Efficient, Accurate, and Flexible PIM Inference through Adaptable Low-Resolution Arithmetic

by

Tanner Andrulis

B.S., Purdue University (2021)

Submitted to the
Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

Authored by: Tanner Andrulis
Department of Electrical Engineering and Computer Science
May 19, 2023

Certified by : Joel S. Emer
Professor of the Practice, Department of Electrical Engineering and Computer Science
Thesis Supervisor

Certified by : Vivienne Sze
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by : Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Efficient, Accurate, and Flexible PIM Inference through Adaptable Low-Resolution Arithmetic

by

Tanner Andrulis

Submitted to the
Department of Electrical Engineering and Computer Science
on May 19, 2023, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

Processing-In-Memory (PIM) accelerators have the potential to efficiently run Deep Neural Network (DNN) inference by reducing costly data movement and by using resistive RAM (ReRAM) for efficient analog compute. Unfortunately, overall PIM accelerator efficiency and throughput are limited by area/energy-intensive analog-to-digital converters (ADCs). Furthermore, existing accelerators that reduce ADC area/energy do so by changing DNN weights or by using low-resolution ADCs that reduce output fidelity. These approaches harm DNN accuracy and/or require costly DNN retraining to compensate.

To address these issues, this thesis explores tradeoffs around ADC area/energy and develops optimizations that can reduce ADC area/energy without retraining DNNs. We use these optimizations to develop a new PIM accelerator, RAELLA, which can adapt the architecture to each DNN. RAELLA lowers the resolution of computed analog values by encoding weights to produce near-zero analog values, adaptively slicing weights for each DNN layer, and dynamically slicing inputs through speculation and recovery. Low-resolution analog values allow RAELLA to both use efficient low-resolution ADCs and maintain accuracy without retraining, all while computing with fewer ADC converts.

Compared to other low-accuracy-loss PIM accelerators, RAELLA increases energy efficiency by up to 4.9× and throughput by up to 3.3×. Compared to PIM accelerators that cause accuracy loss and retrain DNNs to recover, RAELLA achieves similar efficiency and throughput without expensive DNN retraining.

Thesis Supervisor: Joel S. Emer
Title: Professor of the Practice, Department of Electrical Engineering and Computer Science

Thesis Supervisor: Vivienne Sze
Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgments

Writing this thesis has provided me with a valuable opportunity to look back over the past two years, and it is clear that none of what I have achieved would have been possible without all the wonderful people who have gone out of their way to help me in my journey.

First and foremost, I would like to thank my advisors Vivienne Sze and Joel Emer. They have been everything I could have asked for from mentors: patient with me as I develop my communication and research skills, encouraging when I am in doubt or overwhelmed, and always available to provide precise and constructive feedback. As I have progressed as a researcher, Vivienne and Joel have provided me with countless opportunities to develop, from first learning how to conduct a research project to how to present research, collaborate with others, and mentor. Through constant feedback, support, and dedication to helping me grow, Vivienne and Joel have truly shaped me into the researcher I am today.

I also thank the labmates who helped proofread parts of this thesis and provided invaluable feedback. Fisher, Nellie, Michael, Andy, and Peter, your feedback was invaluable in making my first publication the best it could be. Of course, thank you to my other labmates as well, Keshav, Zih-Sing, Jamie, and Dasong, for giving me presentation feedback. Thank you to all of my labmates for creating a supportive, fun, and always positive lab environment that is a joy to work in.

I would like to thank my various collaborators. Thank you Murat for the great discussions and insight around memristive devices. Thank you Ray for working with me to set up the ADC energy modeling tool. Thank you Anni for helping me to integrate NeuroSim into the modeling framework, and to Lillian for providing your feedback in integrating NVMExplorer. Thank you to the Timeloop team, Angshu,

Jenny, and Po-An, for your discussions on modeling in Timeloop and for your feedback on my presentations.

Thank you to the Intel team, especially Hechen and Ram, for giving me feedback on my modeling works and for helping me to develop new models of architectures in Accelergy and Timeloop. Thank you to the Ericsson team for their discussions on new workloads and feedback on my ADC modeling work.

Finally, I would like to thank all of my friends and family for being an unending source of love and support.

This work was funded in part by Ericsson, MIT Quest, the MIT AI Hardware Program, The Jacobs Presidential Fellowship, and the Siebel Scholar Fellowship.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep Neural Network (DNNs) inference is a popular workload in modern datacenter and mobile/edge applications. DNN inference is used in autonomous navigation, language models, recommendation, and image recognition, among many other use cases. Unfortunately, despite the popularity of DNN inference, the application of DNN inference is limited by high energy and compute costs.

These costs are incurred, in part, as a result of the large tensors used in DNNs. Each DNN may be composed of many layers that compute output tensors using both input tensors and trained weight tensors. A weight tensor used in a layer may contain millions of elements [22], and it is energy intensive to move large weight tensors from off-chip memory [24]. In addition to the challenge of high data movement costs, each DNN layer may incur a high compute cost, with some layers requiring the computation of millions of multiply-accumulate (MAC) operations.

Processing-In-Memory (PIM) is a promising solution to address the energy-intensive data movement and high compute costs of DNN inference. PIM accelerators are systems that run DNN inference by executing MAC operations directly in memory. By

computing in memory [50], PIM accelerators avoid expensive off-chip movement of the DNN weights [81]. Furthermore, in-memory computation can realize analog operations by accessing multiple memory elements in parallel and combining the result in a single wire. Analog computation is efficient, addressing the high compute cost of DNN inference.

PIM accelerators often use Resistive-RAM (ReRAM) devices and ReRAM crossbars [72, 10, 77]. A *ReRAM device* is a memory device that can store data and compute analog MAC operations. A *ReRAM crossbar* is an array of ReRAM devices that can compute analog matrix-vector operations [50]. ReRAM crossbars are arranged into rows and columns. Each row computes a set of parallel MACs that share the same input, and each column accumulates the results of computations that go to the same output. A column may accumulate the results computed by hundreds of ReRAM devices, generating a *high-resolution* result. We define the *resolution* of a computation as the base-two logarithm of the number of unique values the computation may produce.

ReRAM crossbars can compute analog operations efficiently and with high density. Unfortunately, despite these advantages, overall PIM accelerator energy may be dominated by analog-to-digital converters (ADCs) that read computed analog values from crossbars and convert them to digital values. Due to ADC overhead, some PIM accelerators [72, 65] do not significantly improve energy over non-PIM accelerators [8] despite the opportunities in PIM.

Some prior works attempt to reduce this ADC overhead by reducing the resolution of the ADC, which exponentially decreases ADC energy [88]. Architectures often partition, or *slice*, the bits in DNN inputs and weights into multiple lower-resolution slices and compute with different slices in multiple steps [72]. Although sliced arithmetic can use lower-resolution ADCs, ADCs must process the results of

each slice, so *these approaches replace each high-resolution ADC convert with multiple low-resolution ADC converts*, and therefore ADCs still dominate overall energy.

Other PIM accelerators reduce ADC energy, but do so at the expense of DNN accuracy. Some designs prune DNNs [107, 102, 66, 13, 40] to reduce DNN weight count, so we call these designs *Weight-Count-Limited*. They reduce the computation count and ADC converts required, but also introduce accuracy loss. Alternatively, other designs use efficient lower-resolution ADCs to process high-resolution analog values from crossbars [37, 10, 12]. We call these designs *Sum-Fidelity-Limited* as the reduced resolution reduces output fidelity and introduces error. These architectures may requantize, or change the values of DNN weights, such that DNN can tolerate ADC resolution limitations. This still results in accuracy loss.

To reduce this accuracy loss, both Weight-Count-Limited and Sum-Fidelity-Limited architectures retrain DNNs. This is a problem; DNN training has a very high computational cost [60], can require cumbersome hyperparameter tuning to achieve high accuracy [28], and may be impossible if the training data is private [84, 68]. Furthermore, cutting-edge DNNs often require particular training schemes [11], which may not be compatible with the retraining scheme required by an architecture.

To avoid accuracy loss without imposing retraining, we look at fidelity limitations. We define *fidelity* as the ability of the ADC to represent the full distribution of computed analog values. Architectures lose fidelity and generate errors when the computed analog value resolution is higher than the ADC resolution. Each DNN produces many distributions of analog values, and prior Sum-Fidelity-Limited approaches modify DNNs to reshape these analog value distributions to fit a resolution-limited ADC range. In contrast, we observe that we can reshape analog value distributions by changing how we realize computations, rather than by changing the DNN.

Using this key insight, this thesis proposes the RAELLA architecture to enable efficient PIM inference without retraining. RAELLA modifies arithmetic and slicing, shaping computed value distributions to produce low-resolution analog results. This allows RAELLA to use efficient low-resolution ADCs while maintaining high fidelity and low DNN accuracy loss.

The main contributions of this thesis are the following. The first contribution is a tool for analysis of PIM accelerators, while the latter three contributions are optimizations that, informed by this analysis, are able to reduce ADC area and energy without retraining.

- An analysis and breakdown of ADC energy factors through the Titanium Law. Given the importance of reducing ADC energy in PIM accelerators, it is essential to understand the available tradeoffs and how they affect ADC energy. We introduce the Titanium Law, which breaks down ADC energy into its constituent terms. Through the lens of the Titanium Law, we review prior works and analyze how they make tradeoffs to reduce ADC energy. The Titanium Law is discussed in Section 2.8.

- Center+Offset encoding to accumulate more values in the analog domain while keeping small, low-resolution sums. Specifically, RAELLA shifts DNN weights to equalize the average magnitude of the positive and negative weight slices in each crossbar column. As analog-domain calculations are accumulated, positive and negative results negate to produce near-zero sums that can be converted with high fidelity. Center+Offset is discussed in Section 4.1.

- Adaptive Slicing of DNN weights at compilation time to balance density, efficiency, and fidelity. Storing more bits in each ReRAM device is denser and more

efficient but can create higher-resolution analog values. For each DNN layer, RAELLA adapts the number of ReRAM devices per weight and the number of bits in each ReRAM device. This enables RAELLA to use the densest and most efficient number of slices possible while keeping computed analog values low-resolution. Adaptive Weight Slicing is discussed in Section 4.2.

- Dynamic Slicing of DNN input activations at runtime for both efficient and high-fidelity computation. RAELLA speculates with an efficient approach that tries to use more bits in each input slice. RAELLA detects and recovers from incorrect results using a less efficient approach that processes inputs over more slices using fewer bits each. This allows RAELLA to further reduce the number of ADC conversions without reducing fidelity. Dynamic Input Slicing is discussed in Section 4.3.

The latter three optimizations are integrated into the RAELLA accelerator, which reduces ADC area and energy while maintaining low DNN accuracy loss and without modifying DNNs. Compared to other low-accuracy-loss PIM accelerators [72], RAELLA can both lower ADC resolution and run DNNs with up to 14× fewer ADC conversions without sacrificing fidelity.

We evaluate RAELLA on seven representative DNNs against three state-of-the-art PIM accelerators. Compared to other low-accuracy-loss PIM accelerators, RAELLA improves energy efficiency by up to 4.9× (geomean 3.9×) and throughput by up to 3.3× (geomean 2.0×). Compared to Weight-Count-Limited and Sum-Fidelity-Limited accelerators that require DNN retraining to recover accuracy, RAELLA provides similar efficiency and throughput while avoiding expensive DNN retraining.

Portions of the work in this thesis will be published in ISCA 2023 [3].

Chapter 2

Background

In this chapter, we discuss important concepts in Processing-In-Memory (PIM) for Deep Neural Network (DNN) inference acceleration. We first discuss the design space around PIM, giving a review of the current literature and proposed methods that bring memory and compute closer together. Next, we discuss memory technologies that can be used for PIM. After that, we show how PIM is realized in detail, discussing DNN inference and how it can be accelerated in PIM designs.

Once we have established an understanding of PIM accelerators, we focus on analog-to-digital converters (ADCs) that limit PIM accelerator efficiency and throughput. We introduce the Titanium Law that shows how to calculate ADC energy, then we use the Titanium Law to discuss prior works. We will discuss in detail two types of approaches, Sum-Fidelity-Limited and Weight-Count-Limited, that reduce ADC area and energy by modifying DNNs.

