TWINN: Training-Free Weight-Input Flipping for Mitigating Crossbar Non-Idealities in Binary Neural Network Accelerators

Akul Malhotra and Sumeet Kumar Gupta

***Abstract*— Compute-in-memory (CiM)-based binary neural network (CiM-BNN) accelerators marry the benefits of CiM and ultra-low precision quantization, making them highly suitable for edge computing. However, CiM-enabled crossbar (Xbar) arrays are plagued with hardware non-idealities like parasitic resistances and device non-linearities that impair inference accuracy, especially in scaled technologies. In this work, we first analyze the impact of Xbar non-idealities on the inference accuracy of various CiM-BNNs, establishing that the unique properties of CiM-BNNs make them more prone to hardware non-idealities compared to higher precision deep neural networks (DNNs). To address this issue, we propose TWINN, a training-free technique that mitigates non-idealities in CiM-BNNs. TWINN utilizes the distinct attributes of BNNs to reduce the average current generated during the CiM operations in Xbar arrays. This is achieved by statically and dynamically flipping the BNN weights and activations, respectively. This minimizes the IR drops across the parasitic resistances, drastically mitigating their impact on inference accuracy. To evaluate our technique, we conduct experiments on ResNet-18 and VGG-small CiM-BNNs designed at the 7nm technology node using 8T-SRAM and 1T-1ReRAM. Our results show that TWINN is highly effective in alleviating the impact of non-idealities, recouping the inference accuracy to near-ideal (software) levels in some cases and providing accuracy boost of up to 77.25%. These benefits are accompanied by energy reduction, albeit at the cost of mild latency/area increase.**

***Index Terms*— Binary neural networks, computing-in-memory, hardware non-idealities, IR drop, technology scaling.**

I. Introduction

RECENTLY, there has been an immense interest in design techniques that can enhance the energy efficiency of deep neural networks (DNNs) and enable their deployment on the edge [1]. From the algorithmic side, reducing the bit precision of DNN parameters via quantization lowers their energy, latency and storage demands, with minimal impact on accuracy [2]. Binary neural networks (BNNs) represent an extreme form of quantization wherein 1-bit weights and activations ($\in \{-1, +1\}$) are utilized, drastically improving the energy/area efficiencies [3].

Received 9 December 2024; revised 24 April 2025; accepted 7 May 2025. This work was supported in part by the Center for the Co-Design of Cognitive Systems (COCOSYS), one of Seven Centers in Joint University Microelectronics Program (JUMP) 2.0, funded by Semiconductor Research Corporation (SRC) and Defense Advanced Research Projects Agency (DARPA), Raytheon, and NSF. This article was recommended by Associate Editor X. Zhang. *(Corresponding author: Akul Malhotra.)*

The authors are with the Elmore Family School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: malhot23@purdue.edu).

Digital Object Identifier 10.1109/TCSI.2025.3570193

From the hardware perspective, computing-in-memory (CiM), in which operations such as vector-matrix multiplications (VMMs) are performed within a crossbar (Xbar) memory array, is a promising direction [4]. CiM alleviates the massive data movement costs that plague standard von-Neumann-based DNN accelerators, leading to large energy and latency reduction. Another hardware aspect that is particularly important to meet the needs of growing DNN model sizes is technology scaling, which reduces the area and energy consumption, facilitating further efficiency gains [5].

Utilizing binary quantization in conjunction with CiM combines the benefits of both the techniques and is highly suitable for edge computing. Several CiM-based BNN hardware designs (CiM-BNNs) have been developed utilizing CMOS as well as various non-volatile memory (NVM) technologies, showcasing substantial energy-latency-area benefits [6], [7], [8].

Although CiM offers a significant energy/latency benefits, Xbar non-idealities due to wire resistance, driver/sink resistance and device non-linearities afflict the computational robustness [16], [17], [18], leading to inaccurate VMM computations and degraded DNN accuracy. This issue is even more concerning in deeply scaled technologies, due to an increase in the wire resistivity and resistance [19]. To address the issue of Xbar non-idealities, various technological and design solutions have been proposed [9], [10], [12], [13], [14], [20], [21]. On the technology side, new interconnect materials/processes could potentially mitigate the Xbar non-idealities, but need more investigation [19]. At the circuit/algorithmic levels, various techniques are being explored to improve CiM robustness, but most of them incur high training/finetuning costs [9], [14], [20] or a large performance penalty [22]. More importantly, most of these works focus on higher-precision DNNs, leaving the analysis of Xbar non-idealities in CiM-BNNs largely unexplored.

In this work, we demonstrate that the impact of parasitic resistances on CiM robustness is significantly more pronounced in binary neural networks (BNNs) compared to high-precision DNNs, primarily due to the inherent characteristics of BNNs–an effect that is further intensified in CiM-BNNs implemented using the NAND-Net architecture (details later). Thus, to harness the benefits of CiM-BNNs, especially in scaled technologies, there is a pressing need to develop techniques that can pointedly mitigate this issue.

To that end, we propose TWINN, a training-free technique that alleviates the impact of Xbar non-idealities on CiM-BNN accuracy. TWINN reduces the magnitude of the CiM outputs of Xbar arrays by statically and dynamically flipping the BNN weights and activations, respectively. The flips are strategically selected to reduce the average output current of the crossbar, thereby minimizing IR drops across the parasitic resistances. The key contributions of our work include:

- We show that the unique attributes of CiM-BNNs make them highly prone to Xbar non-idealities, resulting in huge accuracy loss, especially in scaled technologies.
- We propose TWINN, a training-free non-ideality-mitigating technique tailored for CiM-BNNs.
- We demonstrate the effectiveness of TWINN by applying it on 8T-SRAM and 1T-1ReRAM (resistive RAM) based CiM-BNNs designed in the 7nm technology node.
- We evaluate the hardware implications of TWINN at the macro level, showing energy benefits accompanied with mild latency/area overheads.

The remainder of this paper is organized as follows. Section II presents the necessary background on CiM-BNNs and prior techniques to mitigate crossbar non-idealities. Section III describes the simulation framework developed to evaluate Xbar non-idealities and leverages it to analyze their effects on CiM-BNN inference accuracy. Section IV introduces our proposed solution, TWINN, detailing its static and dynamic flipping as well as hardware implementation. Section V presents experimental results, including partial-sum reduction, accuracy gains, hardware overheads, and comparisons with enhanced baseline configurations. Finally, Section VI concludes the paper.

II. Background and Related Works

A. CiM-Based BNN Hardware (CiM-BNNs)

CiM-BNNs derive their high energy efficiency by (i) restricting the weights and activations to 1 bit each with values of $+1$ or -1 and (ii) performing in-memory VMM. The signed binary representation maps the scalar multiplication between weights and activations to an XNOR operation. To enable XNOR-based CiM, previous works have introduced customized bitcells [6], [7], albeit at significant area and energy costs compared to standard bitcells. To avert these costs, recent works, such as NAND-Net [23], have shown that by applying linear transformations to activations and weights, the XNOR operation can be converted into an AND operation, which can be implemented using standard memory bitcells. These transformations map the original weights (W) and activations (I) to W' and I', respectively, translating from {'-1','1'} to {'0','1'} domain. These transformations are given by: $I = 2I' - 1$ and $W = 2W' - 1$. Hence, the new weights and activations are represented with high/low resistance state (HRS/LRS) of the memory and binary voltages, respectively, leading to seamless AND-CiM with standard memories. Using these transformations, the dot product can be written as:

$$\sum_{i=1}^{n} I_i W_i = 4\sum_{i=1}^{n} I'_i W'_i - 2\sum_{i=1}^{n} I'_i - 2\sum_{i=1}^{n} W'_i + n \qquad (1)$$

Here, n is the length of the W and I vectors. While $\sum_{i=1}^{n} I'_i W'_i$ can be efficiently computed with AND-CiM macros, some pre- and post-processing is needed to obtain $\sum_{i=1}^{n} I_i W_i$. First, a near-memory adder tree is needed to compute $\sum_{i=1}^{n} I'_i$. Note that since $\sum_{i=1}^{n} W'_i$ and n in (1) are fixed, they can be pre-computed and stored. Also, since an activation vector is shared by many weight vectors spread across multiple Xbar arrays, the cost of the adder tree is amortized [23]. Second, some arithmetic circuitry is needed to add/subtract the second, third and fourth terms in (1) from $\sum_{i=1}^{n} I'_i W'_i$ (CiM output). These additional circuitry generally incur much lower cost compared to customized bitcells. Given the multiple benefits of NAND-Net over customized bit-cells, in this work, we focus on NAND-Net based CiM-BNNs. Further details on the advantages of NAND-Net-based CiM-BNNs over XNOR-based CiM-BNNs can be found in [23].

B. Techniques to Mitigate Xbar Non-Idealities

While Xbar arrays seamlessly facilitate the CiM of VMM, they suffer from various non-idealities such as IR drops in the parasitic resistances, device non-linearities etc., which lead to CiM errors, impairing the inference accuracy.

