Hardware-Aware Quantization for Accurate Memristor-Based Neural Networks

Diware, Sumit; Yaldagard, Mohammad Amin; Bishnoi, Rajendra

DOI
10.1145/3676536.3698023

Publication date
2025

Document Version
Final published version

Published in
Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design

Invited Paper

Hardware-Aware Quantization for Accurate Memristor-Based Neural Networks

Sumit Diware Mohammad Amin Yaldagard Rajendra Bishnoi
Computer Engineering Lab, Delft University of Technology, Delft, The Netherlands
{S.S.Diware,M.A.Yaldagard,R.K.Bishnoi}@tudelft.nl

Abstract

Memristor-based Computation-In-Memory (CIM) has emerged as a compelling paradigm for designing energy-efficient neural network hardware. However, memristors suffer from conductance variation issue, which introduces computational errors in CIM hardware and leads to a degraded inference accuracy. In this paper, we present a hardware-aware quantization to mitigate the impact of conductance variation on CIM-based neural networks. We achieve this using the inherent characteristics of fixed-point arithmetic in CIM hardware. By tuning the bit-precision of weights, we align the conductance variation-induced errors with lower-order output bits. This reduces their numerical impact on the fixed-point output. We further decrease the residual errors by selectively discarding bits with low information and high error. This leads to error-free computations and a high inference accuracy. Our proposed methodology achieves 5.6× correct operations per unit energy compared to the conventional approach, while incurring very low hardware overheads.

CCS Concepts

• **Hardware → Emerging architectures**.

Keywords

Deep neural networks (DNNs), Quantization, Memristors, RRAM, Computation-In-Memory (CIM), Processing-In-Memory (PIM), Conductance variation, Fixed-point arithmetic, Non-ideality.

ACM Reference Format:
Sumit Diware Mohammad Amin Yaldagard Rajendra Bishnoi . 2024. Hardware-Aware Quantization for Accurate Memristor-Based Neural Networks. In *Proceedings of 2024 ACM/IEEE International Conference on Computer-Aided Design (ICCAD'24).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Neural networks are the cornerstone of modern artificial intelligence (AI), capable of performing a wide range of cognitive tasks without explicit programming [21]. Traditionally, these networks are implemented on von Neumann architecture-based hardware such as CPUs, GPUs, and specialized AI processors like TPUs [12, 13, 20]. However, the physical separation between memory and processing units in these systems leads to poor energy efficiency, known as the memory wall [5]. Computation-in-memory (CIM) addresses this issue by integrating computation directly within the memory units [9, 35]. This is achieved using emerging non-volatile memory technologies called memristors, which offer high scalability and fabrication compatibility with CMOS technology [32]. Deploying a neural network on CIM hardware starts with quantizing its weights and inputs into fixed-point format, for compatibility with integer-based CIM arithmetic. These post-quantization values are then mapped to CIM hardware components which perform raw in-place calculations. Finally, the raw outputs undergo fixed-point post-processing to obtain the final result. Despite its advantages, CIM suffers from conductance variation nonideality, which causes programmed memristor conductance to deviate from its intended value [10]. This arises due to fabrication imperfections and inherent stochastic nature of device physics. Conductance variation leads to incorrect weight storage in memristors and causes each memristor to contribute computational errors. The impact of these errors intensifies during arithmetic post-processing and propagation through subsequent layers, leading to a reduced neural network accuracy.

State-of-the-art techniques for conductance variation mitigation can be grouped into four categories: i) on-chip training, ii) off-chip training, iii) characterization-driven mapping, iv) hardware compensation. Firstly, on-chip training accommodates conductance variation impact by training the network directly on CIM chip [26, 27]. However, it is not scalable as each chip needs individual training. Moreover, it suffers from high energy consumption and endurance issues due to frequent write operations on memristors. Secondly, off-chip training aims to account for conductance variation during software training. This is done in three ways: (1) incorporating a hardware-calibrated conductance variation model [7, 19], (2) injecting computational noise based on hardware characterization [3, 41], and (3) restricting weight values for mapping to low-variation conductance states [15]. They all suffer from scalability issue. This is because the first two need per-chip characterization and training, while for the last one adds extra complexity to hyperparameter tuning. Thirdly, works like [8, 40] use characterization information to avoid mapping large weights to high-variation memristors. However, they are not scalable due to the need for per-chip characterization and cannot address errors accumulated from variations in small weights. Lastly, some techniques [6, 17, 25] introduce additional hardware components to mitigate conductance variation. They suffer from energy and area overheads, while also increasing the design complexity. Hence, there is a pressing need for a scalable and low-overhead solution to mitigate conductance variation.

In this paper, we present a methodology that leverages fixed-point quantization to overcome the impact of conductance variation.

ICCAD '24, October 27–31, 2024, New York, NY, USA

ACM ISBN 979-8-4007-1077-3/24/10.
https://doi.org/10.1145/3676536.3698023

We begin by analyzing how fixed-point quantization in CIM hardware affects computational errors caused by conductance variation. Based on this analysis, we propose a two-step quantization strategy for conductance variation mitigation. We first tune the weight quantization format, so that the errors due to conductance variation get aligned with lower-order output bits. This diminishes their numeric significance and reduces their impact on layer outputs. Second, we minimize the residual errors by strategically reducing the output bit-size. This is achieved by discarding bits that contain low-value information but high residual errors. Thus, we achieve significant error reduction with minimal information loss, leading to high inference accuracy. Our proposed approach is scalable, as it neither impacts the training process nor requires hardware characterization. Moreover, it incurs minimal design changes and hardware overhead because it only affects binary bit patterns without altering word lengths. Our key contributions are summarized as follows:

- An analysis of fixed-point quantization impact on conductance variation in CIM hardware.
- A weight quantization tuning technique to reduce the impact of conductance variation-induced errors.
- An approach to reduce the residual error while still maintaining high accuracy.

Simulation results show that our proposed approach provides 5.6× correct operations per unit energy compared to the conventional approach, with very small overheads.

The rest of this paper is organized as follows: Section 2 presents the basics of CIM. The proposed quantization approach is described in Section 3, followed by simulation results in Section 4. Finally, Section 5 concludes the paper.