Finally, we discuss how the energy of ADCs scales with various architectural attributes and introduce the motivation behind the RAELLA accelerator.

2.1 Computing Closer to Memory

It can be more energy intensive to load data from off-chip memory than to compute with the data [24], so many DNN accelerators use on-chip buffers to store data to keep the data closer to processing. These accelerators can reduce the energy of data movement through reuse; fetching the operand once, storing it in buffers, and reusing it in many computations to amortize the cost of the original access. Alternatively, an increasing number of architectures explore co-locating compute with large memories. This scheme can eliminate the long-distance movement of some data entirely.

Various methods have been used to co-locate compute and memory. *Compute-near-memory* refers to methods that place large memories, such as DRAM, and digital processors closer together. This is often done via closely-coupled chiplets or by stacking compute and memory chips [61, 17, 29]. Other compute-near-memory works may use embedded DRAM or Resistive RAM (ReRAM) devices to place large memories directly on-chip [86, 8]. Compute-near-memory chips can use traditional digital processing, which increases the flexibility of their computation. However, they are limited in that despite being closer to memory, they must still fetch data from the large memory, incurring data movement costs. Additionally, they are limited in that their digital processing does not address the high compute cost of DNN inference.

Alternatively, *compute-in-peripheral* methods further reduce data movement by using vector operations that compute directly in the peripherals of large memories [1, 23]. We distinguish the memory elements themselves (*e.g.*, SRAM bitcells) from the memory peripherals (*e.g.*, sensing circuitry, addressing, and any circuitry that is adjacent to the memory elements). By computing in the peripheral circuitry, these architectures minimize the distance traveled by data, processing it immediately adjacent to memory arrays. Additionally, this closeness to memory enables compute-

in-peripheral architectures to realize highly-parallel operations due to the parallelism available in large memories. This can partially address the high compute cost of DNN inference. Compute-in-peripheral architectures are still limited, however, in that they must read data from the memory to the peripheral, so they do not fully eliminate reads of large memories.

Finally, some works move computation into the memory elements themselves. These works, referred to as *compute-in-memory* [39, 74, 107, 37, 108, 65, 78, 7, 53, 75, 106], access multiple memory devices in at once, generating analog signals from the combination of those accesses. This allows them to perform parallel multiply-accumulate (MAC) operations (the primary operation of many DNN layers) directly within memory. Analog compute-in-memory can address the challenges of high data movement costs and high compute costs. Data movement is reduced as DNN weights can be kept fully stationary (*i.e.*, used for computation without moving them from memory). Furthermore, compute-in-memory can address high compute costs. Analog computation is more efficient than digital computation [72] and can utilize memory parallelism to compute many operations in parallel [10].

Although analog computation is efficient, it is difficult to realize fully analog DNNs. This is because analog computations introduce noise due to device variation, temperature, and other nonidealities, and this noise accumulates through each DNN layer [44]. Most compute-in-memory works use a hybrid analog/digital method that uses analog computation within memory and digital communication outside of memory. These works use analog/digital converters to convert between the analog and digital domains [107, 37, 108, 65, 78, 7, 53, 75, 106].

In this thesis, we focus on architectures employing this last compute-in-memory method, where analog processing is done within memory, analog/digital converters are used to communicate with digital portions of the chip, and the target workload

is DNN inference. We use the term *Processing-In-Memory (PIM)* to describe these architectures. This specific method is attracting interest because it can exploit efficient analog computation in memory. Furthermore, analog/digital interfaces permit the rest of the system to be designed with more robust digital logic.

One limitation of PIM accelerators is that while analog processing is dense and efficient, PIM accelerators require large, energy-intensive analog-to-digital converters (ADCs) to convert computed analog values back to the digital domain. The area and energy of these ADCs limit PIM systems, so this thesis proposes a variety of strategies to reduce ADC area and energy. Another limitation is that analog computation, unlike digital computation, is vulnerable to noise. The strategies introduced in this thesis allow DNNs to tolerate more noise without loss of accuracy.

We refer the reader to several sources for further reviews of the design space around computing near and in memory [76, 73, 81, 98, 51, 6].

2.2 Technologies for Analog Computation

PIM has been used to accelerate DNNs using a variety of memory technologies. Among them are existing SRAM and DRAM technologies, as well as more novel resistive RAM (ReRAM) technologies. Table 2.1 enumerates some of the differences between these technologies. In this section, we use the word *cell* to refer to the smallest unit of memory storage in a given technology. For example, a *DRAM cell* is the standard capacitor+transistor circuit that stores a bit of data in DRAM.

SRAM-based PIM is popular because SRAM is easy to fabricate in existing logic processes, is reliable, and is relatively low energy to read/write [63]. SRAM is limited in that SRAM cells are larger than DRAM/ReRAM cells, requiring six or more transistors per cell, and SRAMs can only store one bit per cell. Large cell size and

one-bit-per-cell capacity limit the overall density of SRAM [63]. Additionally, SRAM cells must be protected from large voltage swings (*i.e.*, caused by computations using multiple SRAM cells) that may overwrite SRAM cells and corrupt stored data. To prevent this effect, additional read transistors may be added to each SRAM cell to decouple write and compute operations at the cost of additional chip area [51].

DRAM-based PIM can achieve a storage density higher than that of SRAM-based PIM because DRAM cells are simpler and smaller than those of SRAM. DRAM has the additional challenge that DRAM reads erase stored values. Additional hardware, such as added transistors for each cell, may decouple read and compute operations to enable computation without erasing stored values [30]. However, these come at the cost of increasing cell size and reducing storage density. Another limitation is that DRAM cells are volatile (*i.e.*, they lose state over time) and they must be refreshed periodically even while powered on.

ReRAM-based PIM is gaining increasing attention because of the high density and nonvolatility of ReRAM devices. A ReRAM device acts as a programmable resistor that can be programmed to multiple conductance ($1/Resistance$) values. This ability allows a single ReRAM device to store values and realize a memory cell. Furthermore, the conductance of ReRAM devices can be programmed to multiple levels, allowing each ReRAM cell to store multiple bits. Small, multi-bit cells enable highly-dense storage with ReRAM devices. Additionally, ReRAM devices are nonvolatile (*i.e.*, they save state even when powered off). Nonvolatility is attractive for edge devices that may be turned on/off intermittently [63, 37]. ReRAM devices have the limitation of high write energy [63], as well as reliability challenges such as drift (stored values changing over time), leakage current (zero-programmed cells leaking nonzero current), and limited endurance (devices wear out after some number of writes) [12, 41, 103, 63]. ReRAM devices have been fabricated in a variety of

	SRAM	DRAM	ReRAM
Write Energy	**Very Low**	**Low**	**Very High**
Cell Size	**Large**	**Small**	**Small**
Bits per Cell	**1**	**1**	**1-5**
Density	**Low**	**Medium**	**High**
Write Endurance	**Unlimited**	**Unlimited**	**Limited**
Destructive Read	**If Large Voltage Swing**	**Yes**	**No**
Volatility	**Nonvolatile If Powered**	**Needs Refresh**	**Nonvolatile**

Table 2.1: Comparison of SRAM, DRAM, and ReRAM properties for PIM. SRAM devices have low read/write energy but also have low density. DRAM devices have higher density, but are volatile and must be refreshed periodically, and reads are destructive (eliminate stored data). ReRAM devices are nonvolatile and high density, but have a high write energy and limited endurance.

technologies, including STT-RAM, RRAM, and PCM [63]. These technologies are rapidly developing as ReRAM is a relatively new technology compared to DRAM and SRAM [63].

In this thesis, we use ReRAM-based PIM, though the techniques described here can be applied to architectures using any of these devices. We use the multi-bit storage ability of ReRAM devices to perform Adaptive Weight Slicing in Section 4.2. To perform similar multi-bit operations with SRAM or DRAM we can use circuitry that performs weighted sums over multiple one-bit cells [91].

2.3 Deep Neural Network (DNN) Inference

Modern DNNs often utilize convolutional and fully connected layers, which can be accelerated by accelerating tensor (multi-dimensional matrix) operations. The tensor operations of fully-connected and convolutional layers are shown in Figure 2-1. Fully-connected layer operations can be realized with a series of vector-vector mul-

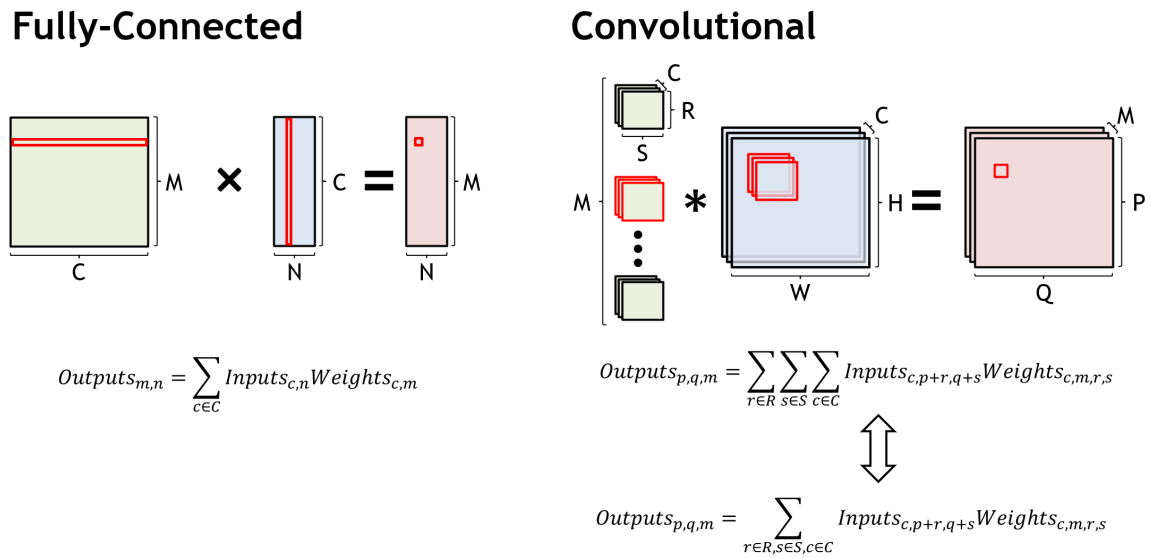

Figure 2-1: Fully-Connected and Convolutional DNN Layer Arithmetic. Left shows Fully-Connected and right shows Convolutional. The layer uses weights (green) and inputs (blue) to compute outputs (red). The convolution operation shows two equations: top is a classical sum, while bottom is the same operation unrolled into a single sum. The red-outlined segments show a single output and the corresponding inputs/weights required to compute that ouptut. The computation of each output requires a significant number inputs, weights, and MAC operations.

tiplications, while convolutional layers use sliding-window operations.

To accelerate convolutional sliding-window operations using PIM, we must realize them using vector-vector multiplications. To do so, notice that the three sum operations in the convolutional layer can be combined into a single sum operation shown in the bottom right of Figure 2-1. This single sum can be expressed as a vector-vector operation. To compute different entries in the output, we then "slide" the window by performing this vector-vector operation with different windows of the layer input or different sets of the layer weights.

The core computation of the vector-vector operation is a multiply-accumulate (MAC) between an input value and a weight value. We call the results of these

computations *partial sums* or *psums*, as they can be thought of as partially-computed outputs. Each layer output may require the computation of many psums from MAC operations using inputs and outputs.

DNNs may have billions of MAC over millions of weights [22]. This makes PIM an attractive choice for DNN inference acceleration. PIM can operate directly in weight memory to reduce data movement [10] and use Resistive RAM (ReRAM) for dense and efficient analog computing. Furthermore, DNNs have some unique advantages that can be used to amortize challenges in PIM. PIM analog computation is inherently noisy, and DNNs have natural noise tolerance.