Several previous works have explored the impact of Xbar non-idealities on DNN accuracy and have proposed techniques to alleviate their impact on CiM-based DNNs, red summarized in Table I. On the algorithmic side, *non-ideality aware* training has been explored [9], [14], [20]. Although non-ideality aware training is very effective in mitigating the impact of Xbar non-idealities, it requires access to large computational resources as well as labelled training data, which may not always be available. To bypass the prohibitive cost of training, some works utilize non-ideality aware finetuning to regain the accuracy lost due to non-idealities [10], [21]. However, these solutions still require access to labelled training data/data statistics, which can be challenging to obtain due to data privacy constraints. Additionally, training/fine-tuning methods will need to re-applied every time a technology parameter is changed, further complicating the hardware design cycle.

From the hardware perspective, novel write schemes, redundancy, non-ideality-aware weight mapping and partial wordline activation (PWA) are commonly used non-ideality-mitigating strategies [12], [13], [21]. For example, [11] and [21] use a redundant row per array to enhance CiM robustness, albeit at the cost of area [21]. Works based on non-ideality-aware mapping [12], [15] reduce IR drops via row/column re-arrangement but have applicability limited to only certain Xbar designs or memory technologies. PWA reduces the current in Xbar arrays, but at the cost of latency [22].

Furthermore, two important points should be noted. First, *most* of the aforementioned works have been evaluated for pre-45nm technology nodes and their effectiveness in deeply scaled technologies, where the parasitic resistance based non-idealities are aggravated, requires further study. Second, these works focus primarily on high-precision DNNs, leaving the impact of Xbar non-idealities in BNNs and their mitigation unaddressed. In this work, we fill this gap by evaluating the

TABLE I
COMPARISON OF PREVIOUS WORKS

Criteria	[9]	[10]	[11]	[12]	[13]	[14]	[15]	This Work
No training needed	✗	✓	✓	✓	✓	✗	✓	✓
No training data/statistics needed	✗	✗	✓	✗	✓	✗	✓	✓
Technology node used	-	45nm	-	65nm	40nm	45nm	7nm	7nm
Studies BNNs	✓	✗	✗	✗	✗	✗	✗	✓
Memory technology agnostic	✓	✓	✓	✗	✓	✗	✗	✓

Xbar non-idealities in CiM-BNNs at the 7nm technology node and proposing TWINN to enhance CiM-BNN accuracy.

III. BNNs AND XBAR NON-IDEALITIES

In this section, we describe the issue of aggravated Xbar non-idealities in CiM-BNNs that stems from the distinct attributes of BNNs. For this, we take ResNet-18 BNN as a case study, and analyze the Xbar outputs, current deviations due to Xbar non-idealities and their impact on BNN accuracy.

A. Partial Sum Analysis for CiM-BNNs

We start by profiling the expected CiM output of the Xbar arrays ($\sum_{i=1}^{n} I * W$), which we refer to as the *ideal* partial sums. Note, DNN weight matrices are typically partitioned and mapped onto multiple Xbars. The partial-sums produced by these Xbars are combined to obtain the VMM output.

For our partial sum analysis, we map a ResNet-18 BNN trained on CIFAR-10 on Nand-Net-based 64 × 64 size Xbar arrays. For comparison, we also map a ResNet-18 4-bit DNN on 64 × 64 Xbar arrays. For this, we store the 4-bit weights in their 2's-complement form with each of their bits stored in a binary memory element (to be compatible with designs such as SRAMs). The 4-bit activations (which are non-negative due to ReLU activation in high precision DNNs) are bit-streamed using binary voltages (0 and V_{DD}). We profile the distribution of the *ideal* partial sums (values ranging between 0 and 64) by performing inference using the entire CIFAR-10 test set.

It can be observed from the y-axes of Fig. 1 that the *total* number of partial-sum computations is significantly higher for the 4-bit W/A DNN compared to the BNN. This is due to its bit-sliced weight storage and bit-streamed activation inputs. Specifically, the number of partial-sums is 16× higher in the 4-bit DNN, as each dot product requires 4 cycles for activation streaming and 4 partial-sum computations per cycle for the weight bits.

Our analysis (Fig. 1) also shows that BNNs produce substantially larger partial-sums on *average* than the 4-bit DNNs. The average 4-bit DNN partial-sum is ~ 4, whereas that for the BNN is ~ 19, 4.6X larger. We attribute this to two distinct properties of BNNs. First, due to *only two possible values* ($+1$ and -1) of weights and activations, the distribution of 1's and -1's (0's in Xbars) in BNNs is more or less uniform. This is also true for other BNNs used in our work (Fig. 6). In contrast, the higher precision DNNs have a zero state and exhibit an approximately normal distribution of DNN weights (and in some cases, activations) centered around that zero state.

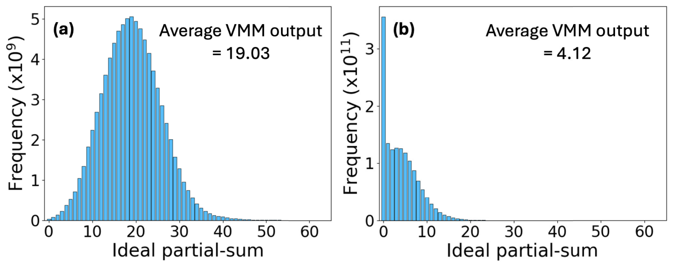

Fig. 1. Histograms showing the frequency of ideal partial-sums in a (a) ResNet-18 BNN and (b) ResNet-18 with 4-bit weights and activations. The Xbar array size is 64 × 64. We observe that the BNN produces larger partial-sums on average than the 4-bit DNN.

Additionally, quantization, even to standard 4-8 bits, collapses close-to-0 values to 0. As a result, high precision DNNs exhibit larger input/weight sparsity compared to BNNs. Bit-streaming of activations, combined with the ReLU activation function, enhances sparsity in higher-precision DNNs. However, this benefit does not extend to BNNs, as their activations are limited to a single bit. Furthermore, BNNs typically employ Tanh or HardTanh activation functions, which do not increase the number of zeros in activations as ReLU does. These two distinctions between BNNs and higher precision DNNs culminate in substantially higher partial-sums in CiM-BNNs. It is worth noting that while our focus is on convolutional DNNs, other architectures, such as transformers and LLMs, also leverage sparsity-enhancing activation functions like GeLU [24]. As a result, these architectures are similarly expected to exhibit reduced partial sums.

The inherently high partial sums in BNNs imply large currents in the Xbar columns, which are expected to severely worsen the hardware non-idealities (such as IR drops). To understand this, we evaluate the Xbar non-idealities and their impact on inference accuracy of ResNet-18 BNN. Before we discuss this, let us describe our evaluation framework.

B. Simulation Framework for Evaluating Xbar Non-Idealities

Our simulation framework is based on a customized Xbar simulator, which exactly and self-consistently solves the Kirchoff's current/voltage laws in conjunction with the memory device models to produce the output currents of the Xbar. Fig. 3(a) shows a column of the Xbar array, which our simulator models and Fig. 2 details the functioning of our simulator. Our framework captures non-idealities such as wire resistance (R_{wire}), driver resistance (R_{driver}), sink resistance (R_{sink}) and device non-linearities. In this work, we utilize the design where the activations (I) are applied on the gates of the

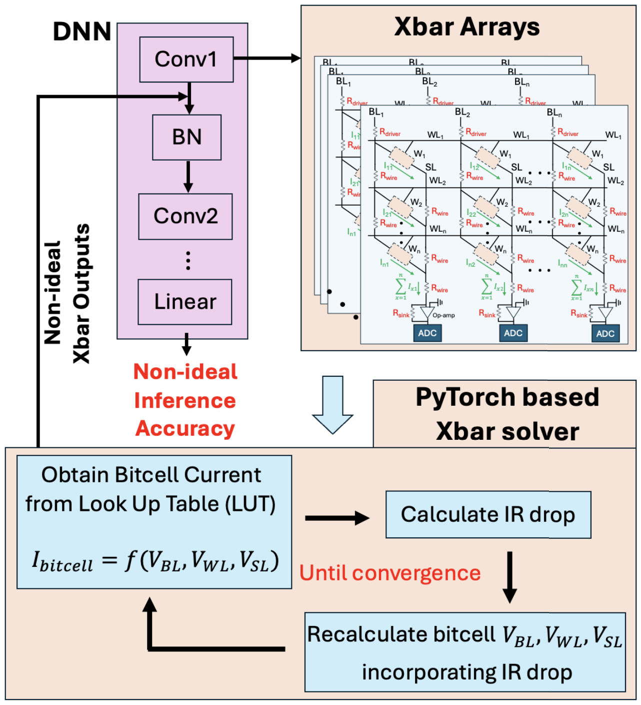
DNN
Conv1
BN
Conv2
Linear
Non-ideal Xbar Outputs
Non-ideal Inference Accuracy
Xbar Arrays
PyTorch based Xbar solver
Obtain Bitcell Current from Look Up Table (LUT)
$I_{bitcell} = f(V_{BL}, V_{WL}, V_{SL})$
Calculate IR drop
Until convergence
Recalculate bitcell V_{BL}, V_{WL}, V_{SL} incorporating IR drop

Fig. 2. The custom Xbar simulator used in this work.