2 Computation-In-Memory (CIM) Paradigm for Neural Networks

2.1 CIM Architecture

Vector-matrix multiplication (VMM) operations account for over 70% of the computations in neural networks [36]. Computation-in-memory (CIM) performs in-situ data processing to achieve energy-efficient VMM operations compared to traditional von Neumann hardware [1, 34, 38]. Fig. 1 shows the mapping of a VMM operation between neural network layers to a CIM hardware. It uses memory elements known as memristors, which store data in the form of conductance. They are organized in a grid-like structure known as a crossbar. It operates in the analog domain and interfaces with other digital components in the system using data converter circuits, such as digital-to-analog converters (DACs) and analog-to-digital converters (ADCs). Network weights are stored as memristor conductances (G) in the crossbar, while the inputs are applied as voltages (V) using DACs. The resulting current through each conductance is equivalent to element-wise multiplication of voltage and conductance, as per Ohm's law. The currents from conductances in the same column accumulate according to Kirchhoff's law. Each resulting output current (I) represents a multiply-and-accumulate operation in the analog domain. The output currents across all columns then collectively represent one VMM operation. These analog VMM outputs are converted to digital outputs using ADCs and then sent to other system components for further processing.

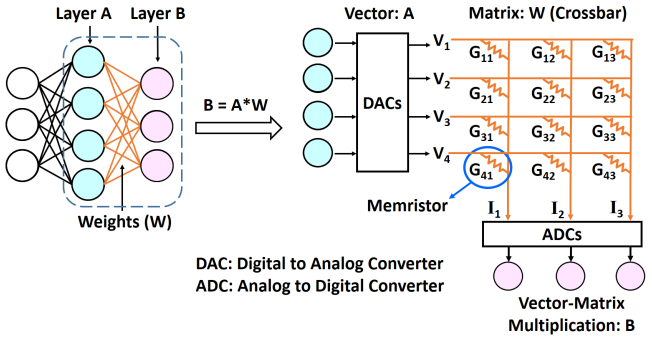
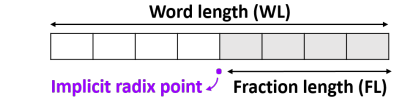


Figure 1: CIM-based vector-matrix multiplication operation in neural network.

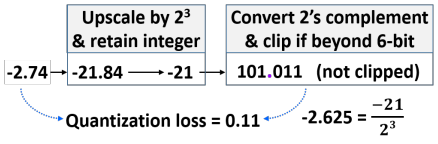
2.2 CIM Arithmetic

2.2.1 Fixed-point Numbers. CIM hardware supports integer arithmetic, while neural network weights and inputs are typically represented as real numbers. To bridge this gap, fixed-point representation is used. It can be specified as (WL, FL), where it expresses a real number as WL-bit 2's complement binary number with an implicit radix point separating FL fractional bits. This is shown in Fig. 2a. Thus, hardware treats fixed-point number as an integer with implicit scaling factor. It is processed using integer arithmetic, coupled with scaling factor recovery via downscaling (right-shifting) the final output. Thus, fixed-point format facilitates processing of real valued neural network using integer-based CIM hardware.

Converting a real-valued weight or input to fixed-point involves two key steps: i) Upscaling by 2^{FL} and retaining the integer part, and ii) Converting this integer to a 2's complement binary format, with clipping if it exceeds the WL-bit limit. An example of this conversion process is shown in Fig. 2b. Quantization loss during fixed-point conversion can be minimized through a careful selection of WL and FL, without affecting network accuracy. After the conversion, the arithmetic operations on fixed-point operands can be carried out as follows. Operands with same (WL, FL) format can be added or subtracted directly, while those with different formats must first be converted to a common format. Operands can be multiplied directly regardless of (WL, FL) formats. The WL and FL of the product are is the sum of WLs and FLs of the operands, respectively.



(a) Representation structure.



(b) An example of conversion process.

Figure 2: Fixed-point number format.

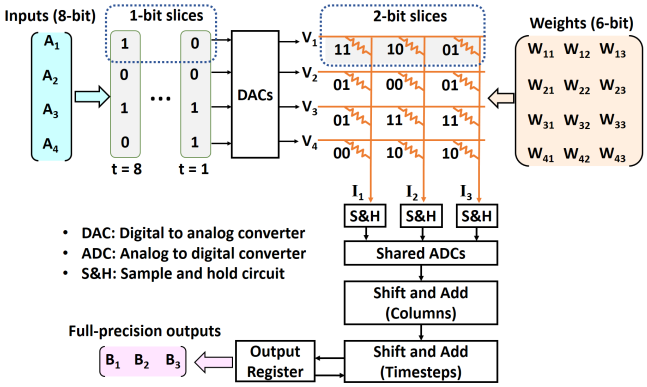


Figure 3: Bit-slicing in CIM hardware.

2.2.2 Bit-slicing Technique. CIM hardware cannot directly meet the bit-size desired for fixed-point weights and inputs. This limitation arises as the bit capacity of memristors is usually less than that desired for fixed-point weights. Additionally, fixed-point layer inputs and outputs require high-resolution data converters (DACs and ADCs), that are costly in terms of energy and area. To address this, a technique called bit-slicing is employed [1, 2, 14, 16, 34]. It involves breaking down full-precision weights and inputs into smaller chunks called slices, as shown in Fig. 3. Weight slices are mapped to conductances in adjacent crossbar columns, while input slices are applied to crossbar at different timesteps. The resulting column currents represent a partial VMM operation for that specific timestep. These partial VMM outputs are converted to digital domain by ADCs. As the ADCs are typically shared across multiple columns due to their large pitch size, sample and hold circuits preserve the column outputs till served by an ADC. The ADC outputs undergo shift-and-add operations to account for weight slicing. An additional round of shift-and-add operations then merges the current timestep output with that of the previous timestep, to account for input slicing. The final full-precision fixed-point output is obtained by repeating this process till the last timestep of input.