2.4 DNN Quantization

While DNNs are generally trained using high-resolution floating-point numbers, DNN inference can be run using lower-resolution, easier-to-process integers. We define *quantization* as the procedure of transforming a DNN to use operands of a specific resolution.

In this thesis, we focus on 8-bit (8b) linear per-channel quantization, where DNN inputs and weights are quantized to 8b and DNN psums are quantized to 16b. DNNs quantized in this manner are widely available and can achieve high accuracy [69, 54, 109, 33, 38].

Using 8b linear per-channel quantization, we can map floating-point inputs and weights to 8b integers using the following equation [59]:

$$X_{Quantized} = round(\frac{X_{Floating\ Point}}{Quantization\ Scale} + Quantization\ Zero) \tag{2.1}$$

The *quantization zero* and *quantization scale* are unique values calculated for

each DNN weight channel (*i.e.*, a quantization zero and scale is shared by all weights with the same M value in Figure 2-1). DNN inputs also have a quantization zero and scale, though these values are often unique per-tensor (*i.e.*, shared by all inputs to a DNN layer) rather than per-channel.

In some cases, the quantization zero and scale can be calculated in one pass [54]. In other cases, they require testing the DNN with a set of inputs [38].

Quantization changes DNN weights, and therefore may cause accuracy loss. To recover accuracy loss after quantization, DNNs can be fine-tuned, or retrained using a subset of the training data.

Later, we will learn of architectures that require specific quantization parameters in their computations. These architectures will *requantize* DNNs, or quantize again while placing additional restrictions on quantization zero, quantization scale. They may add other architecture-specific restrictions to DNN operands as well.

2.5 PIM with ReRAM

The functional unit of PIM systems is the ReRAM crossbar. ReRAM crossbars accelerate DNN layers by computing dense in-memory matrix-vector multiplications. Furthermore, ReRAMs are small and offer high storage density [63]. This density allows ReRAM-based systems to store and run on-chip pipelines that compute DNN layers sequentially [72, 77] without costly accesses to off-chip memory [81]. A disadvantage of ReRAM is high write energy [50]. The cost of writes is amortized in inference as ReRAM is nonvolatile, so written weights can be reused for many inferences [72].

Fig. 2-2 shows a basic 2×2 matrix-vector multiplication executed on a 2×2 ReRAM crossbar. A matrix of weights W is programmed in the ReRAMs. Elements

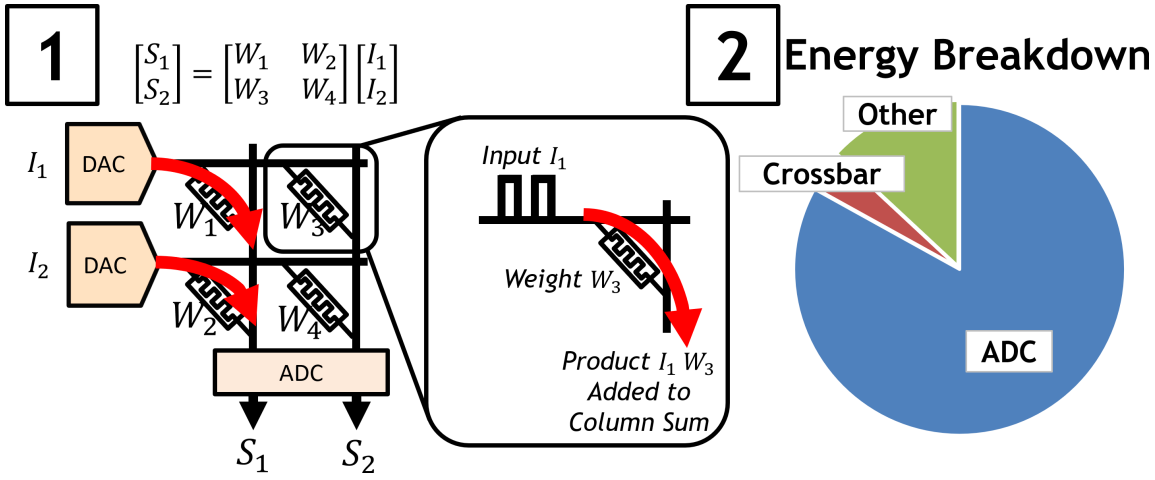

Figure 2-2: Basic PIM crossbar. 1 2×2 MVM. Each operand is a single slice. 2 Energy breakdown of an ISAAC-based design.

of the input vector I are fed to digital-to-analog converters (DACs), which convert the inputs to analog values. Products are accumulated in each column to produce analog *column sums*, which are converted by an analog-to-digital converter (ADC) to produce the digital result S.

In this thesis, we use a temporal representation for inputs, where each weight is programmed to a ReRAM device as a conductance, each input value to a row is encoded as a time value, and each ReRAM device realizes the product of an input and weight as a charge through $Charge_{Device} = Conductance_{Device} \times Time_{Row} \times V_{Read}$. The read voltage V_{Read} is a constant. The charges generated by all devices in a column are summed by Kirchoff's law to produce a column sum as a single charge, following $Charge_{Column} = \sum_{Device \in Column} Charge_{Device}$.

ReRAMs have been shown to be programmable with up to 5b of resolution [2] and 512×512 crossbars have been shown to compute up to 8b of resolution column sums [25] under analog noise limitations. These resolution limits necessitate *slicing* to enable computation of higher-resolution DNN layers.

2.6 Arithmetic with Slices

ReRAM devices may store as many as five or as few as one bit depending on the technology [63]. To run 8b DNN inference using these lower-resolution devices, PIM architectures partition, or *slice*, input and weight bits into *input slices* and *weight slices*. A slice is a subset of bits from an operand, and multiplying two slices yields a *sliced product*.

There are two types of slicing. Temporal slicing processes slices in separate cycles (*e.g.*, bit-serial being the extreme with one bit per slice) and spatial slicing processes slices in separate ReRAMs across parallel crossbar columns. In most PIM accelerators that slice, temporal slicing is used for inputs and spatial slicing is used for weights. We refer to a vector of weights and their slices as a *weight filter* if they are mapped to the same set of columns in one crossbar and they contribute to one dot product for a DNN layer.

Table 2.2 shows an example of sliced arithmetic. Each weight slice is mapped spatially to one crossbar column, while each input slice is processed temporally in one cycle. For each column and cycle, an ADC converts the column sum. The result is shifted and added digitally, allowing PIM architectures to calculate full 16b psums despite low-resolution analog limitations [72, 20].

Table 2.2 shows tradeoffs relating to slicing. Many costs increase with more slices: each additional input slice increments $Cycles/Input$ while each additional weight slice increments $Columns/Weight$. $ADC\ Converts$ scales with the product of input and weight slice counts. The benefit of more slices is that we can use fewer bits per slice, thus reducing MAC resolution and required ADC resolution. We can also decrease $ADC\ Converts$ by using larger crossbars that accumulate more analog values across more rows, but this also increases the required ADC resolution.

Dot Product: 2b input $i_h i_l \cdot$ 2b weight $w_h w_l$					
Sliced Input			✓		✓
Sliced Weight				✓	✓
Cycle	Column				
1	1	$i_h i_l \cdot w_h w_l$	$i_h \cdot w_h w_l$	$i_h i_l \cdot w_h$	$i_h \cdot w_h$
1	2	-	-	$i_h i_l \cdot w_l$	$i_h \cdot w_l$
2	1	-	$i_l \cdot w_h w_l$	-	$i_l \cdot w_h$
2	2	-	-	-	$i_l \cdot w_l$
Bits/MAC		4	2	2	1
Converts/MAC		1	2	2	4

Table 2.2: How Slicing Works & Tradeoffs. A 2b input/weight are multiplied and each may be sliced into two 1b slices. High and low order bits are i_h, w_h and i_l, w_l. Each column/cycle computes the sliced product shown. More slices reduce bits/slice and bits/MAC, permitting a cheaper, lower-resolution ADC. However, cycles, columns, and ADC converts are needed to process each slice. More slices increase ADC Converts/MAC.

2.7 ADCs Limit PIM Accelerators

Fig. 2-2 shows the power breakdown of an 8b PIM architecture based on the foundational PIM accelerator ISAAC [72]. PIM crossbars are dense and efficient, but are limited by ADC area and energy. Crossbars can compute 8b MACs with $<$ 100fJ, but overall efficiency is limited by ADCs. Crossbars are dense, but architectures can spend five times [37] to fifty times [72] times more area on ADC than crossbars. Crossbars can compute with high parallelism, scaling to 1024 rows [48], but the area and energy of ADCs scale exponentially with resolution [88]. Prior work has been limited to as few as 16 activated rows [102] to reduce column sums and ADC resolution requirements.

As ReRAM is dense and low power, you will see that RAELLA trades off more ReRAM for lower-resolution ADCs. Furthermore, by reducing resolution, we use

more crossbar rows/columns with less ADC area/energy scaling. This higher parallelism yields higher throughput and efficiency for the full RAELLA accelerator.

2.8 Reducing ADC Energy

To run efficient PIM inference, we must reduce ADC area and energy. To do so, we present the Titanium Law of ADC energy.[1] While ADC energy is the focus here, a similar analysis can be performed for area by substituting *Converts/MAC* with *#ADCs/Throughput*. Table 2.3 shows the Titanium Law equation for ADC energy and breaks down its factors. ADC energy is the product of four terms:

- *Energy/Convert* is determined by ADC efficiency and scales exponentially with ADC resolution [88].[2]
- *Converts/MAC* is determined by the number of crossbar rows, input slices, and weight slices.
- *MACs/DNN* is determined by the DNN workload.
- *1/Utilization* corresponds to how many crossbar rows are used by the DNN. A utilization of one means all rows used.

Given these factors, Table 2.3 shows how to reduce ADC energy by changing hardware attributes. First, notice the tradeoff generated by *Energy/Convert* and *Converts/MAC* in the first/second rows of the table. Although it may seem that slicing and resizing the crossbar can directly reduce ADC energy, this approach has limited benefits. This is because, to reduce *Converts/MAC*, we must either (1)

[1] Inspired by the Iron Law [93] and titanium-based ReRAM devices [19].

[2] *Energy/Convert* can also be reduced with clever new ADC designs, but there is an efficiency limit [52] due to analog noise. This requires innovations on both the ADC and architecture sides.

The Titanium Law: $\frac{ADCEnergy}{DNN} = \frac{Energy}{Convert} \times \frac{Converts}{MAC} \times \frac{MACs}{DNN} \times \frac{1}{Utilization}$

Term	Hardware Attribute	How to Reduce	Tradeoff	Consequence
Energy/Convert	ADC Resolution	Reduce ADC Resolution	Fewer Crossbar Rows or Bits/Slice	High *Converts/MAC*
			Ⓢ Fidelity Loss, Psum Errors	Ⓢ**Accuracy Loss or Retraining**
Converts/MAC	Crossbar Rows	Increase Crossbar Rows or Bits/Slice	High-Resolution ADC	High Energy/Convert
			Ⓢ Fidelity Loss, Psum Errors	Ⓢ**Accuracy Loss or Retraining**
MACs/DNN	# Weights	Prune/Reshape Weights	Ⓦ Eliminated/Changed Weights	Ⓦ**Accuracy Loss or Retraining**
1/Utilization	Mapping	Improve Mapping	Flexibility Cost, Utilization ≤ 1	Limited Benefits

Table 2.3: The Titanium Law of ADC energy and how to reduce ADC energy components. Of the possible consequences, three are ineffective, and three cause accuracy loss or require DNN retraining. Sum-Fidelity-Limited architectures choose Ⓢ marked cells and Weight-Count-Limited architectures choose Ⓦ marked cells.

increase the crossbar rows and compute more sliced products per ADC convert, (2) increase bits per weight slice, which reduces the number of columns needed to store each weight and reduces the number of ADC converts needed to process each column, or (3) increase bits per input slice, which reduces the number of cycles required and ADC converts to process column sums over all cycles. The limitation, however, is that in all cases we will accumulate larger and higher-resolution column sums. To preserve fidelity, a higher-resolution ADC is needed, which increases *Energy/Convert* and negates our benefits. The converse is true for reducing *Energy/Convert*; preserving high fidelity requires increasing *Converts/MAC*.