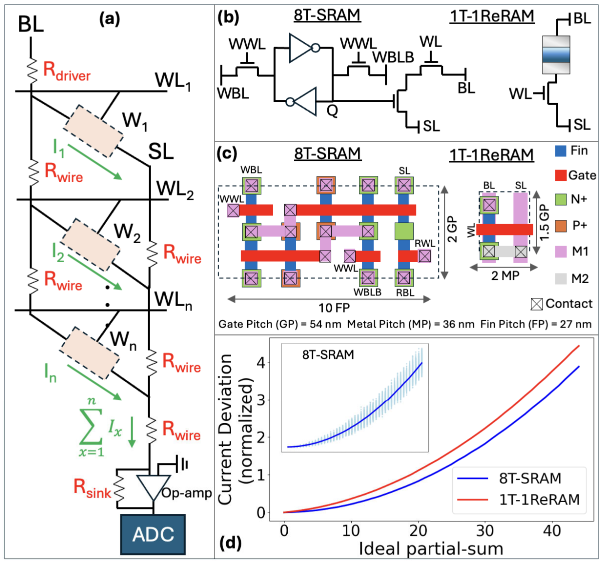

Fig. 3. (a) Xbar array column with non-idealities. (b-c) schematics and layout of 8T-SRAM and 1T-1ReRAM bitcells. (d) average current deviation (normalized to current quantum between adjacent ADC levels) versus partial-sum. The inset shows the range of current deviations for each partial sum corresponding to different input-weight combinations. Current deviation increases superlinearly with rising partial-sum.

access transistors (as it has been shown to be more resilient to hardware non-idealities than other design options [25]). As a result, there is negligible steady-state IR drop along the row. Hence, we can analyze the impact of non-idealities on each Xbar column independently [26]. Note that we use an op-amp to sense the output current before feeding it into an analog-to-digital converter (ADC), mitigating the effects of sink resistance, as proposed in [27]. Such choices enable us to analyze baseline BNNs designed for improved CiM robustness. We show that despite these choices, the inherently high partial sum leads to unacceptably low accuracy in BNNs.

In this work, we analyze Xbars using two memory technologies: 8T-SRAM and 1T-1ReRAM (Fig. 3(b) and (c)). We use 7nm predictive models [28] for the transistors and the experimentally calibrated model in [29] for ReRAMs. For wire resistivity (both line metals and vias), we use models (and their validated parameters) from [30], which capture surface scattering, grain-boundary scattering, and the effect of liner/barrier layers. For an optimistic baseline, we choose interconnects with scaled liners (lower resistance than standard). The wire resistance per bitcell is calculated by multiplying the bitcell height (Fig. 3(c)) with the resistance per unit length of the bitline (BL) and sense-line (SL).

By virtue of the models discussed above, our framework seamlessly captures the non-idealities due to finite current I_{HRS} and I_{OFF} produced by bitcells when $I = 1$ and $W = 0$ and $I = 0$ and $W = 0/1$, respectively. Note that for 8T-SRAM, the ratio of I_{ON} (current produced when $I = W = 1$) to $I_{OFF/HRS}$ is large ($> 10^4$). However, for 1T-1ReRAM, $\frac{I_{ON}}{I_{HRS}}$ is in the range of 10-50 [26], leading to CiM errors under certain scenarios. To mitigate the impact of I_{HRS} in 1T-1ReRAM Xbar, we use a dummy column, as in [26] and [31].

Our simulator (Fig 2) calculates the non-ideal Xbar column (Fig 3(a)) currents as follows: at each iteration, the simulator first calculates the terminal voltages–word-line (WL - V_{WL}), bit-line (BL- V_{BL}), and sense-line (SL - V_{SL})–for each bitcell in the Xbar array (starting from the ideal voltages in the first iteration). It then uses these voltages to look up corresponding bitcell currents from precomputed lookup tables (LUTs). The LUTs are obtained from SPICE simulations of the bit-cells by sweeping $V_{WL,SL}$ and $V_{BL,SL}$ (where $V_{a,b} = V_a - V_b$) and obtaining the corresponding bit-cell current. This use of LUTs enables our simulator to flexibly capture both linear and non-linear current-voltage behavior, thereby supporting a broader range of memory technologies. Next, these current values are used to recalculate terminal voltages, refining their estimates by incorporating IR drop across the parasitic resistances. This voltage-current update loop repeats iteratively until convergence (defined by negligible differences in bitcell currents between successive iterations) is achieved. Thus, the simulator rigorously calculates the non-ideal output current produced by each Xbar column.

We implement our simulator in Pytorch, facilitating its seamless and efficient integration into DNN workloads. The customized simulator has been extensively validated with SPICE simulations of Xbar arrays, showing a *maximum* error of 0.3%.

This general rigorous modeling capability distinguishes our framework from prior works like [14], (which assume ohmic bitcells) or [32] and [33] (which is based on approximate non-ideal Xbar model).

C. Impact of Large Partial-Sums on Xbar Non-Idealities

In this subsection, we evaluate 1) the current deviation (the difference between ideal and non-ideal output current) as a function of expected partial sum output and 2) the non-ideal inference accuracies for both the ResNet-18 CiM-BNN and the 4-bit DNN (discussed in Section III-A). For this, we utilize the simulation framework described above. We substantiate our findings with an approximate analytical study establishing the relationship between partial-sum and current deviation.

For the analysis in this section, we choose the design points (such as routing metal layer M6 for BL/SL, 1μA ON current etc. - more details later) that minimize the Xbar non-idealities. In Section V, we will carry out a more comprehensive discussion for different design points. For *each* ideal partial-sum x, we obtain the non-ideal currents from our simulator for 10,000 unique weight-activation pairs. It can be seen in Fig. 3(d) that the *average* current deviation for both memory technologies not only increases with an increase in x, but grows at a *superlinear* rate. As x increases, there are more 'ON' bitcells in a column, leading to higher BL/SL current and larger IR drops.

As discussed in Section III-A, BNNs inherently produce higher partial sums (average value ~19) compared to higher-precision DNNs. This results in significantly more severe current deviations due to the superlinear trend, causing a larger number of CiM errors and degraded inference accuracy. From Fig. 3(d), it is evident that increasing the partial sum from ~4 to ~19 leads to approximately an 18-fold and 10-fold rise in normalized current deviation for 8T-SRAM and 1R-1ReRAM, respectively. Fig. 4 shows the impact of Xbar non-idealities on both the ResNet-18 CiM-BNN and the 4-bit DNN accuracy on the CIFAR10 dataset. Our results show that the accuracy of the CiM-BNN designed with 8T-SRAM and 1T-1ReRAM Xbars drops from 88.50% (software accuracy) to 52.09% and 18.08%, respectively. In contrast, the accuracy of the 4-bit DNN only drops by 0.10% and 0.28% 8T-SRAM and 1T-1ReRAM, respectively. These drastic accuracy drops, due to the large partial sums in BNNs, deem the CiM-BNN unusable, especially in deeply scaled technologies. As discussed later, other design configurations yield even lower accuracy values. Keep in mind that we use binary memory elements for both the BNN and the 4-bit DNN (each 4-bit weight represented using four binary devices). This ensures compatibility with memory technologies like SRAM. However, an alternative approach using multi-level memory cells for the 4-bit DNN would reduce the number of Xbars required to store the DNN. However, this would also reduce current margins between adjacent output states, thereby increasing susceptibility to non-idealities [26].

Thus, while the accuracy of 4-bit DNNs are minimally affected by the Xbar non-idealities, BNN accuracies are significantly impacted due to much larger partial sums. To understand this phenomenon further, let us explore the relationship between partial-sum and current deviation (which leads to CiM error) for an Xbar column utilizing a simplified analytical approach. To illustrate our point, we consider an optimistic scenario by including the effect of the R_{driver} only i.e. neglecting R_{wire} and R_{sink}. (Note, the inclusion of these effects is expected to further worsen the current deviations). Additionally, to simplify our analysis, we neglect the current for the OFF bitcells (as the bit-cells considered in our work produce low OFF currents). Due to R_{driver}, each ON bitcell produces a current I instead of I_{ON}. For a partial-sum of n (column current is $n * I$), the current deviation is $n * (I_{ON} - I)$ and the IR drop across R_{driver} is:

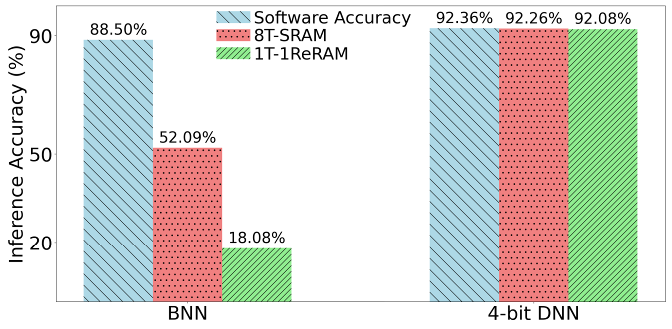

Fig. 4. Inference accuracies of the BNN and 4-bit DNN evaluated on the same Xbars.

$$\Delta V = R_{driver} * I * n \tag{2}$$

ΔV drop on the bitline is responsible for the reduction of bitcell current from I_{ON} to I. This can be written as:

$$\frac{\Delta I}{\Delta V} = \frac{I_{ON} - I}{\Delta V} = G \tag{3}$$

Since our analysis only considers R_{driver}, G here refers to the large-signal conductance of the bitcell with respect to BL. In general, G could be a function of the bias voltages (especially for non-ohmic entities). Using Eq. 2 and 3:

$$I_{ON} - I = n \cdot G \cdot I \cdot R_{driver} \tag{4}$$

$$I = \frac{I_{ON}}{1 + n \cdot G \cdot R_{driver}} \tag{5}$$

$$n \cdot (I_{ON} - I) = n \cdot I_{ON} \cdot \left(1 - \frac{1}{1 + n \cdot G \cdot R_{driver}}\right)$$
$$= \frac{n^2 \cdot I_{ON} \cdot G \cdot R_{driver}}{1 + n \cdot G \cdot R_{driver}} \tag{6}$$

Thus, we see that the current deviation is a *superlinear* function of the partial-sum n, with a linear asymptote. This phenomenon arises because each individual ON current contributes to an IR drop, which in turn influences all other ON currents, resulting in a quadratic behavior in the numerator. However, the IR drop simultaneously reduces the ON current, thereby diminishing the IR drop itself–essentially creating a negative feedback loop. This interplay is effectively captured by the denominator.