2.3 Memristor Device Technology

Memristor, also called resistive random access memory (RRAM), is a non-volatile memory device that stores data in the form of conductance. It is made up of an oxide material sandwiched between two metal electrodes [4, 33], as shown in Fig. 4. Memristor conductance can be modulated by creating and disrupting a conductive filament

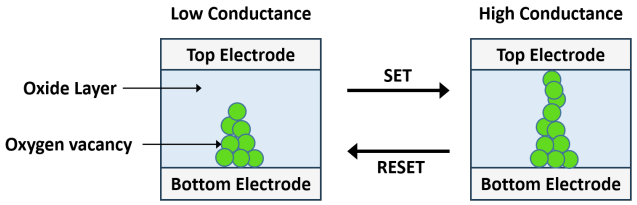
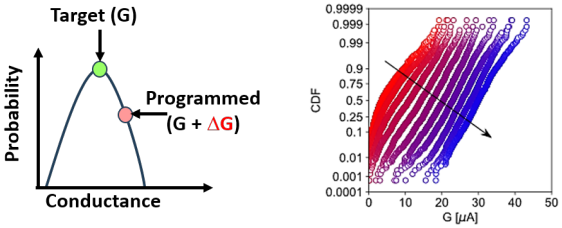


Figure 4: Resistive random access memory (RRAM) device.



(a) Concept illustration. **(b) Measurements in [30].**

Figure 5: Conductance variation in memristors.

(CF) composed of oxygen vacancies within the oxide layer. Its conductance is high when CF connects the electrodes (logic 1), while a disrupted CF leads to low conductance (logic 0). The SET process achieves a high conductance state through a high electric field that forces oxygen ions to drift toward one electrode. This leaves behind vacant oxygen sites in oxide layer, which form CF to increase its conductivity. The RESET process leads to low conductance state using an electric field with polarity opposite to that in SET process. This causes oxygen ions to migrate back into the oxide layer. These ions combine with oxygen vacancies to disrupt the CF, reducing the conductivity of oxide layer. Moreover, a single RRAM device can store multiple bits by exhibiting incremental conductance states via partial SET/RESET processes [22].

2.4 Conductance Variation

The programmed conductance of an RRAM memristor deviates from its target value due to the stochastic nature of filament creation and fabrication imperfections such as variable oxide thickness [8]. This phenomenon is known as conductance variation, shown in Fig. 5. Consequently, when a neural network model is deployed for inference on CIM hardware, the actual memristor conductance (G+ΔG) deviates from the expected conductance (G), as illustrated in Fig. 6. This gives rise to deviation (ΔI) from its ideal current contribution (I). These current deviations get accumulated via Kirchhoff's law and introduce errors in the output. Such erroneous computations then propagate through neural network layers and reduce the inference accuracy on CIM hardware. This undermines the benefits of CIM, as energy-efficient computations are of no value if they are functionally incorrect. In this paper, we improve

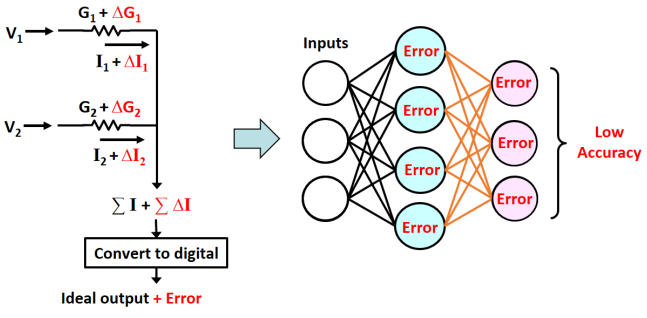


Figure 6: Impact of conductance variation on the accuracy of CIM-based neural network.

the accuracy of memristor-based neural network architectures in the presence of conductance variation.

3 Proposed Quantization Methodology

3.1 Overview

Fig. 7 shows the operation of a CIM-based neural network layer. It involves two distinct quantization phases. The first phase, denoted as q1, converts inputs and weights to fixed-point format. These values then are mapped to CIM hardware, which performs vector-matrix multiplication (VMM) to generate the full-precision output. Directly feeding this output to the next layer is not desirable. This is because it leads to progressively higher output bit-sizes in subsequent layers, increasing the hardware resource required by the network. Moreover, neural networks can maintain good accuracy with lower bit precision outputs as they inherently tolerate imprecise computations. Hence, a second quantization phase, denoted as q2, is employed to reduce the bit precision of the layer output.

In the conventional approach [11, 34, 39], q1 focuses solely on minimizing quantization loss and does not address conductance variation. The q2 phase then discards lower-order bits from the layer output to reduce its bit precision. However, the discarded bits contribute very little to overall conductance variation-induced error, due to their lower numeric significance. Consequently, the resulting low-precision outputs still retains significant errors from conductance variation. As a result, the low-precision outputs still retain significant errors from conductance variation. These errors propagate through subsequent layers, leading to degradation of inference accuracy on CIM hardware.

To address this challenge, our proposed approach first analyzes how fixed-point quantization in CIM hardware affects conductance variation-induced errors. Using this analysis, we develop a two-step quantization strategy for mitigating these errors. First, we tune the weight quantization in q1 phase to align conductance variation-induced errors with lower-order output bits. This reduces numeric significance of these errors, suppressing their impact on CIM arithmetic. Next, the q2 phase reduces the residual impact of these errors by discarding bits with low information and high error content. Thus, our approach achieves significant error reduction with minimal information loss. This results in a high inference accuracy on CIM hardware. We will now discuss the implementation details of this approach in upcoming subsection.

3.2 Quantization Strategy

We start with analyzing the impact of CIM hardware's fixed-point arithmetic on computational errors due to conductance variation. In a CIM crossbar, column current represents a multiply-accumulate operation between conductances (weight slices) and voltages (input slices). Conductance variation introduces additional error current in crossbar column. This analog error propagates through ADC and digital post-processing stages, to produce the full-precision error in the form of a binary integer. The numeric impact of this error in fixed-point format can be quantified as $E \div 2^{FL}$, where E denotes the integer digital error and FL indicates output fraction length. Moreover, the higher-order output bits of this error are less impacted by the output fraction length, as they are positioned beyond the implicit radix point.

We derive the following key insights from this analysis:

- A higher output fraction length can suppress the impact of conductance variation-induced errors on CIM hardware. This is shown in Fig. 8a.
- Residual error can still persist despite high output fraction length and is mainly governed by higher-order bits, as depicted in Fig 8b.