The final column of Table 2.3 shows the consequences of reducing each of the Titanium Law terms. Of the six consequences, three are ineffective for reducing ADC energy. *Converts/MAC* and *Energy/Convert* trade off with each other. *1/Utilization* cannot be reduced below one.

Architecture	High ADC Area/ Energy	Limits Weight Count	Fidelity Loss	High Accuracy Loss or Retraining
ISAAC [72]	**Yes**	-	-	**No**
AtomLayer [65]	**Yes**	-	-	**No**
PUMA [4]	**Yes**	-	-	**No**
FORMS [107]	**No**	**Yes**	-	**Yes**
SRE [102]	**No**	**Yes**	-	**Yes**
ASBP [66]	**No**	**Yes**	-	**Yes**
PIM-Prune [13]	**No**	**Yes**	-	**Yes**
ReCom [26]	**No**	**Yes**	-	**Yes**
Learning the Sparsity [40]	**No**	**Yes**	-	**Yes**
TinyADC [106]	**No**	**Yes**	-	**Yes**
ReRAM-Sharing [79]	**No**	**Yes**	-	**Yes**
TIMELY [37]	**No**	-	**High**	**Yes**
PRIME [10]	**No**	-	**High**	**Yes**
CASCADE [12]	**Yes**	-	**High** [3]	**No**
BRAHMS [78]	**No**	-	**High**	**Yes**
MErging the Interface [36]	**No**	-	**High**	**Yes**
RAELLA	**No**	-	**Low**	**No**

Table 2.4: Comparison to prior works. Previous approaches pay high ADC area/energy or use strategies that cause DNN accuracy loss, requiring retraining to recover.

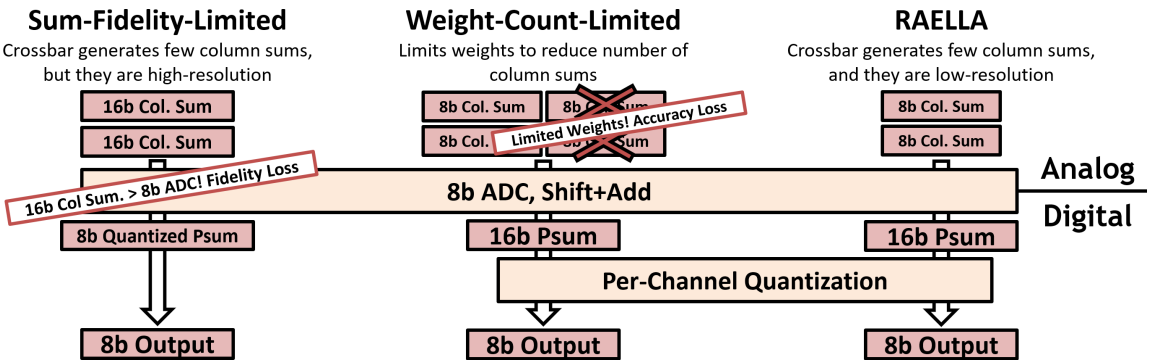

Figure 2-3: Loss-causing architectures alongside RAELLA. Although they decrease *Converts/MAC*, Sum-Fidelity-Limited architectures lose fidelity at the ADCs and force hardware-restricted quantization. Weight-Count-Limited architectures limit DNN weights. RAELLA's arithmetic and slicing strategies maintain high fidelity with low *Converts/MAC*.

Architectures that reduce the ADC energy choose options that end in the consequence **Accuracy Loss or Retraining**. Fig. 2-3 shows how these architectures change DNN operands and lose accuracy. Weight-Count-Limited architectures, in the Ⓦ-marked cells of Table 2.3, prune/reshape DNN weights to lower *MACs/DNN*. Unfortunately, changing weights can cause DNN accuracy loss unless the DNN is retrained. On the other hand, Sum-Fidelity-Limited architectures, in the Ⓢ-marked cells, use more rows, more bits per input/weight slice, and low-resolution ADCs to reduce both *Converts/MAC* and *MACs/DNN*. But because they generate large, high-resolution column sums and use low-resolution ADCs, they lose column sum fidelity. This creates errors in outputs and causes accuracy loss unless the DNN is requantized and retrained.

Counterintuitively, 8b ADCs are not always sufficient for 8b-quantized DNNs in

[3]CASCADE, despite being Sum-Fidelity-Limited does not retrain DNNs as they use higher-resolution 16b computation relative to the 8b used by RAELLA. However, they must compute 4× the total sliced products for 16b operations relative to 8b operations, resulting in a still-high ADC energy. We could scale CASCADE 8b to match RAELLA, but DNNs would then need to be retrained to handle CASCADE's fidelity loss.

Sum-Fidelity-Limited architectures. Psums are 16b after MACs of 8b inputs and weights, and high-accuracy linear quantization strategies need the full 16b psum range [109, 94, 69], so we would like all 16b to have high fidelity. Sum-Fidelity-Limited architectures may generate $> 8b$ column sums and capture them with an 8b ADC. When the ADCs of these architectures lose bits from column sums, they lose bits from the overall psum. This limits high-accuracy quantization strategies to using only the subset of bits that the hardware calculated, rather than the full 16b. This hardware-enforced limitation can cause accuracy loss.

2.9 Sum-Fidelity-Limited Approaches

Like traditional digital accelerators, PIM accelerators may trade off efficiency, throughput, flexibility/programming ability, and area, among many other characteristics. Additionally, PIM accelerators introduce a new tradeoff, fidelity, that deserves in-depth discussion. Sum-Fidelity-Limited approaches use this tradeoff to reduce the area and energy of the ADC.

We define fidelity as the ability of the ADC to represent the full distribution of computed analog values. Architectures often operate with limited fidelity to reduce the area and energy of ADCs, which consume a significant portion of PIM accelerator area/energy. Higher-resolution ADCs are exponentially larger and more energy-intensive than low-resolution ADCs, with area and energy approximately doubling for each additional bit of resolution [89]. With this in mind, reducing ADC resolution can be an effective method to reduce the energy and area of ADCs.

One way to reduce ADC resolution is to exploit the properties of quantized DNNs. A common characteristic of 8b quantized DNNs is that while computed psums may be more than 16b of resolution, they will later be quantized into 8b values. The

quantization procedure often involves dividing the psums by large values on the order of 2^8 to 2^{20} [47, 5], which effectively truncates, or removes, the lowest eight to twenty bits of the psum.

Given that these low-order bits are truncated, it is possible to truncate them earlier in the computation pipeline. This is how Sum-Fidelity-Limited architectures gain efficiency; they use low-resolution ADCs, which truncate bits of information, to save energy.

The challenge of this approach is that the truncated bits depend on the specific *quantization parameters* of the DNN, specifically the quantization zero and scale as discussed in Section 2.4. These quantization parameters can vary between DNNs, DNN layers, and between channels in individual DNN layers [5]. Each column in a PIM crossbar may contain a different DNN filter, so to support these quantized DNNs, each column in a PIM crossbar may need to truncate a different set of bits.

Unfortunately, it is difficult for hardware to support varying bit truncation. If we are using a Sum-Fidelity-Limited ADC that truncates bits, we may have to truncate the same set of bits for all filters in the DNN. This forces the DNN to use the remaining bits only for quantization, which can lead to DNN accuracy loss. Sum-Fidelity-Limited architectures recover this accuracy loss through requantization and retraining. This involves modifying DNN weights and quantization parameters such that the values that DNN computations produce are well-represented by the range of bits kept by hardware.

Despite the challenges, an increasing number of PIM accelerators are turning to Sum-Fidelity-Limited approaches to increase efficiency [37, 10, 12, 78, 73]. This introduces new tradeoffs that are unique to PIM accelerators. Truncation and arithmetic strategies interact with the characteristics of the accelerator and the accuracy of the DNN. By incorporating truncation into hardware, Sum-Fidelity-Limited approaches

trade off flexibility and DNN accuracy to gain efficiency.

One aspect that sets RAELLA apart from these Sum-Fidelity-Limited approaches is that RAELLA's ADC-resolution-reduction strategies do not depend on truncation. Furthermore, RAELLA is also set apart in that it is *adaptive*; it can modify the distribution of generated analog values to reduce fidelity loss to a near-zero level. This gives RAELLA greater flexibility; RAELLA can run DNNs with different operand resolutions and different quantization parameters. Sum-Fidelity-Limited approaches, on the other hand, restrict DNNs to certain operand resolutions and quantization parameters.

2.10 Weight-Count-Limited Approaches

Another approach to reduce ADC energy is by using Weight-Count-Limited approaches. These approaches generally target $Converts/MAC$ through pruning, or elimination of DNN weights. By eliminating DNN weights, PIM designs can reduce the number of MAC operations they compute, and thus reduce the number of ADC converts needed to process computed results.

The pruning strategies of Weight-Count-Limited architectures are assisted by the bit-sliced arithmetic in PIM designs, which opens bits as an extra dimension in which to prune DNN weights. Unlike standard pruning strategies, which must remove eliminate entire DNN weights, bit-sliced pruning can benefit from eliminating certain bits from weights. This can simplify pruning strategies.

Unfortunately, pruning is still often difficult in PIM accelerators due to the structure of ReRAM crossbars. ReRAM crossbars have a fixed row/column structure, and pruning must take this structure into account to reduce the number of ADC converts needed to process a DNN. For example, to avoid requiring an ADC convert

to process a column sum, pruning must eliminate all weights that contribute to the column sum.

To account for the rigid row/column structure of PIM crossbars, many Weight-Count-Limited architectures use structural pruning. Architectures such as FORMS [107], SRE [102] and PIM-Prune [13] prune away groups of DNN weights. When selecting weights to prune, they pick groups of weights that share a crossbar row or column. They then activate only small sets of crossbar rows/columns of crossbars. If all weights in a set of rows or columns have been pruned, the associated computations can be avoided. These works generally activate sets of rows/columns, not all rows/-columns in a crossbar, as it is difficult to prune very large groups of weights without incurring a high DNN accuracy loss. For example, with a 100-row crossbar, 100 weights must be pruned in a column to avoid an ADC convert. If only ten rows are activated in parallel, then only ten weights must be pruned to avoid an ADC convert.

A limitation of these strategies is that, in only activating sets of crossbar rows/-columns, they do not fully exploit the parallelism offered by PIM crossbars. For example, FORMS [107] only activates eight crossbar rows in parallel, compared to other PIM designs which may activate hundreds of rows in parallel [37].

Alternatively, an interesting pruning strategy is presented by TinyADC [106], which uses pruning to reduce $Energy/Convert$ instead of $Converts/MAC$. They exploit bit-sliced arithmetic and prune individual weight slices. They use this pruning to limit the number of nonzero weight slices in each crossbar column, thus restricting the number of nonzero sliced products and limiting the maximum value of column sums. This permits the use of a lower-resolution ADC without fidelity loss.

2.11 How ADC Energy Scales

In this section, we discuss how column sum magnitude and ADC energy scale with various architectural attributes. While we discuss scaling trends to gain insight into the design space, there are many more factors that influence the energy of the ADC. We discuss some of the factors here through the lens of architectural attributes, although for a full design space exploration, we recommend using tools [96, 97, 58] to test each design point.

First, we would like our ADC to maintain high fidelity. This means that it should be able to well-represent the column sums that it reads from the analog crossbar. Put another way, if the column sums are N-bits, then our ADC should be N-bit as well. Given this restriction, we can calculate the ADC resolution as $ceil(log_2(Maximum\,Column\,Sum - Minimum\,Column\,Sum))$. When considering scaling factors, we simplify this to $log_2(Maximum\,Column\,Sum)$.