The analytical results further back the experimental results in Fig. 3(d). It is important to note that our analysis was an optimistic one, neglecting R_{wire} and R_{sink}, whose inclusion would further increase the impact of Xbar non-idealities. Additionally, R_{wire} brings a positional dependency to the current deviation, since different input-weight combinations corresponding to the same partial-sum will produce different current values, as shown in the inset of Fig. 3(d).

Thus, Xbar non-idealities are further aggravated in CiM-BNNs due to their large average partial-sums and the superlinear relation between partial sums and current deviation. It is important to emphasize that the robustness trends we observe in CiM-BNNs versus higher-precision DNNs are influenced by our specific architectural and technological design choices. This work adopts the NAND-Net-based CiM architecture, which leverages single-ended bitcells, and a deeply scaled 7nm technology node. These choices are motivated by their advantages in area, energy, and latency efficiency–making them particularly attractive for edge deployments. However, these choices also amplify the impact of Xbar non-idealities on CiM-BNNs.

For instance, the NAND-Net architecture enables the use of single-ended bitcells (e.g., 1T-1ReRAM) for BNNs, which reduces array footprint and peripheral overhead by requiring only one ADC per weight column. In contrast, differential bitcell configurations (e.g., 2T-2ReRAM) inherently consume more area and require two ADCs per weight column but offer improved tolerance to non-idealities due to common-mode noise cancellation [34], [35].

Additionally, the use of the 7nm technology node introduces pronounced interconnect non-idealities due to increased resistivity of copper wires. As scaling progresses, the barrier and liner layers required to mitigate reliability issues such as electromigration do not scale proportionally, leading to a reduced effective conduction area. Combined with enhanced sidewall electron scattering, this results in significantly higher wire resistivity and, consequently, more severe IR drops [19], [36]. Since our analysis shows that BNNs are particularly sensitive to IR drop due to their large average partial sums, the 7nm node presents an especially demanding challenge.

Thus, the aim of this work is to be able to get the best of both worlds: to benefit from the resource efficiency of the NAND-Net architecture as well the the 7nm technology node without compromising on robustness against Xbar non-idealities. In the next section, we propose TWINN, a training-free technique that tackles this issue.

IV. TWINN: The Proposed Solution

TWINN reduces the partial sums in BNN Xbar arrays by simultaneously employing two (twin) operations, namely static weight flipping and dynamic activation flipping. Both these operations capitalize on the *binary-valued* weights and activations in BNNs that facilitate some partial-sum reducing transformations while ensuring no effect on the VMM functionality.

A. Static Weight and Dynamic Activation Flipping

Recall, in the NAND-Net architecture (described in Section II-A), -1 and $+1$ weights are mapped to 0 (HRS) and 1 (LRS), respectively in the Xbar arrays. Similarly, -1 and $+1$ activations are mapped to 0 and V_{DD}, respectively. Therefore, if the number of -1's in the weight matrix or activation vectors are increased, it would imply increasing the zero weight bits stored in the Xbar or the zero input activation voltages applied to the wordlines (WL). In this section, we define weight and activation sparsification in this context i.e. reducing the number of $+1$'s (or increasing the number of -1's) so that the Xbar array or the WL voltage vectors are sparsified. Such a weight/activation sparsification

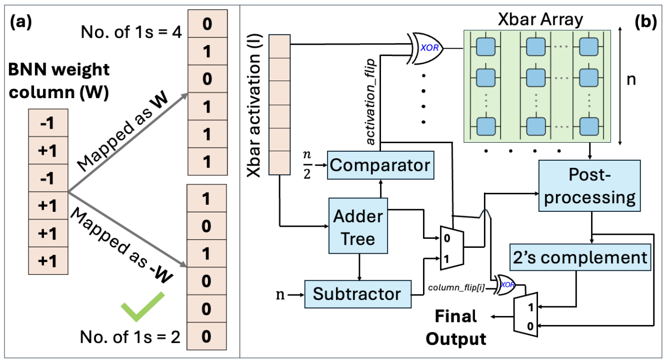

Fig. 5. (a) Static weight flipping being applied on a BNN weight column. (b) Hardware implementation of TWINN.

would naturally reduce the partial sums and reduce the impact of non-idealities.

Fig 5(a) illustrates the proposed static weight flipping operation utilized by TWINN for an Xbar column. The key role of this operation is to minimize the number of 1s in each column of the weight sub-matrix (W) *before* they are mapped onto an Xbar column. We achieve this by first calculating $\sum W$ and checking whether it is ≥ 0. If yes, then there are more $+1$s than -1s in W. In that case, we multiply all elements in W by -1 (flip). If $\sum W < 0$, we keep W unchanged. We also keep track of whether or not a flip has been performed in a column using a $column\_flip$ vector. If the i^{th} weight column is stored as $-W$ (W), $column\_flip[i]$ is equal to 1 (0). Once applied, the modified weights are deployed on the Xbar arrays and the $column\_flip$ vector is stored in a peripheral register. For an Xbar with n columns, $column\_flip$ is of size n bits. The $column\_flip$ vector is used to maintain the correct VMM functionality. If $column\_flip[i]$ is equal to 1, the corresponding VMM output of the column is multiplied by -1 (since $\sum I * W = -1 \cdot \sum I * (-W)$). Otherwise, the VMM output is used as it is.

Thus, by choosing between storing W or $-W$ based on $\sum W$, we enhance the sparsity in the Xbar weights. Note that we refer to the weight sparsification operation as *static*. This is because the BNN weights are constant during inference, enabling us to perform this step in advance (i.e. in software) rather than during inference.

Similar to the weight flipping, activation flipping is performed by selecting between activation sub-vectors I or $-I$ based on $\sum I$. Here, I is the sub-vector that is applied as an input to an Xbar array and has the size equal to the number of rows of the Xbar array (n). It is important to note that unlike weights, activations are dynamic and vary with each input, requiring flipping *during* inference. Thus, the selection between I or $-I$ must be performed in the hardware before the modified activation vectors can be applied as WL voltages. Recall, the NAND-Net hardware utilizes I in the [0,1] domain. Therefore, we compare $\sum I$ to $\frac{n}{2}$. If $\sum I > \frac{n}{2}$, there are more 1s than 0s (equivalent to -1s) in I, and we flip every element of I. Otherwise, I is unchanged. We name the output signal of the comparator $activation\_flip$. If an activation sub-vector is flipped, $activation\_flip$ is 1; else it is 0. $activation\_flip$ is utilized to ensure the correct VMM functionality. If $activation\_flip$ is 1, we negate the partial sums obtained from all the columns of the corresponding Xbar array (i.e. on which the flipped input is applied). This is because $\sum I * W = -1 \cdot \sum(-I) * (W)$. Else, the partial sums of the Xbar are used as they are.

B. Hardware Implementation of TWINN

Fig 5(b) provides an overview of a CiM-BNN memory array utilizing TWINN. There are four key hardware modifications to implement TWINN. First, to support static weight flipping, an additional register (1 bit per Xbar column) is required for the storage of $column\_flip$.

Second, for dynamic activation flipping, computing $\sum I$ requires an adder tree near-memory. Interestingly, the NAND-Net architecture already includes an adder tree for computing $\sum I$ for the post-processing, allowing us to utilize the existing hardware without significant hardware overheads. The additional circuitry needed includes a digital comparator (to compare $\sum I$ to $\frac{n}{2}$) and XOR gates (1 per row) to flip I or not, depending on $activation\_flip$. Additionally, when flipping I, the $\sum I$ required for post-processing (Equation 1) becomes $n - \sum I$ due to the flip. To provide the correct value for post-processing, we incorporate a subtractor and a multiplexer to switch between $\sum I$ and $n - \sum I$ based on the value of $activation\_flip$. The subtraction is implemented using a ripple-carry adder in two's complement form: $n - \sum I$ is computed as $n + ones\_complement(\sum I) + 1$. The ones' complement is obtained by passing $\sum I$ through a bank of inverters, and the carry-in for the adder is set to 1.