Thus, a quantization strategy must maximize the output fraction length and reduce the residual errors to mitigate conductance variation. Output fraction length can be optimized in q1 phase, it depends on fixed-point formats of inputs and weights. The q2 phase can handle residual errors, as truncation can be leveraged to discard

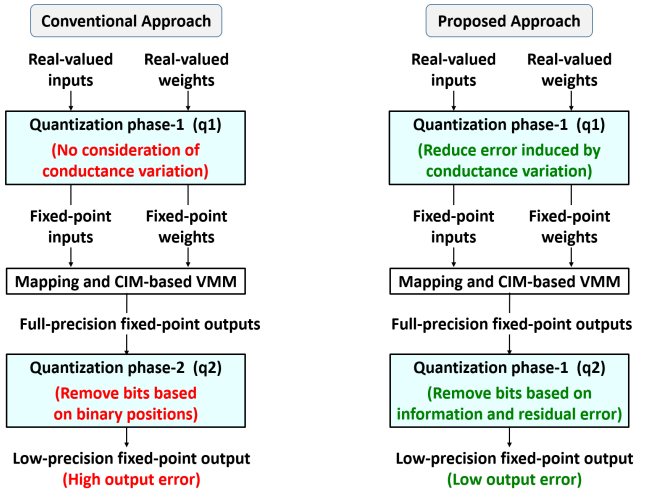
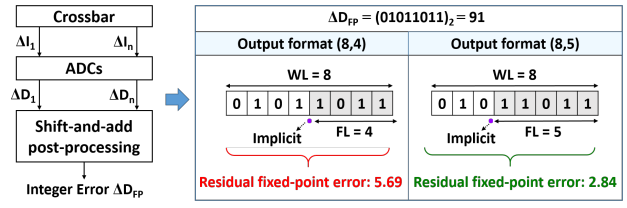
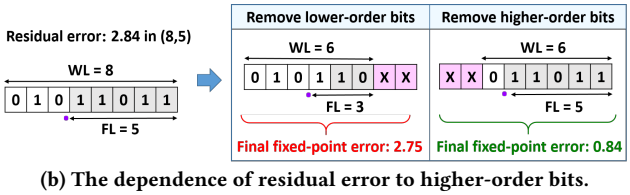


Figure 7: Overview of the conventional and proposed quantization approaches for CIM-based neural network layers.



(a) The impact of output fraction length on post-processed error.



(b) The dependence of residual error to higher-order bits.

Figure 8: Examples illustrating how the CIM fixed-point arithmetic affects errors induced by conductance variation.

them. Upcoming subsections will detail how the q1 and q2 phases achieve these objectives.

3.3 Quantization Phase q1

In this phase, our goal is to maximize the output fraction length to reduce the impact of conductance variation. Achieving this directly is challenging, because the output fraction length depends on both weight and input fraction lengths. Hence, we indirectly achieve this by maximizing the fraction length of weights or inputs. However, dealing with input fraction length is difficult and less scalable, due to the broad input data distribution across various deployment environments. In contrast, adjusting the weight fraction length is more manageable and scalable, as weights typically remain static after training. Hence, we decide to maximize the weight fraction length (denoted as W_{FL}) in this phase.

To maximize W_{FL}, we have to examine its impact on CIM hardware. A small W_{FL} leads to a shorter fractional part in the output, which may not effectively suppress conductance variation errors. However, it accommodates a broader range of weight values and provides enough integer bits to correctly capture full precision layer output. Thus, a small W_{FL} offers limited error suppression but maintains good arithmetic precision. In contrast, a large W_{FL} increases the fractional part of the output, enhancing the suppression of conductance variation errors. But this comes at the cost of representing a narrower range of weight values and insufficient integer bits to capture the full precision layer output. Hence, a large W_{FL} delivers better error suppression but provides poor arithmetic precision. However, achieving high inference accuracy on CIM hardware requires both effective error suppression and good arithmetic precision. Although extreme W_{FL} values are suboptimal for this purpose, an optimal balance can be found at the intermediate values W_{FL} values through design space exploration. This exploration must be performed in conjunction with the q2 phase, otherwise bit truncation in q2 can negate the benefits of an optimal W_{FL}. The next subsection will first present q2 implementation, followed by a unified design space exploration for both phases.

3.4 Quantization Phase q2

In this phase, we aim to minimize the residual errors in higher-order bits of the layer output while reducing its bit-size. This can be achieved by leveraging two key insights. Firstly, position of a bit does not necessarily reflect its information content. This is because the output bit width is determined at design time to accommodate the maximum output. However, run-time output often falls short of this maximum and leads to underutilized higher-order bits. These positions are filled by sign extension bits in 2's complement format, which do not contain any new information. In contrast, lower-order bits can hold significant information when working with small inputs, as most weights have magnitudes less than 1. Hence, it is better to discard a mix of higher and lower-order bits, unlike the conventional approach which only discards lower-order bits [11, 34, 39]. Secondly, higher-order output bits are the major source of residual error as discussed in Section 3.2. Thus, discarding more higher-order bits decreases the residual error. We combine these findings to infer that minimizing residual error requires maximizing the removal of higher-order bits containing less information.

To determine the optimal number of higher-order bits to be discarded, we start by defining full precision output word length as

$$O_{\mathrm{WL}} = A_{\mathrm{WL}} + W_{\mathrm{WL}} + log_2 R \tag{1}$$

where A_{WL} and W_{WL} denote word lengths of input activations and weights respectively, while R denotes the number of crossbar rows. The total number of output bits to discard is given by

$$D = O_{\mathrm{WL}} - O_{\mathrm{RWL}} = D_{\mathrm{H}} + D_{\mathrm{L}} \tag{2}$$

where O_{RWL} denotes the desired reduced output word length, D_H denotes higher-order bits discarded from the integer part and D_L denotes lower-order bits discarded from the fractional part. This gives us the number of higher-order bits to be discarded as

$$D_{\mathrm{H}} = A_{\mathrm{WL}} + W_{\mathrm{WL}} + log_2 R - O_{\mathrm{RWL}} - D_{\mathrm{L}} \tag{3}$$

Thus, we end up with two optimization variables, D_H here and W_{FL} from q1 phase. We can optimize them both through a single design space exploration by setting $O_{RWL} = A_{WL}$ and $D_F = W_{FL}$ to obtain

$$D_{\mathrm{H}} = W_{\mathrm{WL}} + log_2 R - W_{\mathrm{FL}} \tag{4}$$

Since R and W_{WL} are design time constants, determining optimal W_{FL} via design space exploration suffices to determine D_H using Eq. 4. This approach is also quite intuitive. A small W_{FL} leads to a small fraction part in the output and leaves more bits for the integer part. This creates more sign extension bits at higher-order positions, thereby allowing more higher-order bits to be discarded. This is corroborated by Eq. 4, which shows that a small W_{FL} leads to a high D_H. Conversely, a large W_{FL} leads to small integer part in the output. This reduces the number of sign extension bits and permits fewer higher-order bits to be discarded. This also is consistent with Eq. 4, where a large W_{FL} corresponds to a lower D_H.

