bulat, and G. Tzimiropoulos, "Training binary neural networks with real-to-binary convolutions," *arXiv preprint arXiv:2003.11535*, 2020.

[11] L. Hou, Q. Yao, and J. T. Kwok, "Loss-aware binarization of deep networks," *arXiv preprint arXiv:1611.01600*, 2016.

[12] C. Liu, W. Ding, X. Xia, B. Zhang, J. Gu, J. Liu, R. Ji, and D. Doermann, "Circulant binary convolutional networks: Enhancing the performance of 1-bit dcnns with circulant back propagation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2691–2699.

[13] S. Darabi, M. Belbahri, M. Courbariaux, and V. P. Nia, "Bnn+: Improved binary network training," 2018.

[14] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "Fp-bnn: Binarized neural network on fpga," *Neurocomputing*, vol. 275, pp. 1072–1086, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231217315655

[15] C. Fu, S. Zhu, H. Su, C.-E. Lee, and J. Zhao, "Towards fast and energy-efficient binarized neural network inference on fpga," *arXiv preprint arXiv:1810.02068*, 2018.

[16] H. Yang, M. Fritzsche, C. Bartz, and C. Meinel, "Bmxnet: An open-source binary neural network implementation based on mxnet," in *Proceedings of the 25th ACM international conference on Multimedia*, 2017, pp. 1209–1212.

[17] S. Hamdioui, H. A. Du Nguyen, M. Taouil, A. Sebastian, M. L. Gallo, S. Pande, S. Schaafsma, F. Catthoor, S. Das, F. G. Redondo, G. Karunaratne, A. Rahimi, and L. Benini, "Applications of computation-in-memory architectures based on memristive devices," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 486–491.

[18] J. Chen, S. Wen, K. Shi, and Y. Yang, "Highly parallelized memristive binary neural network," *Neural Networks*, vol. 144, pp. 565–572, 2021.

[19] T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, "Binary convolutional neural network on rram," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 782–787.

[20] L. Huang, J. Diao, H. Nie, W. Wang, Z. Li, Q. Li, and H. Liu, "Memristor based binary convolutional neural network architecture with configurable neurons," *Frontiers in neuroscience*, vol. 15, p. 328, 2021.

[21] Y.-F. Qin, R. Kuang, X.-D. Huang, Y. Li, J. Chen, and X.-S. Miao, "Design of high robustness bnn inference accelerator based on binary memristors," *IEEE Transactions on Electron Devices*, vol. 67, no. 8, pp. 3435–3441, 2020.

[22] Y. Zhao, Y. Wang, R. Wang, Y. Rong, and X. Jiang, "A highly robust binary neural network inference accelerator based on binary memristors," *Electronics*, vol. 10, no. 21, p. 2600, 2021.

[23] Y. Halawani, B. Mohammad, M. A. Lebdeh, M. Al-Qutayri, and S. F. Al-Sarawi, "Reram-based in-memory computing for search engine and neural network applications," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 388–397, 2019.

[24] T. Hirtzlin, M. Bocquet, B. Penkovsky, J.-O. Klein, E. Nowak, E. Vianello, J.-M. Portal, and D. Querlioz, "Digital biologically plausible implementation of binarized neural networks with differential hafnium oxide resistive memory arrays," *Frontiers in neuroscience*, vol. 13, p. 1383, 2020.

[25] O. Golonzka, U. Arslan, P. Bai, M. Bohr, O. Baykan, Y. Chang, A. Chaudhari, A. Chen, J. Clarke, C. Connor *et al.*, "Non-volatile rram embedded into 22ffl finfet technology," in *2019 Symposium on VLSI Technology*. IEEE, 2019, pp. T230–T231.

[26] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnology*, pp. 1–16, 2020.

[27] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems*, vol. 28, 2015.

[28] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang *et al.*, "Memristor-based analog computation and neural network classification with a dot product engine," *Advanced Materials*, vol. 30, no. 9, p. 1705914, 2018.

[29] Y. Kim, W. H. Jeong, S. B. Tran, H. C. Woo, J. Kim, C. S. Hwang, K.-S. Min, and B. J. Choi, "Memristor crossbar array for binarized neural networks," *AIP Advances*, vol. 9, no. 4, p. 045131, 2019.

[30] D. Ahn, H. Oh, H. Kim, Y. Kim, and J.-J. Kim, "Maximizing parallel activation of word-lines in mram-based binary neural network accelerators," *IEEE Access*, vol. 9, pp. 141 961–141 969, 2021.

[31] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[32] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-m. W. Hwu, J. P. Strachan, K. Roy *et al.*, "Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 715–731.

[33] M. Zahedi, R. van Duijnen, S. Wong, and S. Hamdioui, "Tile Architecture and Hardware Implementation for Computation-in-Memory," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2021, pp. 108–113.

[34] P. Narayanan, S. Ambrogio, A. Okazaki, K. Hosokawa, H. Tsai, A. Nomura, T. Yasuda, C. Mackin, S. Lewis, A. Friz *et al.*, "Fully on-chip mac at 14nm enabled by accurate row-wise programming of pcm-based weights and parallel vector-transport in duration-format," in *2021 Symposium on VLSI Technology*. IEEE, 2021, pp. 1–2.

[35] "Mnemosene project," http://www.mnemosene.eu, accessed: 2010-09-30.

[36] G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-memory hyperdimensional computing," *Nature Electronics*, vol. 3, no. 6, pp. 327–337, 2020.

[37] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 27–39.