Third, TWINN needs some additional circuitry to post-process the partial sums. Recall, NAND-Net requires the VMM output to undergo some post-processing to obtain the final VMM output (equation 1). On top of that, TWINN requires selective negation of partial sums based on the values of $activation\_flip$ and $column\_flip$. If $column\_flip[i]$ is 1 and $activation\_flip$ is 0 or vice versa, then the post-processed VMM output must be further multiplied by -1. However, if *both* $activation\_flip$ and $column\_flip$ are 1 or 0, then no further processing is needed, since $\sum I * W = \sum(-I) * (-W)$. Thus, the VMM output has to be multiplied with $(-1)^{(activation\_flip \oplus column\_flip)}$. We implement this using a XOR gate to calculate $activation\_flip \oplus column\_flip$, 2's complement circuitry to perform multiplication with -1, and a multiplexer to choose the final VMM output, with $activation\_flip \oplus column\_flip$ as the select signal.

Fourth, TWINN, interestingly, offers an opportunity to offset some of the overheads of the additional hardware discussed above. Typically for an Xbar where n rows are asserted in parallel, $\log(n)$ bit analog to digital converters (ADCs) are required to digitize the analog current/voltage. However, static weight and dynamic activation sparsification ensure that Xbar weight columns and activations have $\leq \frac{n}{2}$ 1's, respectively. As a result, the VMM outputs are guaranteed to be $\leq \frac{n}{2}$. This allows the use of a $\log(n) - 1$ bit ADC for digitization without introducing any errors, leading to some energy, latency, and area savings. We will quantify the overall hardware benefits/overheads of TWINN in Section V-C.

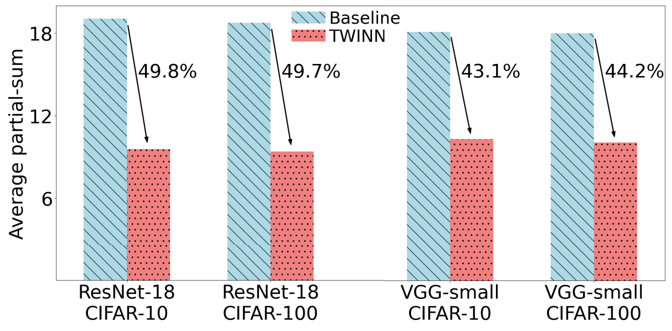

Fig. 6. Average partial-sum reduction using TWINN.

Finally, it is worth highlighting other works that utilize 'flipped' states in Xbars for various purposes. In [37], row and column flipping in DNN weight matrices are employed to reduce model storage requirements in ReRAM Xbar-based DNN accelerators. Similarly, [38] leverages row and column flipping in BNN weight matrices to improve resilience against stuck-at faults. Additionally, the ISAAC accelerator [39], like our work, utilizes the 'flipped' state for the weights to lower ADC precision by one bit, providing hardware benefits. TWINN extends these findings for BNNs by demonstrating that the benefits of 'flipped' weight encoding go beyond ADC precision reduction, also addressing the impact of non-idealities effectively. The non-ideality mitigation provided by TWINN is further strengthened by the proposed dynamic activation flipping.

V. Results

In this section, we evaluate the efficacy of TWINN by testing it on various BNNs across multiple datasets. We also evaluate the hardware overhead of TWINN across multiple design points.

A. Partial-Sum Reduction

First, we examine the partial-sum reduction achieved by TWINN for the above-mentioned workloads. For this evaluation, we apply the methodology outlined in Section III-A. Fig. 6 presents the average partial-sums for various workloads with and without TWINN (baseline). Our results show that TWINN reduces the average partial-sum magnitude by 43.1% - 49.8%. This sizeable reduction can be attributed to the multiplicative interaction between activations (I) and weights (W) in the Xbar. Since a bitcell is 'ON' only when both I and W are 1, transforming either I or W to 0 reduces the number of 'ON' bitcells. Consequently, the combined effects of static weight and dynamic activation flipping lead to a large reduction in partial-sum magnitude.

B. Accuracy Analysis

Next, we examine the accuracy of BNNs when deployed on NAND-Net based non-ideal Xbars, and quantify the accuracy improvements provided by TWINN. We utilize the framework described in Section III-B for our analysis. We perform a comprehensive analysis across multiple datasets–CIFAR-10

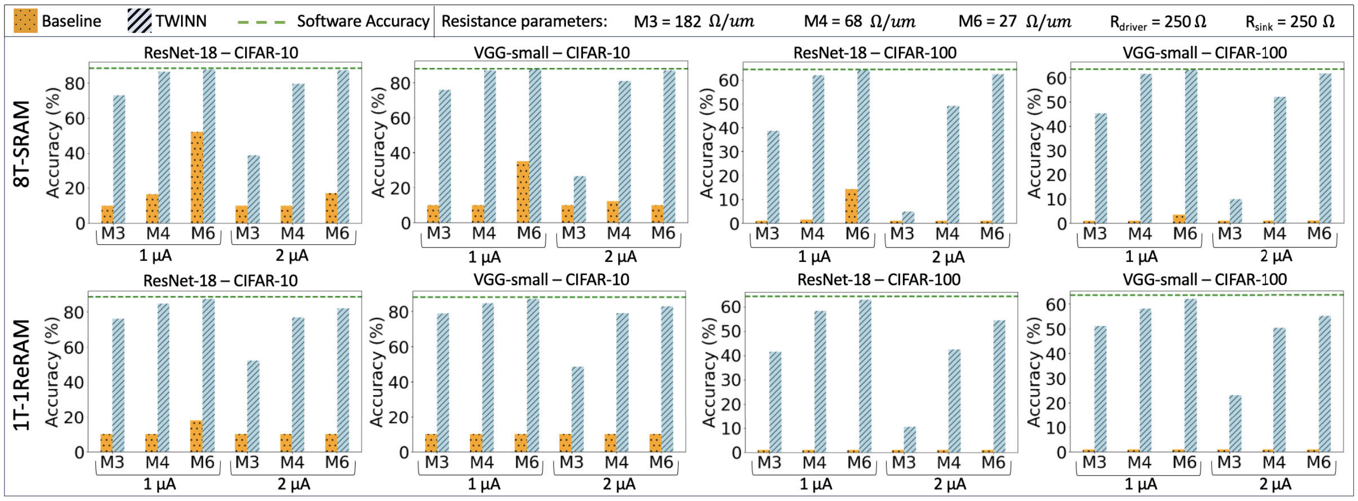

Fig. 7. Non-ideal inference accuracy for ResNet-18 and VGG-small CiM-BNNs across various hardware design points. Baseline CiM-BNN accuracy is severely degraded due to Xbar non-idealities, due to the large partial-sums. TWINN is able to significantly improve accuracy, even bringing to near-ideal values for some cases. M3, M4 and M6 refer to the BL/SL routing metal layers, and 1 μA and 2 μA refer to the ON currents of the bit-cell. The 1T-ReRAM HRS current is ~ 0.1 μA. We modulate the gap length in 1T-1ReRAM and the terminal voltages in 8T-SRAM to achieve the two different on-currents.

[40], CIFAR-100 [40], SVHN [41], and FashionMNIST [42], two memory technologies (8T-SRAM and 1T-1ReRAM), and two BNNs: ResNet-18 [43] and VGG-small [44].

We evaluate the accuracies for two bitcell 'ON' currents: 1 μA and 2 μA. The ON currents are controlled via V_{DD} tuning in 8T-SRAM and filament tunneling gap tuning in ReRAMs. While designs with higher 'ON current' offer larger distinguishability of output states and mitigate the effect of $I_{HRS/OFF}$ (especially in ReRAM), they also suffer from larger IR drops in the parasitic resistances. Further, we assess the accuracies for BL/SL routed in M3, M4, and M6 metal layers. Higher metal layers, such as M6, have larger width and height, leading to lower wire resistance and reduced non-idealities, albeit at the cost of array area, higher capacitance (energy) and lower porosity. We evaluate TWINN across these design points to shows its efficacy for various design choices.

Fig.7 summarizes the comparisons for CIFAR-10 and CIFAR-100. The baseline accuracy (without TWINN) is *severely* degraded by Xbar non-idealities, with maximum accuracies of 52.09% for CIFAR-10 and 18.08% for CIFAR-100 (for favorable design points such as M6 - 1 μA). For several other design points, near random-guess accuracies (10% for CIFAR-10 and 1% for CIFAR-100) are observed. On the other hand, TWINN provides significant accuracy improvements. For the M6 - 1μA case, TWINN helps attain accuracies within 0.5% of the ideal accuracies for all CiM-BNNs. Even for M6 - 2 μA and M4 - 1μA, the accuracies are increased to within 10% of the ideal values. Thus, the partial-sum reduction (Fig. 6) translates to significant accuracy gains. However, we observe that for the cases with larger non-idealities, e.g. M3 - 2 μA, the accuracy improvements may not be sufficient (but are still larger than the baseline). For such cases, TWINN needs to be used in conjunction with with other non-ideality mitigation solutions.