We now present Algorithm 1 which determines optimal W_{FL}, thereby indirectly determining D_H. It is provided with fixed-point

Algorithm 1: Determining optimal weight fraction length and number of higher-order output bits to discard, for high inference accuracy on CIM hardware.

input : Neural network (NN), CIM architecture (CimA), Input data (A), Input fixed-point format (A_{WL}, A_{FL}), Weight word length (W_{WL}), Weight fraction length list (List_W_{FL}), accuracy threshold (Acc_{th})

output: Weight fraction length (W_{FL}), Number of discardable higher-order bits in layer output (D_H)

```
HW_acc_database ← ∅;
for FL in List_W_FL do
    D_H = W_WL + log2R - FL;
    W_q = fxp_quant(W, W_WL, FL);
    A_q = fxp_quant(A, A_WL, A_FL);
    fxp_acc ← fxp_sim(NN, W_q, A_q, FL, D_H);
    if fxp_acc > Acc_th then
        HW_acc ← HW_sim(CimA, W_q, A_q, FL, D_H);
        HW_acc_database.insert(Hw_acc, FL, D_H);
W_FL, D_H ← maximum_accuracy(HW_acc_database);
return W_FL, D_H;
```

format of network inputs, weight word length, an accuracy threshold, and a list of potential W_{FL} values. It is also equipped with details of neural network structure and CIM system architecture, to estimate ideal fixed-point accuracy and CIM hardware accuracy, respectively. For each value in the list of potential W_{FL} values, we quantize the inputs as well as the weights and also calculate the corresponding D_H value from Eq. 4. We then estimate the ideal fixed-point accuracy using these values. If it is above the predefined threshold, we proceed to estimate the CIM hardware accuracy. The resulting hardware accuracy value is stored in a database along with its corresponding W_{FL} and D_H. Finally, we select the W_{FL} and D_H corresponding to highest CIM accuracy as optimal values.

3.5 Hardware Design Considerations

Our proposed approach requires minimal design changes at hardware level. In the q1 phase, it only modifies the fractional length of weights without altering the word length. Despite the new fraction length, fixed-point value still appears as an integer with unchanged word length to the hardware, due to implicit nature of radix point. Hence, no hardware modifications are needed for the q1 phase. Moving to the q2 phase, we need configurable truncation logic rather than a fixed implementation. This is because the truncation bit positions depend on the weight fractional length. Implementing this requires only an offset register and multiplexing logic, leading to a very minor design change. Moreover, the offset register can be configured post-fabrication to adapt to various workloads or networks. Hence, our quantization method provides seamless integration with existing hardware architectures.

4 Simulation Results

4.1 Setup

We have developed a Python-based framework to simulate the neural network inference on CIM hardware. It leverages in-situ multiply-accumulate (IMA) unit described in [1, 34]. The energy and area data for various IMA components is acquired from [34]. To obtain power and area for the modified truncation logic required by our quantization approach, we performed RTL synthesis using Cadence Genus with TSMC 40nm technology. Our simulations consider memristors with 2-bit capacity. The memristor device parameters and conductance variation data are obtained from [31], which presents experimental results on real RRAM devices. We consider the following three datasets for evaluation:

- **Street view house numbers (SVHN)** [28]: It contains real-world images of house numbers taken from Google Street View, presenting 10 classes for digits 0 to 9.
- **CIFAR-10** [23]: It contains 10 different classes such as airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.
- **CIFAR-100** [23]: It has 100 classes that can be grouped into 20 superclasses. We consider its classification with 100 classes, as it is a complex task involving similar yet distinct inputs.

We evaluate these datasets by one-to-one mapping with three distinct neural networks Alexnet [24], VGG [37], and ResNet [18]. We need to modify these networks as their original versions target 224×224 images, while our datasets contain 32×32 images. These modified networks are described as follows. (Here, nCm denotes n convolution filters of size m with ReLU, standalone numbers indicate neurons in fully connected layers with ReLU (no ReLU in the last layer), Residual denotes a cascade of nCm blocks as described in [18] and MP represents maxpooling.)

- **Modified Alexnet for SVHN**: Image ⇒ 32c5 ⇒ MP ⇒ 32c3 ⇒ MP ⇒ 64c3 ⇒ 64c3 ⇒ 128c3 ⇒ MP ⇒ flatten ⇒ 256 ⇒ 128 ⇒ 64 ⇒ 10.
- **Modified VGG for CIFAR-10**: Image ⇒ 64c3 ⇒ MP ⇒ 128c3 ⇒ MP ⇒ 256c3 ⇒ 256c3 ⇒ MP ⇒ 512c3 ⇒ 512c3 ⇒ MP ⇒ flatten ⇒ 512 ⇒ 512 ⇒ 10.
- **Modified ResNet for CIFAR-100**: Image ⇒ 64c3 ⇒ 128c3 ⇒ MP ⇒ Residual (128c3, 128c3) ⇒ 256c3 ⇒ MP ⇒ 512c3 ⇒ MP ⇒ Residual (512c3, 512c3) ⇒ MP ⇒ flatten ⇒ 100

We first train these networks using PyTorch [29] in software (in a hardware-unaware manner). We achieve floating point baseline accuracy of 91.65% for SVHN, 88.88% for CIFAR-10 and 66.73% for CIFAR-100. Trained weights are fed into our Python framework. It first quantizes weights to 8 bits and activations to 16 bits, and then evaluates hardware inference accuracy.