Given that we would like our ADC to maintain high fidelity, Table 2.5 shows how the magnitude of column sums and the energy of the ADC scale with various architectural attributes. As the architectural attributes change, high fidelity is maintained by varying the ADC resolution (and thus, energy). We can break down the scaling factors as follows:

- Column Sum Magnitude scales with $2^{Bits/Slice}$ as the value in a slice scales with $2^{Bits/Slice}$. If sliced products have a nonzero mean, column sum magnitude scales linearly with the number of crossbar rows due to the accumulation of additional values. If sliced products have a zero mean, however, the average magnitude of column sums is always zero. In this case, we look at the distribution of column sums. The mean column sum is zero, but the standard deviation in column sums scales with the square root of the number of crossbar

rows (one row contributes one accumulated sliced product).

The square root scaling is due to the Central Limit Theorem, which states that, if we accumulate many independent random values, the distribution of the sum will approach a Gaussian distribution with a standard deviation proportional to the square root of the number of accumulated values [20].

- ADC $Energy/Convert$ scales with approximately $2^{ADC\ Resolution}$ [89]. Recall that, to maintain fidelity, ADC resolution must scale with $log_2(Maximum\ Column\ Sum)$. After following this logarithm and exponent, we find that $Energy/Convert$ increases proportionally to column sum magnitude.

- $Converts/MAC$ scales inversely with the bits per input/weight slice and with the number of crossbar rows. Intuitively, more bits per slice means fewer slices, and fewer ADC converts required to process all slices. Similarly, more crossbar rows mean more sliced products accumulated per ADC convert, and therefore fewer ADC $Converts/MAC$.

- ADC Energy scales with the product of the $Energy/Convert$ and $Converts/MAC$ scaling factors, by the Titanium Law.

We would like to minimize the ADC energy needed to capture column sums. These factors show how we can make design tradeoffs to do so.

First, we can examine the interaction between ADC energy and bits per slice. Notice that it is energy-intensive to process many bits per input or weight slice due to the scaling factor $Energy/Convert \sim 2^{Bits/Slice}/(Bits/Slice)$. However, when bits per slice is low (1-4), we can trade off bits per slice efficiently. This is because, for small numbers of bits per slice, ADC energy scales slowly with bits per input/weight slice as the $2^{Bits/Slice}$ and $1/(Bits/Slice)$ portions of the ADC energy scaling factor

Architectural Attribute	Column Sum Magnitude	$Energy/Convert$	$Converts/MAC$	ADC Energy
Bits per input slice	2^x	2^x	$1/x$	$2^x/x$
Bits per weight slice	2^x	2^x	$1/x$	$2^x/x$
Crossbar rows, nonzero-mean sliced products	x	x	$1/x$	1
Crossbar rows, zero-mean sliced products	$\sqrt{x}$	$\sqrt{x}$	$1/x$	$1/\sqrt{x}$

Table 2.5: How column sum magnitude, $Energy/Convert$, and $Converts/MAC$, and ADC energy scale with various architectural attributes. x represents the architectural attribute. We assume each bit of ADC resolution doubles ADC energy [89]. ADC resolution (and thus, energy) is varied to preserve fidelity. Constant factors are omitted. While we discuss scaling trends to gain insight into the design space, there are many more factors that influence the energy of the ADC. We discuss some of the factors here through the lens of architectural attributes, although for a full design space exploration, we recommend using tools [96, 97, 58] to test each design point.

oppose one another. In these regions with 1-4 bits per slice, we can trade off bits per slice for number of slices. We will exploit this property with Adaptive Weight Slicing in Section 4.2 and Dynamic Input Slicing in Section 4.3. These strategies trade off bits per input/weight slice and number of input/weight slices while considering the values stored in each slice.

Next, we can examine how ADC energy interacts with the number of crossbar rows. Notice that if the sliced products have a nonzero mean, increasing the number of crossbar rows is not helpful; increasing $Energy/Convert$ is negated by decreasing $Converts/MAC$. However, with zero-mean sliced products, we get the interesting result that as we increase the number of crossbar rows, $Energy/Convert$ increases with $\sqrt{x}$ while $Converts/MAC$ decreases with $1/x$, and overall ADC energy decreases with $1/\sqrt{x}$. This suggests that we would like to use as many crossbar rows as possible to reduce ADC energy. In reality, the size of the crossbar is limited by

$1/Utilization$. Most DNNs filters do not compute more than a few hundred MACs in parallel, and if there are more crossbar rows than MACs in a filter, the extra rows will not be needed. The number of crossbar rows is also limited by analog nonidealities and analog noise. Still, the $1/\sqrt{x}$ scaling factor has a beneficial effect even with reasonably sized 512-row crossbars. We exploit this scaling factor with Center+Offset in Section 4.1, which lowers ADC resolution by attempting to generate zero-mean sliced products.

2.12 Motivation

Sum-Fidelity-Limited and Weight-Count-Limited approaches both modify DNNs. This DNN modification causes accuracy loss, and prior works combat accuracy loss by retraining DNNs. FORMS [107], a Weight-Count-Limited architecture, achieves a 2× *MACs/DNN* reduction on ResNet18 by pruning and retraining. TIMELY [37], a Sum-Fidelity-Limited architecture, achieves up to a 512× *Converts/MAC* reduction over [72] by using large crossbars and many bits per input/weight slice. However, TIMELY also loses 16b of fidelity from each column sum and recovers accuracy with DNN requantization and retraining. Table 2.4 shows a gap in recent PIM works: some PIM architectures are inefficient and do not reduce high ADC area and energy, while others that reduce ADC area and energy cause DNN accuracy loss and retrain to compensate.

Retraining DNNs can be a challenge due to high computational cost [60], cumbersome hyperparameter tuning [28], and the potential lack of access to training datasets [84, 68]. Additionally, highly efficient DNNs such as highly-reduced-precision models [14, 18, 11] often depend on their own training/quantization procedures. If an architecture requires different training/quantization procedures, it may be difficult

or impossible to run these cutting-edge DNNs. The motivation behind RAELLA is to deliver efficient inference and avoid accuracy loss without retraining or modifying DNNs.

2.13 Prior Works without Retraining

Other works, like RAELLA, explore reducing ADC area and energy in PIM architectures without relying on retraining DNNs.

Xiao et al. [99] provide an in-depth and insightful exploration of DNN accuracy versus fidelity, differential encoding, and PIM design space decisions. We urge the reader to read this work for a deeper understanding of the tradeoffs explored in RAELLA.

Newton [53] proposes flexible strategies to improve PIM accelerator efficiency. Newton uses heterogeneous crossbars (different on-chip crossbars have different dimensions and ADC resolutions) and arithmetic transformations. Newton also varies ADC resolution. These improvements are orthogonal to those of RAELLA; it would be interesting to see how an accelerator may combine both.

Several other works reduce ADC resolution while attempting to maintain high fidelity. Guo et al. [21] exploit naturally low column sum to reduce ADC resolution and scale the number of crossbar rows used based on a column sum prediction. McDanel et al. [49] explore low-resolution ADC quantization and DNN error tolerance. RAELLA achieves much greater ADC resolution reductions than these works (2b-3b vs. 10b).

2.14 Summary

In this chapter, we discuss important concepts in Processing-In-Memory (PIM) for Deep Neural Network (DNN) inference acceleration. We review the design space around PIM and discuss the various methods that co-locate memory and compute. Focusing specifically on one method, PIM, we discuss the memory technologies that can be used to realize PIM and described how PIM computation is carried out.

We also focus on the analog-to-digital converters (ADCs) that limit PIM accelerators, showing how to calculate ADC energy with the Titanium Law and using the Titanium Law as a lens to look at how Sum-Fidelity-Limited and Weight-Count-Limited architectures reduce ADC energy. After doing a deep dive into these architectures, we finally discuss how ADC energy scales with various architectural attributes, discuss related works, and give the motivation behind the RAELLA accelerator.

Chapter 3

RAELLA: Low Resolution, High Fidelity

This chapter introduces the RAELLA accelerator at a high level. We first introduce the problems that RAELLA attempts to solve and show RAELLA's approach to maintaining high fidelity even as it reduces the energy and area cost of ADCs. We then outline each of RAELLA's three strategies, discussing the problem targeted by the strategy, how the strategy helps, what tradeoffs the strategy makes, and the effects of the strategy on the system.

3.1 RAELLA's Approach

To be efficient, we would like to reduce ADC resolution. But if column sum resolution is greater than ADC resolution, we lose fidelity and DNN accuracy. We identify three architectural tradeoffs that create high-resolution column sums:

- More sliced products per ADC convert → fewer ADC converts, higher-resolution

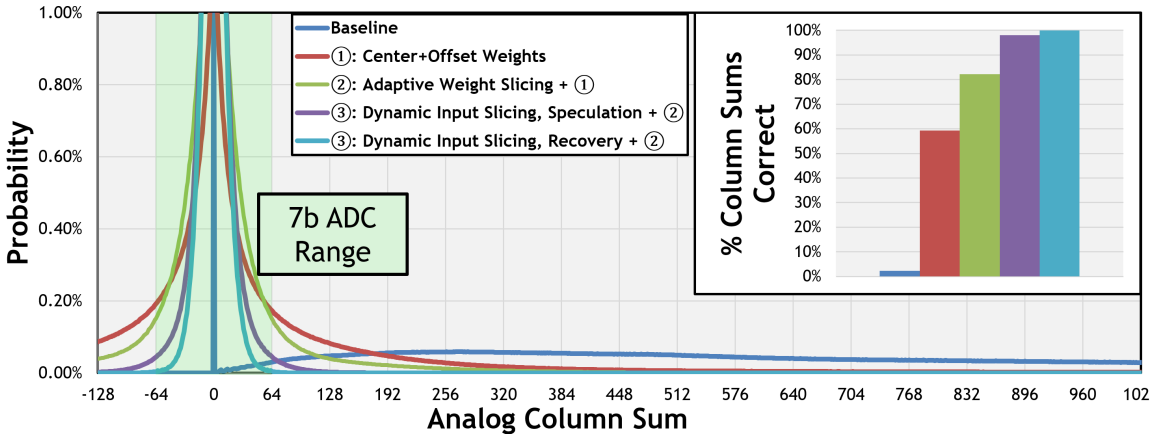

Figure 3-1: Column sum distribution with each of RAELLA's strategies while running ResNet18 on ImageNet. RAELLA reduces column sum resolution from 17b to 7b and reduces ADC saturation rate from 98% to 0.1%.

column sums.

- More bits per weight slice → fewer weight columns, fewer ADC converts, higher-resolution column sums.

- More bits per input slice → fewer input cycles, fewer ADC converts, higher-resolution column sums.

Here, we give an overview of RAELLA's strategies targeting these three tradeoffs. We start with a baseline that uses a 512×512 crossbar and 4b input/weight slices. Shown in Fig. 3-1, this setup will produce a very wide distribution of column sums that range from zero to tens of thousands. It requires 17b to represent these column sums. RAELLA's strategies tighten the column sum distribution until it can be represented with a signed 7b range of $[-64, 64)$.

To capture this 7b range without losing fidelity, we set RAELLA's ADC to always capture the seven least-significant bits (LSBs) of column sums. That is, if a single crossbar row is on and produces a sliced product of one, the ADC will read the

column sum and output the value one. This small step size preserves full fidelity for in-range column sums.[1] The drawback is that a small step size means a small range; the ADC saturates and loses fidelity if the column sum is outside $[-64, 64)$. With 4b input/weight slices, even a single crossbar row can produce sliced products up to $(2^4 - 1)^2 = 225$, which would be saturated at 63. RAELLA must avoid saturation while summing 512 rows at once.

By reshaping the column sum distribution, RAELLA's strategies reduce the probability of saturation to near-zero. This is how RAELLA achieves high fidelity with a low-resolution ADC: RAELLA's ADC only loses fidelity if column sums are large, and the following strategies make column sums small. For each strategy, we report the ADC saturation rate running ResNet18 on ImageNet.

3.2 Center+Offset Weights

Problem

Standard ReRAM crossbars compute unsigned sliced products. If each sliced product is ≥ 0, then accumulating many sliced products will generate large-valued, high-resolution column sums.