As discussed in Section III-C, our choice of the NAND-Net architecture–motivated by its hardware efficiency through the use of single-ended bitcells–does result in heightened vulnerability to Xbar non-idealities as compared to deployment on differential Xbars. To contextualize the impact, we also deploy BNNs on differential 8T-SRAM based Xbars for two design points: M6-1μA and M3-1μA. As expected, the differential configuration shows stronger resilience, yielding 87.05% accuracy for M6-1μA–substantially higher than the NAND-Net baseline. However, for more aggressive design points, even differential arrays exhibit severe accuracy degradation, with accuracy dropping to 20.92% for M3-1μA. These results highlight the general sensitivity of BNNs to Xbar non-idealities, regardless of the type of bitcell. Importantly, TWINN substantially enhances the robustness of NAND-Net based Xbars–achieving higher accuracy than differential configurations–while preserving the hardware efficiency benefits of single-ended bitcells and incurring only minor overheads, as discussed in Section V-C.

To broaden our analysis, we further evaluate ResNet-18 BNNs trained on the SVHN and FashionMNIST datasets across three hardware design points: 1) M6 - 1μA (low non-idealities), 2) M3 - 1μA (moderate non-idealities), and 3) M3 - 2μA (severe non-idealities). These design points were selected to represent varying degrees of Xbar non-idealities as well as energy/area trade-offs, as detailed previously. These simpler datasets are included to investigate whether baseline BNNs can achieve higher accuracy under non-ideal conditions in less complex workloads, and to assess the effectiveness of TWINN in mitigating non-idealities in such scenarios.

Fig. 8 presents the results for these datasets. Although SVHN and FashionMNIST are comparatively simpler datasets than CIFAR-10 and CIFAR-100–as evidenced by their higher software accuracies–they still exhibit severe accuracy degradation under baseline conditions. This degradation is especially pronounced for the M3 configurations, where accuracy falls sharply to near random-guess levels ($\sim$10%), mirroring the performance observed in the CIFAR benchmarks. For the relatively favorable M6 - 1μA case, baseline accuracies also see a reduction from the software accuracy levels: from 95.35%

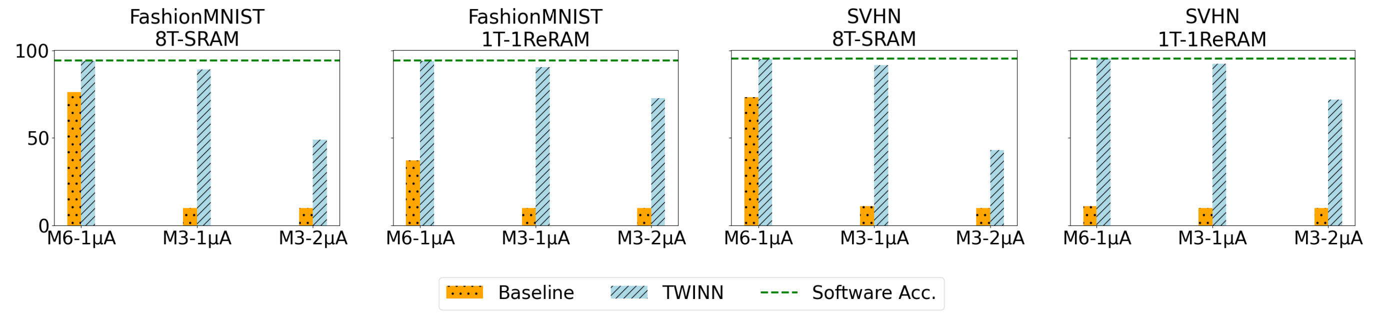

Fig. 8. Accuracy of ResNet-18 BNN on FashionMNIST and SVHN datasets.

TABLE II

Hardware Metrics of TWINN Relative to Baseline (Overhead % on Top, Absolute Values for TWINN in Parentheses Below). Energy and Latency Figures Correspond to a Single Vector-Matrix multiplication–i.e., One Input Vector Processed by a 64×64 Crossbar Array to Generate 64 Dot Products. Area Values Represent the Footprint of a Single Memory Macro, as Illustrated in Fig. 5 (b)

Memory	Energy (%)	Latency (%)			Area (%)			EDAP (%)		
		1 ADC	8 ADCs	64 ADCs	1 ADC	8 ADCs	64 ADCs	1 ADC	8 ADCs	64 ADCs
8T-SRAM	-9.0 (18.9 pJ)	-11.8 (440.2 ns)	-8.8 (56.9 ns)	14.8 (9.0 ns)	7.7 (212.9 μm^2)	4.9 (524.3 μm^2)	3.4 (3014.9 μm^2)	-13.6	-12.9	8.0
1T-1ReRAM	-9.4 (17.1 pJ)	-13.0 (396.1 ns)	-9.7 (51.4 ns)	16.3 (8.3 ns)	14.8 (117.4 μm^2)	6.0 (428.7 μm^2)	3.6 (2919.4 μm^2)	-9.0	-13.3	9.2

Note: Negative overhead indicates improvement compared to baseline. Absolute values reflect the actual metric values for TWINN.

down to 73.17% (8T-SRAM) and 11.07% (1T-1ReRAM) for SVHN, and from 94.19% down to 76.13% (8T-SRAM) and 37.13% (1T-1ReRAM) for FashionMNIST. However, TWINN dramatically improves accuracy, nearly restoring software accuracy for the M6 - 1μA case. Even for moderate non-idealities (M3 - 1μA), TWINN achieves accuracy within $\sim$ 5% of the ideal for both datasets and memory technologies. Although the most severe scenario (M3 - 2μA) continues to present substantial challenges, TWINN's accuracy improvements remain significant over the baseline suggesting its utility as a foundational mitigation strategy that can enhance the effectiveness or reduce the overheads of other mitigation solutions.

C. Hardware Overhead

We evaluate the hardware implications of TWINN at the memory macro level (Xbar + peripheral circuits). The actual impact may be lower once other components of the system are included. We evaluate the energy, latency and area of 64 × 64 Xbar arrays based on 8T-SRAM and 1T-1ReRAM using SPICE simulations and custom layout [26]. Note that the BL and SL of the Xbars are routed in M3 for the hardware evaluation. We obtain the energy-latency-area of the Xbar peripherals in the 7nm technology node based on the circuit modules from Neurosim [25]. For the baseline and TWINN, we utilize 6-bit and 5-bit SAR-ADCs, respectively, based on our discussion in Section IV-B. We conduct our hardware evaluation for three cases: 1) each Xbar *column* is equipped with a dedicated ADC (64 ADCs), resulting in reduced latency but larger area, 2) 8 Xbar columns share a single ADC (8 ADCs), leading to a smaller area but higher latency and 3) each Xbar has a single ADC (1 ADC), leading to the smallest area but largest latency.

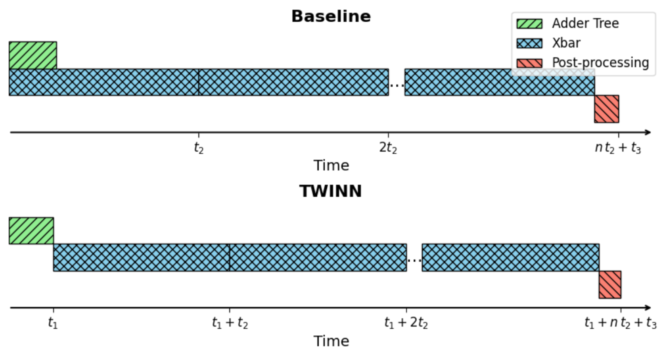

Fig. 9. Timing diagram showing that in the baseline, adder tree latency is masked by the Xbar operation, while in TWINN, it lies on the critical path and increases total latency. The number of Xbar operations depends on the degree of ADC sharing within the array. The exact latency (as shown in Table II) is affected by this as well as by the underlying memory technology and the specific design point.

Table II summarizes the results. We observe that despite the additional hardware (See Section IV), TWINN offers a 9-9.4% *reduction* in energy. This is because of the SAR-ADC bit precision reduction that offsets the effect of added peripherals.

The latency is governed by two opposing factors. First, there is an improvement in latency due to the reduced ADC precision. However, in the baseline, the output of the adder tree is only needed for post-processing, allowing its latency to be masked by the CiM latency. In contrast, TWINN needs the adder tree output for dynamic activation sparsification, adding the adder tree latency to the critical path (as shown in Fig. 9). For the design with 64 ADCs per array, the adder tree latency dominates, leading to a 14.8-16.3% increase in latency. Conversely, when 8 ADCs are used per array, the

ADC latency becomes the bottleneck, resulting in a latency reduction of 8.8-9.7%.

The area overhead of TWINN mainly arises from the extra register for storing the $column\_flip$ vector, is minimal, ranging from 3.4%-7.7% and 3.6%-14.8% for 8T-SRAM and 1T-1ReRAM,respectively, across all evaluated design points.

Lastly, to consolidate our results, we calculate the energy-delay-area product (EDAP) change caused by TWINN. We see that EDAP change ranges from -13.6% - 8.0% for 8T-SRAM and -13.3% - 9.2% for 1T-1ReRAM. Thus, TWINN offers significant accuracy improvements in CiM-BNNs with only minor hardware overhead.