4.2 Neural Network Accuracy

In this experiment, we determine the optimal number of fractional bits for the modified AlexNet network on the SVHN dataset. This analysis also indirectly gives us the number of high-order bits to be discarded from the full precision output, as detailed in Section 3.4. We apply the methodology in Algorithm 1 to explore weight fractional lengths (W_{FL}) ranging from 6 to 11 bits. This results in the following fixed-point formats for weights: (8,6), (8,7), (8,8), (8,9), (8,10), (8,11), and (8,12). When W_{FL} exceeds the word length in a fixed-point format, its fractional part includes implicit zeros after the radix point. For instance, a fixed-point number in (8,9) format is represented as 0.0bbbbbbbb, where 'b' denotes a variable bit.

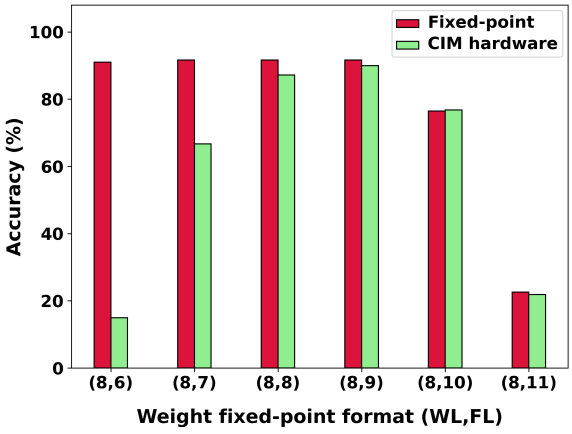
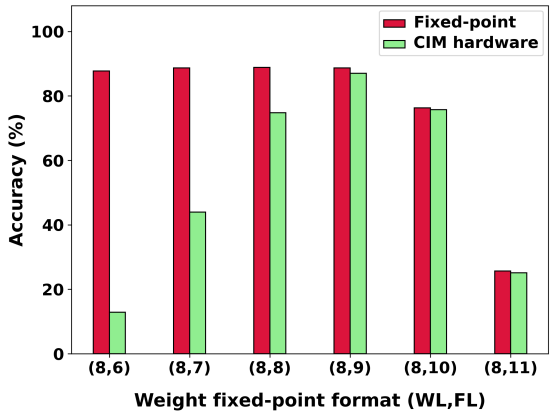
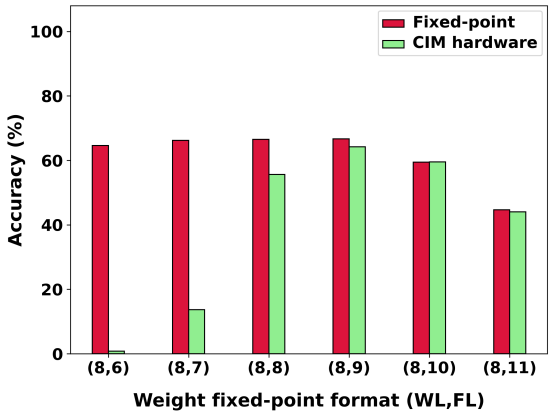


Figure 9: Determining optimal weight fraction length for high CIM inference accuracy. WL denotes word length, while FL denotes fraction length of weights.



(a) Modified VGG with CIFAR-10.



(b) Modified ResNet with CIFAR-100.

Figure 10: Demonstrating the scalability of proposed quantization approach using complex datasets and neural networks. WL denotes word length of weights, while FL denote fraction length of weights.

The ideal fixed-point accuracy (red bars) for various W_{FL} values is shown in Fig. 9. Accuracy remains high as W_{FL} increases from 6 to 9. This is because these fixed-point formats effectively represent weights and provide sufficient numeric precision to capture layer inputs and outputs. Accuracy even improves slightly within this range due to reduced quantization loss ($1 \div 2^{W_{FL}}$). However, it begins to decline beyond a W_{FL} of 9 bits. This occurs because fixed-point formats with higher W_{FL} cannot represent weights effectively and fail to provide adequate precision to capture layer input/output.

The CIM hardware accuracy across these formats (green bars) is also shown in Fig. 9. With a W_{FL} of 6 bits, q1 fails to suppress errors effectively due to the small W_{FL}. This leaves behind excessive residual errors that q2 cannot handle, leading to low accuracy. The accuracy improves with increasing W_{FL}, reaching its peak at W_{FL} of 9 bits. This is because a higher W_{FL} allows q1 to suppress the errors more effectively. This enables q2 to handle the residual errors more adeptly, resulting in nearly error-free outputs. Formats with W_{FL} greater than 9 bits show good error reduction. However, their hardware accuracy is hampered due to insufficient precision for representing quantized weights and capturing layer inputs/outputs. Thus, W_{FL} = 9, corresponding to the weight format (8,9), emerges as the best choice for optimal hardware accuracy.

4.3 Scalability

We evaluate the scalability of our approach by applying it to a modified VGG on CIFAR-10 and a modified ResNet on CIFAR-100. Using Algorithm 1 again, we aim to determine the optimal W_{FL} for high CIM accuracy. This leads to design space exploration with W_{FL} values from 6 to 11, corresponding to fixed-point formats: (8,6), (8,7), (8,8), (8,9), (8,10), and (8,11).

Figure 10 shows that both networks maintain high fixed-point accuracy up to a W_{FL} of 9. This is due to effective weight representation and adequate numeric precision in these formats. It declines for formats with W_{FL} beyond 9 bits, due to reduced effectiveness in weight representation and inadequate numeric precision. CIM hardware accuracy improves with increasing W_{FL} from 6 to 9 bits, thanks to superior error suppression with larger W_{FL}. However, increasing W_{FL} beyond 9 bits does not benefit from good error suppression, as the inadequate numeric precision outweighs this benefit. Thus, a W_{FL} of 9 i.e. weight fixed-point format (8,9) is the optimal choice for both networks. This demonstrates the scalability of our approach, as the trends, insights and effectiveness from Section 4.2 all have successfully extended to these more complex datasets and networks.