Solution

We shift weights by a center value such that approximately half of the weights are above the center and half are below. As a result, when we slice weights and compute with them in a crossbar, approximately 50/50% of the sliced products come from

[1] This approach contrasts with the approach of many Sum-Fidelity-Limited architectures, which drop LSBs from computations [37, 10, 12] While dropping LSBs permits a lower saturation chance, it also necessarily loses fidelity in every psum.

positive/negative weights. We then sum the signed sliced products in-crossbar. Positive and negative sliced products negate, yielding small-valued column sums even as many sliced products are accumulated. To maximize the beneficial negation that occurs, Center+Offset chooses centers that balance the magnitude of positive/negative slices in each crossbar column.

Tradeoff

RAELLA trades off higher crossbar area to implement signed arithmetic in-crossbar. Crossbars are dense, so the area tradeoff is worthwhile to reduce ADC area and energy. RAELLA also uses additional storage and low-cost digital circuitry to store and process center values.

Result

With Center+Offset weights, the column sum distribution labeled ① in Fig. 3-1 is signed and centered around zero. Column sum resolution is $\leq$ 7b 59.2% of the time.

3.3 Adaptive Weight Slicing

Problem

More bits per weight slice increase the values stored in weight slices, raising column sum values and resolutions.

Solution

Shown in Fig. 4-4, RAELLA adaptively slices weights at compilation time. We can reduce the average values stored in weight slices and reduce column sum resolution

by using fewer bits in each weight slice. However, additional weight slices increase the storage footprint and number of ADC converts by increasing the number of columns, so we would like to minimize the number of slices used. During compilation, we measure errors caused by fidelity loss. We choose the number of bits in each weight slice to control errors and minimize the number of slices used. RAELLA can use a different number of bits for each slice, but all weights in a layer use the same slicing.

Tradeoff

RAELLA trades off storage density, ADC converts, and compilation-time preprocessing. The number of ReRAMs and ADC converts needed increases with the number of weight slices. RAELLA uses a simple preprocessing strategy and reuses DNN weights for many inferences to minimize preprocessing costs.

Result

With Adaptive Weight Slicing, the column sums labeled ② in Fig. 3-1 are more tightly distributed. Column sum resolution is $\leq$ 7b 82.1% of the time.

3.4 Dynamic Input Slicing

Problem

More bits per input slice increase the values of input slices, raising column sum values and resolutions.

Solution

Shown in Fig. 4-6, RAELLA dynamically slices the inputs at runtime. RAELLA can use fewer bits per input slice to reduce column sums. However, this requires more cycles and more ADC converts. RAELLA uses a dynamic approach by speculating with an efficient strategy of more bits per input slice. RAELLA recovers from large-column-sum saturation errors by using fewer bits per input slice. To save energy when using the less-efficient fewer bits per input slice, RAELLA only activates the ADC if there was an error while using more bits per slice (*i.e.*, most columns speculate successfully, and during recovery the ADC only process results from the unsuccessful columns). This strategy achieves high efficiency from speculation and high fidelity from recovery.

Tradeoff

RAELLA trades off throughput and crossbar energy. While typically speculation is used to increase speed, RAELLA's speculation trades off speed to gain efficiency. Extra cycles are needed to run both speculation and recovery. Additionally, RAELLA's crossbars consume energy for both speculation and recovery. As crossbars are high-throughput and efficient, it is worth the cost to reduce the ADC overhead.

Result

With Dynamic Input Slicing, speculation and recovery column sum distributions labeled ③ in Fig. 3-1 are further tightened. In speculation and recovery cycles,

column sum resolution is $\leq$ 7b 98.0% and 99.9% [2] of the time, respectively.

3.5 Accepting Fidelity Loss

Problem

With all of RAELLA's optimizations, the column sum resolution can still be greater than ADC resolution. We use a 7b ADC and produce >7b column sums 0.1% of the time. These cause the ADC output to saturate at its min/max of -64/63 and propagate incorrect values to the psum.

Solution

DNNs are inherently noise-tolerant [32, 64] so a low error rate is acceptable. Table 6.1 shows that RAELLA's fidelity errors cause low loss for a variety of DNNs.

RAELLA uses a 512-row crossbar and a 7b ADC. Even with minimal 1b input and 1b weight slices, column sum resolution may be 9b, so it is impossible to guarantee perfect fidelity. With minimal weight slice sizing, RAELLA reduces ADC-related fidelity errors to a rate on the order of one error in ten million psums, or one incorrect psum per ResNet50 [22] inference.

3.6 Summary

In this chapter, we introduce the three strategies of RAELLA and discuss how they make tradeoffs to reduce large-valued column sums. Center+Offset encoding at-

[2] 99.9% measures the aggregate number of column sums, not just the ones that were active during recovery. Put another way, if we were to run speculation cycles without recovery cycles, then 98% of column sums would be $\leq$ 7b. If we were to run recovery cycles without speculation cycles, 99.9% of column sums would be $\leq$ 7b.

tempts to use signed, zero-mean weight slices in each crossbar column such that generated column sums are small. Adaptive Weight Slicing chooses the densest, most efficient slices for each DNN layer while preserving efficiency. Finally, Dynamic Input Slicing speculates with aggressive-efficient input slices and recovers incorrect results with conservative input slices.

Through these strategies, RAELLA is able to reshape the distribution of column sums, decreasing column sum resolution from 17b to 7b and decreasing 7b ADC saturation rate from 98% to 0.01%. This permits RAELLA to use an efficient low-resolution ADC with low fidelity loss and low DNN accuracy loss.

Chapter 4

Implementing RAELLA's Strategies

This chapter provides the implementation details of each of RAELLA's strategy. We first discuss Center+Offset, showing the importance of Center+Offset, how Center+Offset arithmetic works, how to calculate optimal center values, and finally how Center+Offset is implemented in hardware. Next, we discuss Adaptive Weight Slicing, showing how changing the number of weight slices can trade off efficiency, density, and fidelity. We then show the algorithm used to select the slices for each DNN layer. Finally, we discuss Dynamic Input Slicing, showing how it is implemented in hardware, and showing the system effects of speculation and recovery.

4.1 Implementing Center+Offset Weights

Shown in Fig. 4-1, we represent DNN weights as a center value plus or minus a small offset. We select centers to make positive and negative weight slices approx-

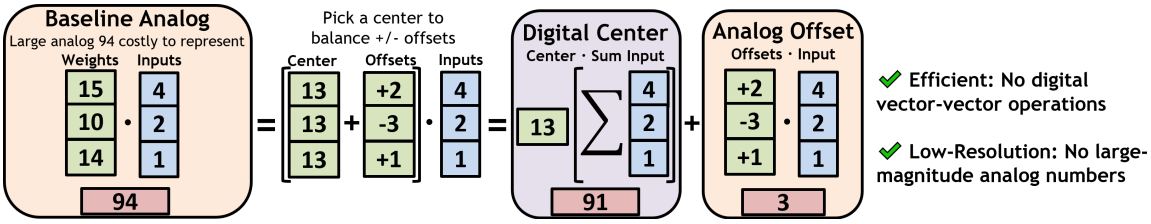

Figure 4-1: Center+Offset weights. Standard dot products create high-resolution values which are difficult to represent in analog. Center+Offset digitally subtracts a center from weights, computing with near-zero-average offsets in-crossbar.

imately the same average magnitude for each column. RAELLA computes signed analog arithmetic, and sliced products from the magnitude-balanced positive/negative weight slices negate to produce near-zero column sums. This allows RAELLA to keep small column sums while accumulating sliced products from many crossbar rows. Meanwhile, RAELLA efficiently processes high-resolution centers in the digital domain.

We first discuss why Center+Offset is important for balancing positive/negative weight slices, then show how RAELLA computes arithmetic with Center+Offset encoding and describe how we calculate optimal center values. Finally, we show the hardware for computing dot products with Center+Offset encoded weights.

4.1.1 Why Balance Slices

While DNN weight distributions are commonly cited as zero-mean [99, 20], a zero mean for all weight values over a DNN does not necessarily mean any given weight filter is zero-mean, nor that column sums are zero-mean. For that, we need each individual crossbar column to have weight slice values with a zero mean. This is often not the case, as the values in individual weight filters and columns of slices

randomly converge to different distributions.[1]

A growing body of works are exploring differential encoding, which, like Center+Offset, computes signed analog arithmetic [92, 9, 43, 108, 20, 48, 99]. Differential encoding uses positive-valued slices to represent positive weights and negative-valued slices to represent negative weights; it can benefit from Center+Offset to balance positive/negative weight slice values and reduce column sum resolution.

Center+Offset can be especially beneficial for filters where weight slice value distributions have noticeable nonzero averages. This can occur in filters where there is a greater number of negative than positive weights, such as the filter shown in Fig. 4-2. Differential encodings represent these mostly negative weights with mostly negative slice values, yielding a negative average for the slices in each column. After dot products with hundreds of slices, even slight negative average values can accumulate to create large negative column sums. This effect can significantly increase ADC saturation and cause DNN accuracy loss, as shown in Table 6.1. By balancing positive/negative slice values, Center+Offset reduces per-column biases and protects from accuracy loss.

4.1.2 Center+Offset Arithmetic

Given a weight w and center ϕ, we calculate positive offset $w_+ = max(w - \phi, 0)$ and negative offset $w_- = max(\phi - w, 0)$. For weights above the center, w_+ is the difference between the weight and the center while w_- is zero. The converse is true

[1]When we say "filter" we mean a set of weights from one dot product that fit in one crossbar. An output channel of a DNN layer (or "filter" in the traditional sense) may be partitioned over multiple crossbars if its weights do not fit in a crossbar. The important aspect is that each crossbar column produces a unique column sum distribution, regardless of the characteristics of the overall DNN. To account for this, Center+Offset attempts to balance positive/negative weight slice values in each column.

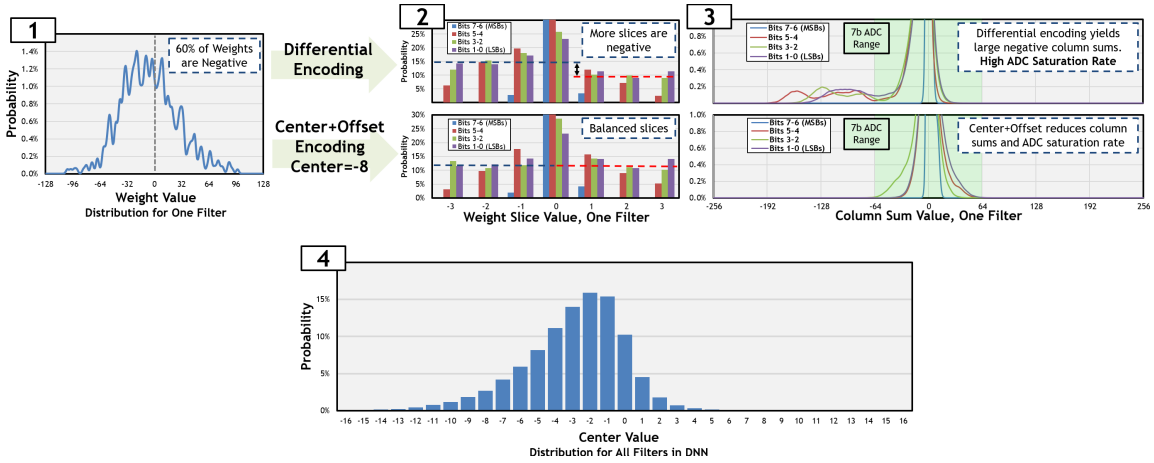

Figure 4-2: Differential vs. Center+Offset Encoding. Distributions for an InceptionV3 [83] filter with negative-average weight slice values is shown for illustrative purposes. 8b weights / inputs are sliced into four 2b / eight 1b slices. 1 Most of the weights in a filter are negative. 2 Differential encoding represents negative weights with negative-valued slices, yielding mostly negative slice values for the filter. Center+Offset balances positive/negative slice values. 3 Dot products with mostly negative-valued slices yield large negative column sums that cause ADC saturation. Center+Offset reduces column sums. 4 Each DNN filter needs a different center.

for weights below the center. Given weight filter W programmed as positive/negative offset vectors W_+, W_-, RAELLA computes a dot product with input vector I as:

$$W \cdot I = (\phi + W_+ - W_-) \cdot I = \left(\phi \sum I\right) + (W_+ - W_-)I \tag{4.1}$$

RAELLA computes Eq. 4.1 at runtime, with $\phi \sum I$ computed digitally and $(W_+ - W_-)I$ in analog.