D. Baseline Enhancement: ADC Reference Optimization and Partial Wordline Activation (PWA)

Our accuracy results discussed previously show that the baseline design is severely impacted by the Xbar non-idealities. To boost the baseline accuracy and assess the effectiveness of TWINN in comparison to an enhanced baseline design, we investigate two techniques–ADC reference optimization and partial wordline activation (PWA). These techniques are chosen for their compatibility with CiM-BNNs, training-free nature, and applicability across different memory technologies, much like the proposed TWINN technique.

1) ADC Reference Optimization: In conventional CiM designs, the ADC quantization step size (I_{quant}), which is the difference between successive ADC reference currents, is typically set to match the ON current of the bitcell (I_{LRS}), assuming ideal behavior [45], [46]. However, due to Xbar non-idealities such as parasitic resistances and current leakage, the actual column currents deviate from ideal values, causing inaccurate dot-product outputs. ADC reference optimization mitigates this by determining a workload-specific I_{quant} that minimizes the weighted average error of Xbar outputs. This error metric is computed by multiplying the average deviation for each partial-sum value with its probability of occurrence, derived from inference distributions over 10,000 training samples.

Note that employing ADC reference optimization makes the ADC references dependent on the specific workload and dataset, thereby necessitating additional on-chip circuitry capable of reconfiguring the reference currents for multi-workload scenarios. In this study, we focus on accelerators designed for a single workload–a common configuration for edge devices–which avoids the additional overhead associated with adaptive ADC reference optimization circuitry. However, for a more general purpose platform, the technique considered here to boost the baseline accuracy will have overheads.

Since our analysis has established that the general accuracy trends for the baseline and TWINN hold across various datasets, we focus on evaluating the impact of ADC reference optimization using the ResNet-18 BNN trained on CIFAR-10 dataset. The analysis is performed for both 8T-SRAM and 1T-1ReRAM-based Xbars across three hardware configurations: M6-1μA (low non-ideality), M3-1μA (moderate non-ideality), and M3-2μA (severe non-ideality). Accuracy results, presented in Fig. 10, highlight several key findings. First, strong improvements in the optimized baseline are seen in low and moderate scenarios (e.g., M6-1μA: from 52.1% to 87.8% in 8T-SRAM, and from 18.1% to 87.5% in 1T-1ReRAM), but no improvements are observed in the M3-2μA case due to overwhelming non-idealities. Second, TWINN without ADC optimization still outperforms the optimized baseline in all scenarios, emphasizing its efficacy. Third, integrating ADC reference optimization with TWINN consistently achieves substantial accuracy improvements across all hardware scenarios, bringing accuracy levels very close to the ideal software accuracy (88.5%).

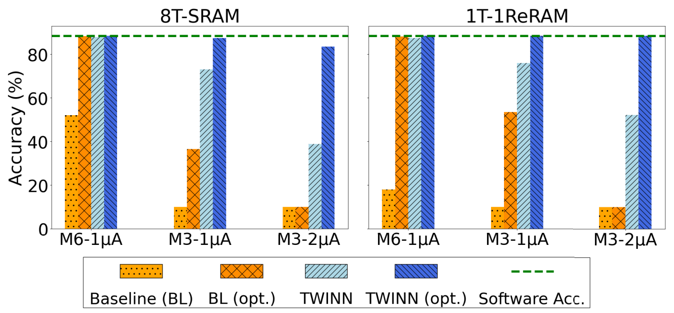

Fig. 10. Accuracy of ResNet-18 on the CIFAR-10 dataset with optimized ADC references.

2) Partial Wordline Activation (PWA): PWA is a common non-ideality mitigation strategy in CiM designs, where only a subset of wordlines is activated per cycle–effectively partitioning the dot-product computation [22], [47]. This reduces both the average and maximum partial-sum values per CiM operation, thereby lowering the impact of current deviation, which increases superlinearly with partial-sum magnitude. Additionally, PWA reduces ADC bit-precision requirements, enabling energy, latency, and area savings per cycle. However, since a full dot-product now spans multiple CiM cycles, total energy and latency increase.

We evaluate PWA using the same ResNet-18/CIFAR-10 setup and hardware configurations as ADC reference optimization. We specifically select a PWA value of 32, meaning that 32 out of the 64 rows in the 64×64 Xbar array are activated during each CiM cycle. This choice limits the maximum partial-sum to 32, thereby reducing the required ADC precision from 6 bits to 5 bits. We make this choice since it is consistent with TWINN, which also limits the partial-sum at 32 to achieve the same reduction in ADC precision.

Our findings, which are summarized in Fig. 11, reveal that PWA significantly boosts baseline accuracy under mild non-idealities (e.g., M6-1μA: 52.1% to 87.4% in 8T-SRAM and 18.1% to 71.9% in 1T-1ReRAM). However, it also is ineffective under severe conditions (e.g., M3-2μA: baseline remains at 10%).

TWINN again outperforms the PWA-enhanced baseline across all configurations. For example, in the M3-1μA case, TWINN achieves 73.1% (8T-SRAM) and 76.0% (1T-1ReRAM), compared to PWA-enhanced accuracies of just 16.0% and 13.1%, respectively. Unlike PWA, which incurs cumulative errors from multiple CiM cycles, TWINN completes each dot-product in a single cycle, avoiding compounded errors and the near 2x latency and energy overheads

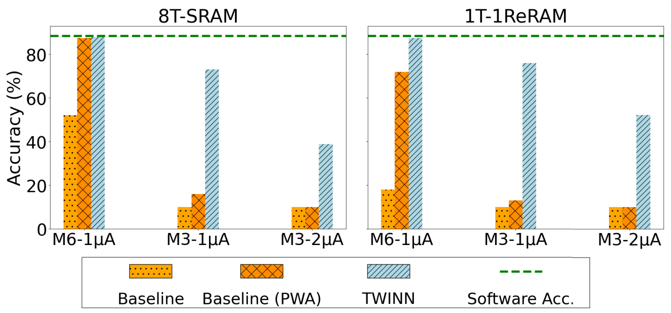

Fig. 11. Accuracy of ResNet-18 on the CIFAR-10 dataset with partial wordline activation (PWA).

associated with multiple CiM operations. This highlights the superiority of TWINN as a robust and efficient mitigation strategy for Xbar non-idealities.

VI. Conclusion

In this work, we examined the impact of Xbar non-idealities on the accuracy of CiM-BNN accelerators designed in scaled technology nodes. We showed that inherently, BNNs produce larger partial sums compared to higher precision DNNs, due to which the influence of Xbar non-idealities in CiM-BNNs is more pronounced. To address this challenge, we proposed TWINN–a training-free technique designed to mitigate the Xbar non-idealities specifically for CiM-BNNs. TWINN reduces partial sums and, consequently, the non-idealities in BNN Xbar arrays by employing two key operations: static weight flipping and dynamic activation flipping. Our experiments conducted on ResNet-18 and VGG-small BNNs demonstrate that TWINN achieves a 43.1% to 49.8% reduction in average partial-sum magnitudes. This leads to substantial accuracy improvements over a standard BNN, with accuracy gains reaching up to 77.25%. Furthermore, we assessed the hardware implications of TWINN and showed that the accuracy improvements are accompanied by a 9.0% to 9.4% reduction in energy consumption compared to the baseline. These benefits come at the cost of up to a 16.3% and 6.0% increase in latency and area, respectively. We also compared TWINN to other non-ideality mitigation techniques like ADC reference optimization and partial wordline activation, and found that TWINN outperforms them over a range of hardware design points, establishing it as a robust and low-overhead solution for mitigating Xbar non-idealities in CiM-BNNs.

Acknowledgment

The authors would like to thank Chunguang Wang for his help with the hardware evaluation.

References

[1] J. Chen and X. Ran, “Deep learning with edge computing: A review,” *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.

[2] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Quantized neural networks: Training neural networks with low precision weights and activations,” *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, Jan. 2016.

[3] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/d8330f857a17c53d217014ee776bfd50-Paper.pdf

[4] N. Verma et al., “In-memory computing: Advances and prospects,” *IEEE Solid-State Circuits Mag.*, vol. 11, no. 3, pp. 43–55, Aug. 2019.

[5] J. Lee, A. Lu, W. Li, and S. Yu, “NeuroSim V1.4: Extending technology support for digital compute-in-memory toward 1nm node,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 71, no. 4, pp. 1733–1744, Apr. 2024.

[6] S. Yin, Z. Jiang, J.-S. Seo, and M. Seok, “XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks,” *IEEE J. Solid-State Circuits*, vol. 55, no. 6, pp. 1733–1743, Jun. 2020.

[7] R. Liu et al., “Parallelizing SRAM arrays with customized bit-cell for binary neural networks,” in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6, doi: 10.1109/DAC.2018.8465935.

[8] K. Cho, A. Malhotra, and S. K. Gupta, “XNOR-VSH: A valley-spin Hall effect-based compact and energy-efficient synaptic crossbar array for binary neural networks,” *IEEE J. Explor. Solid-State Comput. Devices Circuits*, vol. 9, pp. 99–107, 2023.

[9] S. Lee, G. Jung, M. E. Fouda, J. Lee, A. Eltawil, and F. Kurdahi, “Learning to predict IR drop with effective training for ReRAM-based neural network hardware,” in *Proc. 57th Annu. ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.