4.4 Hardware Performance Evaluation

Table 1 compares the hardware performance of our proposed quantization approach with conventional quantization approach in CIM architectures [11, 34, 39]. As detailed in Section 3.5, we only require a minor change in the output truncation logic design to adjust the truncated bit positions based on W_{FL}. RTL synthesis results for configurable truncation logic show that it consumes just 109 μW power and occupies 343 μm^2 area. Consequently, its overall impact on IMA metrics is minimal, with only a 1.2% increase in energy consumption and a 6.3% increase in area as inferred from Table 1 Thus, our approach incurs very small overheads.

The total energy consumption does not reflect its share used for performing correct computations. To address this, we introduce a new metric called "correct operations per unit energy". It is defined

Table 1: Hardware metrics per in-situ multiply-accumulate unit for conventional and proposed quantization approaches. Hardware accuracy and GOP/J data are for SVHN dataset.

Metric	Conventional approach [11, 34, 39]	Proposed approach
Hardware accuracy (%)	15.94	89.97
Energy (pJ)	3738	3782
Area (μm^2)	21765	23137
Correct operations per unit energy (GOP/J)	43.7	243.6

as the ratio of correct operations to total energy consumption, expressed in the unit Giga operations per joule (GOP/J). The correct operations are calculated by multiplying hardware accuracy (as a fraction) with the total number of operations. Our proposed approach achieves 5.6× correct operations per unit energy compared to the conventional approach, highlighting its effectiveness.

5 Conclusions

We presented a quantization methodology for CIM-based neural networks to achieve high accuracy in the presence of conductance variation. This was achieved by tuning fraction length of weights to suppress errors due to conductance variation. The residual error was reduced by discarding bits containing significant error but less information content. This resulted in error-free computations and a high inference accuracy on CIM hardware. Our proposed quantization approach achieved 5.6× correct operations per unit energy compared to the conventional quantization approach, with very low overheads. This can facilitate effective deployment of CIM-based neural networks in resource-constrained edge-AI environments.

Acknowledgments

This work is partially funded by the European Union, DAIS (Grant No. 101007273), NEUROKIT2E (Grant No. 101112268) and also supported by the TU Delft AI labs program.

References

[1] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R. Stanley Williams, Paolo Faraboschi, Wen Mei Hwu, John Paul Strachan, Kaushik Roy, and Dejan S. Milojicic. 2019. 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 (ASPLOS)*. 715–731.

[2] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Sapan Agarwal, Matthew Marinella, Martin Foltin, John Paul Strachan, Dejan Milojicic, Wen Mei Hwu, and Kaushik Roy. 2020. PANTHER: A Programmable Architecture for Neural Network Training Harnessing Energy-Efficient ReRAM. *IEEE Trans. Comput.* 69, 8 (2020), 1128–1142.

[3] Alessio Antolini, Carmine Paolino, Francesco Zavalloni, Andrea Lico, Eleonora Franchi Scarselli, Mauro Mangia, Fabio Pareschi, Gianluca Setti, Riccardo Rovatti, Mattia Luigi Torres, Marcella Carissimi, and Marco Pasotti. 2023. Combined HW/SW Drift and Variability Mitigation for PCM-Based Analog In-Memory Computing for Neural Network Applications. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 13, 1 (2023), 395–407.

[4] Rajendra Bishnoi, Sumit Diware, Anteneh Gebregiorgis, Simon Thomann, Sara Mannaa, Bastien Deveautour, Cédric Marchand, Alberto Bosio, Damien Deleruyelle, Ian O'Connor, Hussam Amrouch, and Said Hamdioui. 2023. Energy-efficient Computation-In-Memory Architecture using Emerging Technologies. In *International Conference on Microelectronics (ICM)*. 325–334.

[5] Fuxi Cai, Justin M. Correll, Seung Hwan Lee, Yong Lim, Vishishtha Bothra, Zhengya Zhang, Michael P. Flynn, and Wei D. Lu. 2019. A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations. *Nature Electronics* 2, 7 (2019), 290–299.

[6] Chih Cheng Chang, Shao Tzu Li, Tong Lin Pan, Chia Ming Tsai, I. Ting Wang, Tian Sheuan Chang, and Tuo Hung Hou. 2022. Device quantization policy in variation-aware in-memory computing design. *Nature Scientific Reports* 12, 1 (2022), 112.

[7] Gouranga Charan, Abinash Mohanty, Xiaocong Du, Gokul Krishnan, Rajiv V. Joshi, and Yu Cao. 2020. Accurate Inference With Inaccurate RRAM Devices: A Joint Algorithm-Design Solution. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* 6, 1 (2020), 27–35.

[8] Lerong Chen, Jiawen Li, Yiran Chen, Qiuping Deng, Jiyuan Shen, Xiaoyao Liang, and Li Jiang. 2017. Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 19–24.

[9] Pai-Yu Chen, Xiaochen Peng, and Shimeng Yu. 2018. NeuroSim: A Circuit-Level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 12 (2018), 3067–3080.

[10] Pai-Yu Chen and Shimeng Yu. 2019. Technological Benchmark of Analog Synaptic Devices for Neuroinspired Architectures. *IEEE Design & Test* 36, 3 (2019), 31–38.

[11] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory. In *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 27–39.

[12] Intel Corporation. 2024. *Intel Processor Family*. https://www.intel.in/content/www/in/en/products/processors/core.html

[13] NVIDIA Corporation. 2024. *NVIDIA Turing Architecture GPUs*. https://www.nvidia.com/en-in/geforce/turing/

[14] Sumit Diware, Anteneh Gebregiorgis, Rajiv V. Joshi, Said Hamdioui, and Rajendra Bishnoi. 2021. Unbalanced Bit-slicing Scheme for Accurate Memristor-based Neural Network Architecture. In *IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. 1–4.

[15] Sumit Diware, Anteneh Gebregiorgis, Rajiv V. Joshi, Said Hamdioui, and Rajendra Bishnoi. 2023. Mapping-aware Biased Training for Accurate Memristor-based Neural Networks. In *IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. 1–5.