4.1.3 Calculating Optimal Centers

We calculate centers/offsets with one-time preprocessing before programming RAELLA. We calculate a center for each weight filter independently, as weight distributions and optimal centers vary for different weight filters.

We define an optimization problem to solve for the center value ϕ. First, we define a slice S as a sequence of inclusive bit indices $[h \dots l]$ from the most to least significant index h to l (*e.g.*, slice $[7 \dots 4]$ contains the four most significant bits). Then, we define a slicing function $D(h, l, x)$ that crops signed number x to contain the bits from indices h to l (shifted so bit l is the least significant position), preserving the sign. Given a weight filter W and slices $S_{i \in \{1,2,\dots,N\}} = [h_i \dots l_i]$, we solve for the center ϕ of W as follows:

$$\underset{\phi \in \{1,2,\dots,255\}}{\arg\min} \sum_{i=1}^{N} 2^{l_i} \left(\sum_{w \in W} D(h_i, l_i, w - \phi) \right)^4, \tag{4.2}$$

where N is the total number of slices, and w is a weight in W. Eq. (4.2) balances positive/negative values in each column of weight slices, assigning higher costs for columns with larger nonzero sums. The inner sum $\left(\sum_{w \in W} D(h_i, l_i, w - \phi)\right)^4$ calculates the cost for a single column, equal to the sum of weight slice values in the

column raised to the power of four. Four is chosen empirically; we find that too low a power does not sufficiently penalize large sums, while too high a power overvalues the largest-sum column and fails to consider all columns. The outer sum $\sum_{i=1}^{N} 2^{l_i}(...)$ weights cost by bit position (*e.g.*, the most significant bit in an 8b number has a magnitude of 2^7 and the cost of a 1b slice containing only this bit would also be scaled by 2^7) and sums costs for all columns. Costs are weighted by bit position as saturations in higher-order slices have a greater impact on the psum.

We calculate centers for each weight filter (*i.e.*, a single dot product) in the crossbar independently. A coarser granularity, such as a single center for a full crossbar (*i.e.*, 100+ different filters), would not be as effective, as different DNN filters can have different weight distributions and require different centers.

RAELLA's per-filter centers have the drawback that each center balances multiple columns for one filter, and therefore may not be optimal for any one column. Ideally, per-column centers would be able to precisely zero the average weight slice value for each column. However, this strategy is limited by integer precision centers. Consider the case where a column of slices has an average value of 0.4. We could shift the values of all weight slices in the column by -1, but this would worsen the average by shifting it to -0.6. Instead, we shift full-precision weight values before slicing, which can reshape (rather than shift) the value distribution for each individual slice. This distribution reshaping can make smaller adjustments to the average weight slice value in each column.

We found that RAELLA's per-filter centers are able to reduce the nonzero averages in each crossbar column to near-zero. There are still improvements to be made, however. Slight averages– on the order of -0.1 to 0.1 or less– may still remain after Center+Offset encoding. Looking at the full-DNN column sum distributions in Fig. 3-1, there are slight asymmetries even after Center+Offset encoding. These can

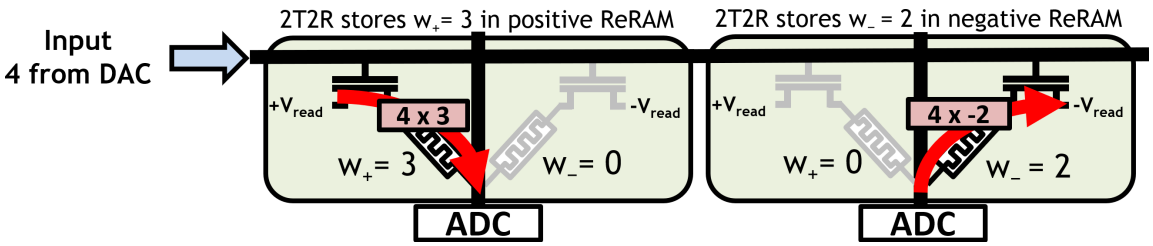

Figure 4-3: 2T2R devices compute signed arithmetic in-crossbar. Red shows the magnitude and direction of the current flow. Grayed-out devices are off (set to the high-resistance state).

be observed by noticing where the red curve crosses the 0.20% probability mark.

To remedy these slight asymmetries that still remain, per-column or input-aware strategies may be necessary. Per-column strategies will require a solution to the integer quantization problem discussed, in order to correct for the very small remaining biases.

4.1.4 Center+Offset In Hardware

Computing psums with Eq. (4.1) requires two terms. The first term, the input sum $\phi \sum I$, is calculated digitally. Crossbar columns share input vectors, so the sum calculation is amortized across columns. Further, we use a running-sum input strategy that exploits input reuse to consume even less power.

We note that an input sum and multiplication are required for linear quantization strategies that have zero points [33], and, furthermore, the Center+Offset strategy is equivalent to choosing a different quantization zero for each weight filter in the crossbar. This computation can be combined with quantization to have zero additional cost.

The second term is the vector-vector multiplication with offsets. $(W_+ - W_-)I$ is calculated in analog by the crossbar. RAELLA uses 2T2R devices, shown in Fig. 4-

3, to realize analog subtraction in-crossbar.[2] 2T2R, with two ReRAMs (2R) per weight accessed via two access transistors (2T), have been explored as a method to represent signed weights [92, 9, 43, 42, 100]. One ReRAM device is connected to a positive source and the other a negative source, letting 2T2Rs add to or subtract from column sums. For each weight, we program positive/negative offsets w_+/w_- into the two ReRAMs. As one offset is zero for any given weight, one ReRAM device is used in each pair. Added ReRAMs and access transistors increase RAELLA's crossbar size, but crossbars are small, and the increase in system area is only $\sim 10\%$.

4.2 Implementing Adaptive Weight Slicing

Adaptive Weight Slicing minimizes the number of weight slices used for each DNN layer. It uses as many bits as possible in each slice without excess fidelity loss. Fig. 4-4 shows various slicings available to RAELLA. More bits per slice means fewer slices per weight, denser storage, and fewer ADC converts, but more bits also increase the values stored in each weight slice and raise the chance of high-resolution column sums. RAELLA can use a different number of bits for each slice, but all weights in a layer use the same slicing.

The bit density, or probability that a given bit is 1, affects the values in weight slices. Fig. 4-5 shows per-bit densities for DNN inputs and weights in a typical DNN layer. Input values generally follow right-skewed distributions, yielding sparse high-order bits. Weight values generally follow rough bell curves. When represented with Center+Offset encoding, this also yields sparse high-order bits. Due to sparsity in the high-order weight bits, in most layers, 4b weight slices can store the highest-order 4b of weights with low values and low column sums. Low-order weight bits are denser

[2] Analog subtraction can also be done with circuits [31] 1T2R [108], and SRAMs [105, 57].

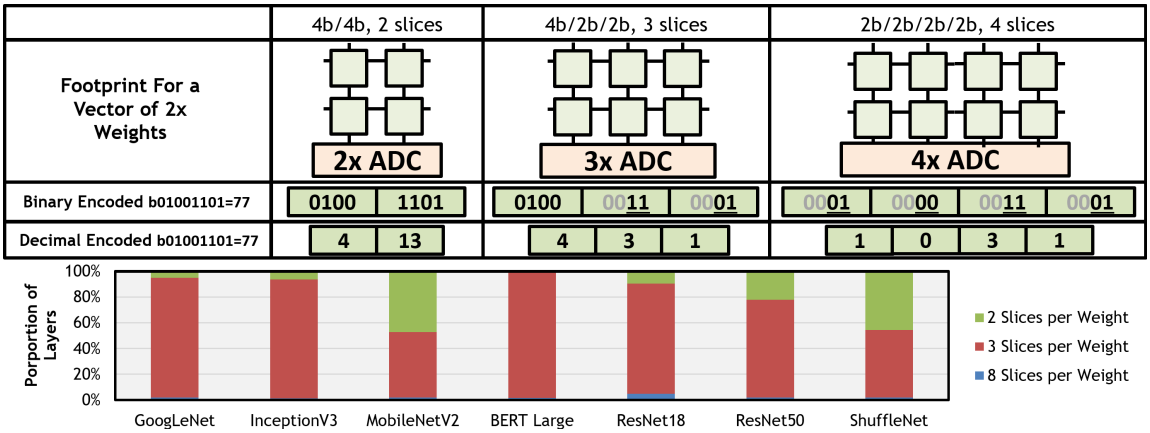

Figure 4-4: **(Top)** Weight Slice Crossbar Footprints. **(Bottom)** DNN Per-Layer Weight Slicings. Increasing slice count lowers column sums and saturation chance, but increases *Converts/MAC*. Most layers use three slices per weight.

and usually require a lower 2b per weight slice. Fig. 4-4 shows the per-layer slicings of DNNs. Most layers use the 4b-2b-2b setup with three weight slices: one weight slice for the highest-order four bits and two weight slices storing two low-order bits each.

4.2.1 Error Budgets

We could choose weight slices to minimize saturation, but this is too conservative. DNNs can tolerate some error, so we would like to allow a small amount of saturation. Unfortunately, it is difficult to predict how slicing impacts saturation and how saturation affects error in DNN layer outputs. Too-large column sums cause saturation, and column sums are affected by input/weight distributions, input/weight slice value distributions, correlations between distributions, the number of weights per filter, and random variance. Furthermore, 16b psums are digitally quantized into 8b outputs [109], which may magnify or shrink the error.

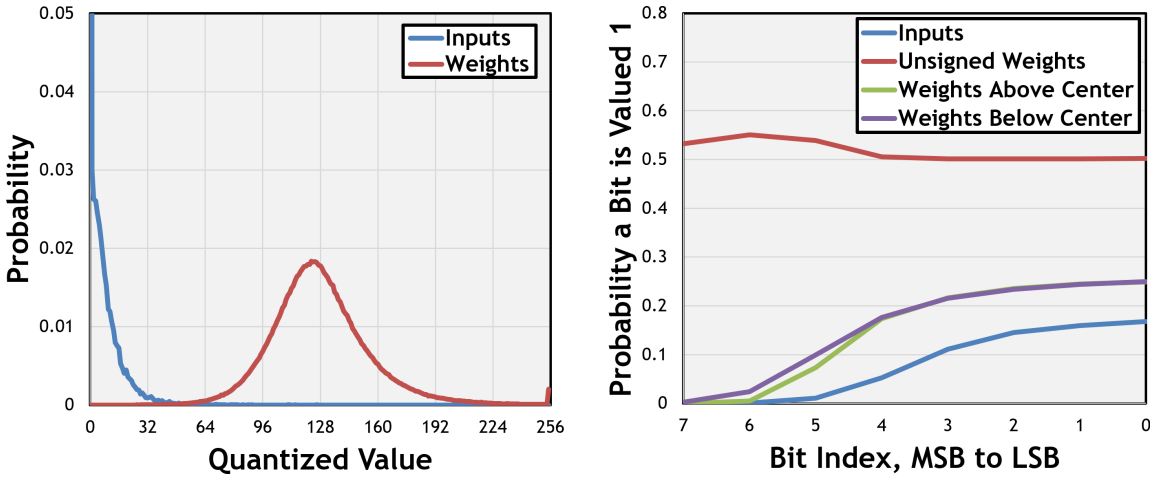

Figure 4-5: **(Left)** DNN input/weight value distributions without slicing and **(Right)** per-bit densities. The second-to-last layer of ResNet50 is shown, representing a typical DNN layer. Bell-curve-distributed weights can be split about a center into two similar distributions with sparse high-order bits. Unsigned inputs have naturally sparse high-order bits.