[10] S. Jain and A. Raghunathan, “CxDNN: Hardware–Software compensation methods for deep neural networks on resistive crossbar systems,” *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 6, pp. 1–23, Nov. 2019, doi: 10.1145/3362035.

[11] C. Huang, N. Xu, K. Qiu, Y. Zhu, D. Ma, and L. Fang, “Efficient and optimized methods for alleviating the impacts of IR-drop and fault in RRAM based neural computing systems,” *IEEE J. Electron Devices Soc.*, vol. 9, pp. 645–652, 2021.

[12] A. Agrawal, C. Lee, and K. Roy, “X-CHANGR: Changing memristive crossbar mapping for mitigating line-resistance induced accuracy degradation in deep neural networks,” 2019, *arXiv:1907.00285*.

[13] B. Crafton, C. Talley, S. Spetalnick, J.-H. Yoon, and A. Raychowdhury, “Characterization and mitigation of IR-drop in RRAM-based compute in-memory,” in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2022, pp. 70–74.

[14] S. Jain, A. Sengupta, K. Roy, and A. Raghunathan, “RxNN: A framework for evaluating deep neural networks on resistive crossbars,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 2, pp. 326–338, Feb. 2021.

[15] J. Victor, D. Eun Kim, C. Wang, K. Roy, and S. Gupta, “WAGONN: Weight bit agglomeration in crossbar arrays for reduced impact of interconnect resistance on DNN inference accuracy,” 2024, *arXiv:2406.14706*.

[16] Z. He, J. Lin, R. Ewetz, J. Yuan, and D. Fan, “Noise injection adaption: End-to-end ReRAM crossbar non-ideal effect adaption for neural network mapping,” in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, Jun. 2019, pp. 1–6.

[17] M.-G. Lin et al., “D-NAT: Data-driven non-ideality aware training framework for fabricated computing-in-memory macros,” *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 12, no. 2, pp. 381–392, Jun. 2022.

[18] X. Sun et al., “PCM-based analog compute-in-memory: Impact of device non-idealities on inference accuracy,” *IEEE Trans. Electron Devices*, vol. 68, no. 11, pp. 5585–5591, Nov. 2021.

[19] G. Bonilla, N. Lanzillo, C.-K. Hu, C. J. Penny, and A. Kumar, “Interconnect scaling challenges, and opportunities to enable system-level performance beyond 30 nm pitch,” in *IEDM Tech. Dig.*, Dec. 2020, pp. 20.4.1–20.4.4.

[20] M. J. Rasch et al., “Hardware-aware training for large-scale and diverse deep learning inference workloads using in-memory computing-based accelerators,” *Nature Commun.*, vol. 14, no. 1, p. 5282, Aug. 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID

[21] N. Lepri, M. Baldo, P. Mannocci, A. Glukhov, V. Milo, and D. Ielmini, “Modeling and compensation of IR drop in crosspoint accelerators of neural networks,” *IEEE Trans. Electron Devices*, vol. 69, no. 3, pp. 1575–1581, Mar. 2022.

[22] Y. Park, S. Y. Lee, H. Shin, J. Heo, T. J. Ham, and J. W. Lee, “Unlocking wordline-level parallelism for fast inference on RRAM-based DNN accelerator,” in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2020, pp. 1–9.

[23] H. Kim, J. Sim, Y. Choi, and L.-S. Kim, "NAND-Net: Minimizing computational complexity of in-memory processing for binary neural networks," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 661–673.

[24] D. Hendrycks and K. Gimpel, "Gaussian error linear units (GELUs)," 2016, *arXiv:1606.08415*.

[25] X. Peng, S. Huang, Y. Luo, X. Sun, and S. Yu, "DNN+NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies," in *IEDM Tech. Dig.*, Dec. 2019, p. 32.

[26] C. Wang, J. Victor, and S. K. Gupta, "Comparative evaluation of memory technologies for synaptic crossbar arrays— Part I: Robustness-driven device-circuit co-design and system implications," 2023, *arXiv:2307.04261*.

[27] A. Jaiswal, I. Chakraborty, A. Agrawal, and K. Roy, "8T SRAM cell as a multibit dot-product engine for beyond von Neumann computing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 11, pp. 2556–2567, Nov. 2019.

[28] *ASAP 7nm Predictive PDK*. Accessed: Aug. 2024. [Online]. Available: https://asap.asu.edu/

[29] Z. Jiang et al., "A compact model for metal-oxide resistive random access memory with experiment verification," *IEEE Trans. Electron Devices*, vol. 63, no. 5, pp. 1884–1892, May 2016.

[30] X. Chen, C.-L. Lo, M. C. Johnson, Z. Chen, and S. K. Gupta, "Modeling and circuit analysis of interconnects with TaS2 Barrier/Liner," in *Proc. Device Res. Conf. (DRC)*, Jun. 2021, pp. 1–2.

[31] S. Yu, P.-Y. Chen, Y. Cao, L. Xia, Y. Wang, and H. Wu, "Scaling-up resistive synaptic arrays for neuro-inspired architecture: Challenges and prospect," in *IEDM Tech. Dig.*, Dec. 2015, p. 17.

[32] I. Chakraborty, M. Fayez Ali, D. Eun Kim, A. Ankit, and K. Roy, "GENIEx: A generalized approach to emulating non-ideality in memristive xbars using neural networks," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.

[33] A. S. Rekhi et al., "Analog/mixed-signal hardware error modeling for deep learning inference," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, Jun. 2019, pp. 1–6.

[34] J. Park et al., "Multi-level, forming and filament free, bulk switching trilayer RRAM for neuromorphic computing at the edge," *Nature Commun.*, vol. 15, no. 1, p. 3492, Apr. 2024, doi: 10.1038/s41467-024-46682-1.

[35] S. K. Roy et al., "Compute SNDR-boosted 22-nm MRAM-based in-memory computing macro using statistical error compensation," *IEEE J. Solid-State Circuits*, vol. 60, no. 3, pp. 1092–1102, Mar. 2025.

[36] Y. Emma Wang, G.-Y. Wei, and D. Brooks, "Benchmarking TPU, GPU, and CPU platforms for deep learning," 2019, *arXiv:1907.10701*.

[37] L. Zhao, Y. Zhang, and J. Yang, "Flipping bits to share crossbars in reram-based dnn accelerator," in *Proc. 2021 IEEE 39th Int. Conf. Comput. Design (ICCD)*, 2021, pp. 17–24.

[38] A. Malhotra, C. Wang, and S. K. Gupta, "BNN-flip: Enhancing the fault tolerance and security of Compute-in-Memory enabled binary neural network accelerators," in *Proc. 29th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2024, pp. 146–152.

[39] A. Shafiee et al., "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, Seoul, South Korea, 2016, pp. 14–26.

[40] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[41] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS*, Jan. 2011, pp. 1–13. [Online]. Available: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf

[42] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, *arXiv:1512.03385*.

[44] D. Zhang, J. Yang, D. Ye, and G. Hua, "LQ-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Jan. 2018, pp. 373–390.

[45] I. Chakraborty et al., "Resistive crossbars as approximate hardware building blocks for machine learning: Opportunities and challenges," *Proc. IEEE*, vol. 108, no. 12, pp. 2276–2310, Dec. 2020.

[46] H. Jia, H. Valavi, Y. Tang, J. Zhang, and N. Verma, "A programmable heterogeneous microprocessor based on bit-scalable in-memory computing," *IEEE J. Solid-State Circuits*, vol. 55, no. 9, pp. 2609–2621, Sep. 2020.

[47] T.-H. Yang et al., "Sparse ReRAM engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *Proc. ACM/IEEE 46th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2019, pp. 236–249.

Akul Malhotra received the B.E. degree in electrical and electronics engineering from the Birla Institute of Technology and Science, Pilani, India. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Purdue University, advised by Prof. Sumeet Gupta. During the Ph.D. degree, he interned with MediaTek in 2023 and IBM in 2024. His research focuses on improving the robustness of compute-in-memory-based hardware accelerators through circuit and system design techniques.

Sumeet Kumar Gupta received the B.Tech. degree in electrical engineering from Indian Institute of Technology, Delhi, India, in 2006, and the M.S. and Ph.D. degrees in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 2008 and 2012, respectively. He is currently an Elmore Associate Professor in electrical and computer engineering with Purdue University. Prior to this, he was an Assistant Professor in electrical engineering with The Pennsylvania State University from 2014 to 2017 and a Senior Engineer with Qualcomm Inc., from 2012 to 2014. His research interests include low power VLSI circuit design, in-memory computing, AI hardware design, nanoelectronics and spintronics, device-circuit co-design, and nano-scale device modeling and simulations. He has published over 150 articles in refereed journals and conferences and is a Senior Member of EDS. He was a recipient of DARPA Young Faculty Award in 2016, Early Career Professorships by Purdue and Penn State in 2021 and 2014, respectively, and the Sixth TSMC Outstanding Student Research Bronze Award in 2012. He has also received Magoon Award and the Outstanding Teaching Assistant Award from Purdue University in 2007 and Intel Ph.D. Fellowship in 2009.