[16] Sumit Diware, Abhairaj Singh, Anteneh Gebregiorgis, Rajiv V. Joshi, Said Hamdioui, and Rajendra Bishnoi. 2023. Accurate and Energy-Efficient Bit-Slicing for RRAM-Based Neural Networks. *IEEE Transactions on Emerging Topics in Computational Intelligence* 7, 1 (2023), 164–177.

[17] Jingyu He, Yucong Huang, Miguel Lastras, Terry Tao Ye, Chi Ying Tsui, and Kwang Ting Cheng. 2023. RVComp: Analog Variation Compensation for RRAM-Based in-Memory Computing. In *28th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 1–6.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*. 1–12.

[19] Weiwen Jiang, Qiuwen Lou, Zheyu Yan, Lei Yang, Jingtong Hu, Xiaobo Sharon Hu, and Yiyu Shi. 2021. Device-Circuit-Architecture Co-Exploration for Computing-in-Memory Neural Accelerators. *IEEE Trans. Comput.* 70, 4 (2021), 595–605.

[20] Norman P. Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Cliff Young, Xiang Zhou, Zongwei Zhou, and David Patterson. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*. 1–14.

[21] Hanul Kim, Mihir Jain, Jun-Tae Lee, Sungrack Yun, and Fatih Porikli. 2021. Efficient Action Recognition via Dynamic Knowledge Propagation. In *IEEE/CVF International Conference on Computer Vision (ICCV)*. 13719–13728.

[22] Wonjoo Kim, Anupam Chattopadhyay, Anne Siemon, Eike Linn, Rainer Waser, and Vikas Rana. 2016. Multistate memristive tantalum oxide devices for ternary arithmetic. *Nature Scientific reports* 6, 1 (2016), 36652.

[23] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report. University of Toronto. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*. 1–9.

[25] Nicola Lepri, Artem Glukhov, and Daniele Ielmini. 2022. Mitigating read-program variation and IR drop by circuit architecture in RRAM-based neural network accelerators. In *IEEE International Reliability Physics Symposium (IRPS)*. 3C.2–1–3C.2–6.

[26] Can Li, Daniel Belkin, Yunning Li, Peng Yan, Miao Hu, Ning Ge, Hao Jiang, Eric Montgomery, Peng Lin, Zhongrui Wang, Wenhao Song, John Paul Strachan, Mark Barnell, Qing Wu, R. Stanley Williams, J. Joshua Yang, and Qiangfei Xia. 2018. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nature Communications* 9, 1 (2018), 2385.

[27] S. R. Nandakumar, Manuel Le Gallo, Christophe Piveteau, Vinay Joshi, Giovanni Mariani, Irem Boybat, Geethan Karunaratne, Riduan Khaddam-Aljameh, Urs Egger, Anastasios Petropoulos, Theodore Antonakopoulos, Bipin Rajendran, Abu Sebastian, and Evangelos Eleftheriou. 2020. Mixed-Precision Deep Learning Based on Computational Memory. *Frontiers in Neuroscience* 14 (2020).

[28] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. 2011. Reading Digits in Natural Images with Unsupervised Feature Learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*. http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems (NeurIPS)*. 1–12.

[30] Giacomo Pedretti, Elia Ambrosi, and Daniele Ielmini. 2021. Conductance variations and their impact on the precision of in-memory computing with resistive switching memory (RRAM). In *IEEE International Reliability Physics Symposium (IRPS)*. 1–8.

[31] Amit Prakash and Hyunsang Hwang. 2016. Multilevel Cell Storage and Resistance Variability in Resistive Random Access Memory. *Physical Sciences Reviews* 1, 6 (2016), 20160010.

[32] Abu Sebastian, Manuel Le Gallo, Riduan Khaddam-Aljameh, and Evangelos Eleftheriou. 2020. Memory devices and applications for in-memory computing. *Nature Nanotechnology* (2020).

[33] Abu Sebastian, Manuel Le Gallo, Riduan Khaddam-Aljameh, and Evangelos Eleftheriou. 2020. Memory devices and applications for in-memory computing. *Nature Nanotechnology* 15, 7 (2020), 529–544.

[34] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 14–26.

[35] Taha Shahroodi, Mahdi Zahedi, Can Firtina, Mohammed Alser, Stephan Wong, Onur Mutlu, and Said Hamdioui. 2022. Demeter: A Fast and Energy-Efficient Food Profiler Using Hyperdimensional Computing in Memory. *IEEE Access* 10 (2022), 82493–82510.

[36] Sunil Shukla, Bruce Fleischer, Matthew Ziegler, Joel Silberman, Jinwook Oh, Vijayalakshmi Srinivasan, Jungwook Choi, Silvia Mueller, Ankur Agrawal, Tina Babinsky, Nianzheng Cao, Chia Yu Chen, Pierce Chuang, Thomas Fox, George Gristede, Michael Guillorn, Howard Haynie, Michael Klaiber, Dongsoo Lee, Shih Hsien Lo, Gary Maier, Michael Scheuermann, Swagath Venkataramani, Christos Vezyrtzis, Naigang Wang, Fanchieh Yee, Ching Zhou, Pong Fei Lu, Brian Curran, Leland Chang, and Kailash Gopalakrishnan. 2018. A scalable multi-TeraOPS core for AI training and inference. *IEEE Solid-State Circuits Letters* 1, 12 (2018), 217–220.

[37] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR)*. 1–14.

[38] Abhairaj Singh, Rajendra Bishnoi, Ali Kaichouhi, Sumit Diware, Rajiv V. Joshi, and Said Hamdioui. 2023. A 115.1 TOPS/W, 12.1 TOPS/mm2 Computation-in-Memory using Ring-Oscillator based ADC for Edge AI. In *IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. 1–5.

[39] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. 2017. PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 541–552.

[40] Zhuoran Song, Yanan Sun, Lerong Chen, Tianjian Li, Naifeng Jing, Xiaoyao Liang, and Li Jiang. 2021. ITT-RNA: Imperfection Tolerable Training for RRAM-Crossbar-Based Deep Neural-Network Accelerator. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 1 (2021), 129–142.

[41] Qiwen Wang, Yongmo Park, and Wei D. Lu. 2022. Device Variation Effects on Neural Network Inference Accuracy in Analog In-Memory Computing Systems. *Advanced Intelligent Systems* 4, 8 (2022), 2100199.