To capture all these factors, we take an empirical approach. We define the *error budget* as the average magnitude error allowed for nonzero outputs of a layer after outputs are fully computed and quantized to 8b. Only nonzero outputs are counted when calculating average magnitude error to give a more consistent calculation for layers with varying output sparsity.[3]

To calculate error, we use ten inputs chosen randomly from the validation set. It suffices to use so few inputs because changing slicings may change the error by an order of magnitude or more (*i.e.*, moving from two slices to three slices may change the average error from 0.2 to 0.02), and these differences are easily detected.[4] The order-of-magnitude differences stem from the shape of the column sum distribution

[3]This is important when ReLU is folded into quantization. If ReLU zeros an output, it will likely zero any error associated with that output as well. If ReLU zeros many outputs, then average error is lowered while error per nonzero output remains consistent.

[4]In fact, this algorithm usually picks the same slicings when testing just one input. If we test with Gaussian noise as input, then slicings match for $> 90\%$ of layers.

(Fig. 3-1). The distribution tails shrink exponentially, so changes in the distribution width (*i.e.*, due to slicings) have an exponential effect on the saturation rate.

The error budget is set to 0.09 in our tests, corresponding to one in eleven 8b outputs being off by one on average. After quantization, the errors created by ADC saturation are generally small and cause a low accuracy loss, shown in Table 6.1.

4.2.2 Choosing Weight Slices

Weight slices are calculated with the preprocessing procedure shown in Algorithm 1. Preprocessing occurs once when compiling a DNN for RAELLA, taking 10-1000ms per layer on an Nvidia RTX 2060 GPU. After preprocessing, sliced+encoded weights are programmed to crossbars for use with any number of inferences.

For an M-bit weight and up to N bits per ReRAM, we define a *slicing* as a tuple of integers $1 \leq s_0..s_j \leq N$ such that $\sum s_i = M$. For 8b weights, $\leq$ 4b per ReRAM, slicings include (4b,4b), (2b,1b,1b,4b), and (1b,2b,2b,3b). There are 108 slicings in total.

To find the best slicing for a DNN layer, we iterate through all 108 slicings. For each, we Center+Offset encode weights following Section 4.1, simulate the crossbar with ten test inputs, and record error. We choose the slicing that uses the fewest slices and has below-budget error. For slicings with the same number of weight slices, the lower-error slicing is chosen.

Testing all 108 slicings for each layer would be a slow process. However, we know that if a slicing has below-budget error, it will be chosen over all slicings with more slices. We can avoid testing most slicings by testing slicings from the fewest slices to the most slices, and stopping after the desired error budget has been reached for some number of slices.

We use 1b input slices when comparing weight slicings. We always use the most conservative 1b per weight slice for the last layer. The last layer has an outsized effect on DNN accuracy [11] and a less efficient last-layer slicing has little effect on overall energy/throughput as most DNNs have many more layers than the last (Fig. 4-4).

4.2.3 Adaptive Weight Slicing in Hardware

Given 4b ReRAMs, each can be programmed with $2^4 - 1$ analog levels, plus zero. To program 3b or 2b slices, we use the lowest $2^3 - 1$ or $2^2 - 1$ levels, plus zero. Given a 3b weight slice XXX, this corresponds to programming a device with 0XXX. This is only a restriction of the available range and therefore does not require a change to ReRAMs. Crossbars already need shift+add circuits to add column sums across weight and input slices; adaptive slicing requires only changing the shift+add pattern.

The main overhead depends on the number of weight slices. Each additional weight slice increases required ReRAMs and ADC converts. RAELLA can use between two weight slices (4b/slice, most efficient) and eight weight slices (1b/slice, least efficient). Most layers use three weight slices.

Algorithm 1: Preprocessing Weight Slicing and Centers

```
1  Func SliceEncodeWeights(layer, testInputs, errorBudget)
       /* DNN layer preprocessing.  Requires a layer (shape,
          quantization, weights), test inputs (activations from ten
          images/tokens in this paper), and a scalar error budget
          (0.09 in this paper).                                    */
       slicing = FindBestSlicing(layer, testInputs, errorBudget)
2      centers = FindOptimalCenters(layer, slicing)
3      return slicing, centers
4
5  Func FindBestSlicing(layer, testInputs, errorBudget)
       /* Implementation of Adaptive Weight Slicing from Sec. 4.2.
          10-1000ms per layer.                                     */
6      expectedOutputs = layer.Run(testInputs)
7      possibleSlicings = GetAllPossibleSlicings()
8      bestSlicing = possibleSlicings[0]
9      bestNSlices = CountSlices(bestSlicing)
10     bestError = ∞
11     for slicing ∈ possibleSlicings do
12         centers = FindOptimalCenters(layer, slicing)
13         outputs = layer.SimulateCrossbar(testInputs, slicing, centers)
14         errors = |expectedOutputs - outputs|
15         meanError = Mean(errors[expectedOutputs != 0])
16         nSlices = CountSlices(slicing)
17         betterSlicing = nSlices < bestNSlices || (nSlices == bestNSlices &&
            meanError < bestError)
18         if meanError < errorBudget && betterSlicing then
19             bestSlicing = slicing
20             bestNSlices = nSlices
21             bestError = meanError
22     return bestSlicing
23
24 Func FindOptimalCenters(layer, slicing)
       /* Solve Center+Offset Eq. (4.2).  <1ms per layer.  Returns a
          center for each weight filter.                           */
25     centers = SolveOptimizationProblem(layer, slicing)
26     return centers
```

4.3 Implementing Dynamic Input Slicing

Dynamic Input Slicing balances high-efficiency more-bit input slices and high-fidelity fewer-bit input slices. We would like to minimize the input slices and thus ADC converts, while also avoiding fidelity loss due to high-resolution column sums. Unlike with weights, the input slicing can be changed at runtime. This allows us to speculate with an efficient, aggressive slicing and recover with a conservative slicing. In speculation, RAELLA uses three input slices, which has high efficiency but a higher chance of creating large, high-resolution column sums. In recovery, RAELLA uses the most conservative eight 1b input slices.

The procedure for speculation and recovery is shown in Fig. 4-6. First, a 4b high-order slice is speculatively fed to the crossbar, and column sums are converted by ADCs. If a column sum is too large, it will saturate at the ADC bounds of $[-64, 64)$. If an ADC output equals either of these bounds, an error is detected and marked as a speculation failure. Next, after all columns are processed, the 4b input slice is resliced into 1b slices and processed again over four recovery cycles. To save energy in recovery, ADCs are turned off for columns that speculated successfully. In the rare event that an ADC saturates in recovery, we accept fidelity loss and propagate the saturated value. After the four recovery slices are processed, the process repeats for the following speculation and recovery cycles.

4.3.1 Dynamic Input Slicing In Hardware

Given a 4b DAC, analog input slices can take one of $2^4 - 1$ analog levels, plus zero. For an N-bit input slice, we can use the lowest $2^N - 1$ levels, plus zero. Given a 3b input slice XXX, this corresponds to converting 0XXX. As this is only a restriction of the available range, it does not require changing the DAC hardware.

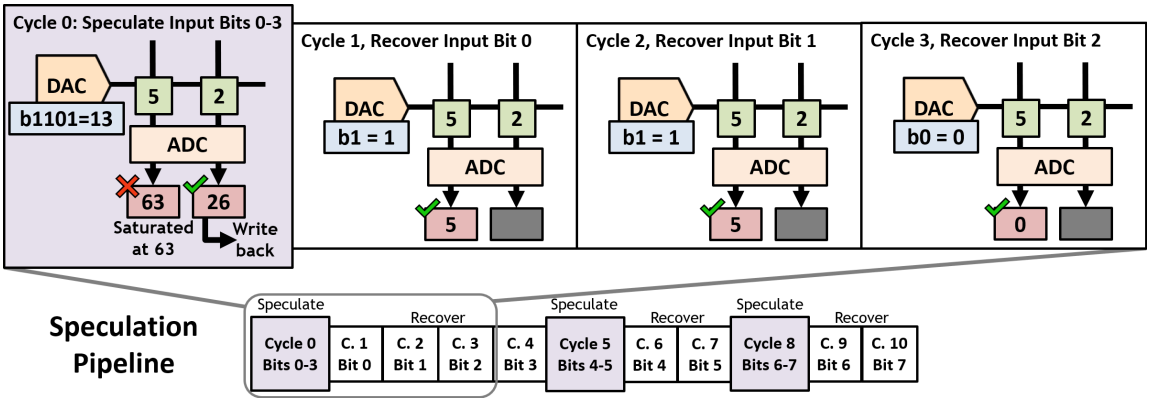

Figure 4-6: Speculative Computation. Speculative cycles use 2–4 bits per input slice, with fewer ADC converts per input but a higher saturation chance. Recovery cycles use 1-bit input slices and ADCs only process columns that failed speculation.

To track successful/failed speculations, RAELLA stores speculation success flags in a buffer for each crossbar. In recovery, ADCs only convert column sums that failed speculation.

The entire ReRAM crossbar is one unit, so all columns speculate and recover together. As it is highly likely that at least one column will fail speculation, crossbars always run recovery. Therefore, RAELLA's speculation saves energy at the cost of speed (unlike the common use of speculation for speed, *e.g.*, CPU branch prediction).

Speculation also increases crossbar energy, as all columns and ReRAMs run both speculation and recovery cycles. Recovery cycles consume less energy than speculation cycles, as ReRAM devices use less energy with smaller input values [45] and ADCs only process a small fraction of columns in recovery cycles.

4.3.2 Dynamic Input Slicing System Effects

RAELLA can run without speculation, processing eight recovery slices alone (as opposed to three speculation slices + eight recovery slices). With this approach, each

column would require eight ADC converts for all eight input slices. With speculation, three ADC converts are needed instead to process three 2-4b speculative input slices. Across our baselines, speculation fails approximately 2% of the time, requiring 2-4 recovery converts depending on which speculative slice failed. Overall, speculation succeeds $\sim$ 98% of the time and reduces ADC converts by $\sim$ 60% over a recovery-only approach. An average of three speculative converts + 0.3 recovery converts are required to process each column.

While RAELLA saves ADC converts with speculation, it trades off throughput and crossbar energy. RAELLA's crossbars require eleven cycles to run all three speculation + eight recovery slices. Alternatively, a no-speculation approach could run only the eight recovery slices, increasing throughput but also increasing the number of ADC converts required.

4.4 Summary

In this chapter, we show the precise implementation for each of RAELLA's strategies.

Center+Offset uses 2T2R devices to compute signed offset arithmetic in-crossbar. Meanwhile, digital center computation is moved into the digital domain. An optimization function is used to calculate centers during the compile-time procedure for Center+Offset encoding of DNN weights.

Adaptive Weight Slicing chooses slices for each DNN layer. Its algorithm tests each layer and evaluates the amount of error when running a set of test inputs. Given the results, it picks the best slicing possible that yields below-budget error.

Finally, Dynamic Input Slicing can run both speculation and recovery stages with minimal hardware changes. Dynamic Input Slicing speculates by more-efficient input slices with more bits per slice, then recovers by using single-bit input slices in

recovery rounds. Energy is saved in recovery stages by running the ADC only for columns that failed speculation.

All of RAELLA's strategies function with minimal hardware overhead. Center+Offset uses more ReRAM devices, which are small and have low system area impact. Adaptive Wieght Slicing and Dynamic Input Slicing require no changes to analog hardware.

Chapter 5

RAELLA Architecture and Pipeline

In this chapter, we will describe the RAELLA architecture from the bottom up. We will first describe RAELLA's crossbars and the hardware they use to implement each of RAELLA's strategies. We will then move to the In-Situ Multiply Accumulate units that combine several crossbars with buffers and communication networks. Finally, we will discuss RAELLA's largest spatial unit, the tile. The high-level RAELLA architecture is shown in Fig. 5-1. RAELLA's organization mostly follows that of ISAAC [72].

After going through each of RAELLA's components, we will discuss how DNNs are run on RAELLA, including programming and dataflow. Finally, we include a section discussing how RAELLA's strategies interact with analog nonidealities.

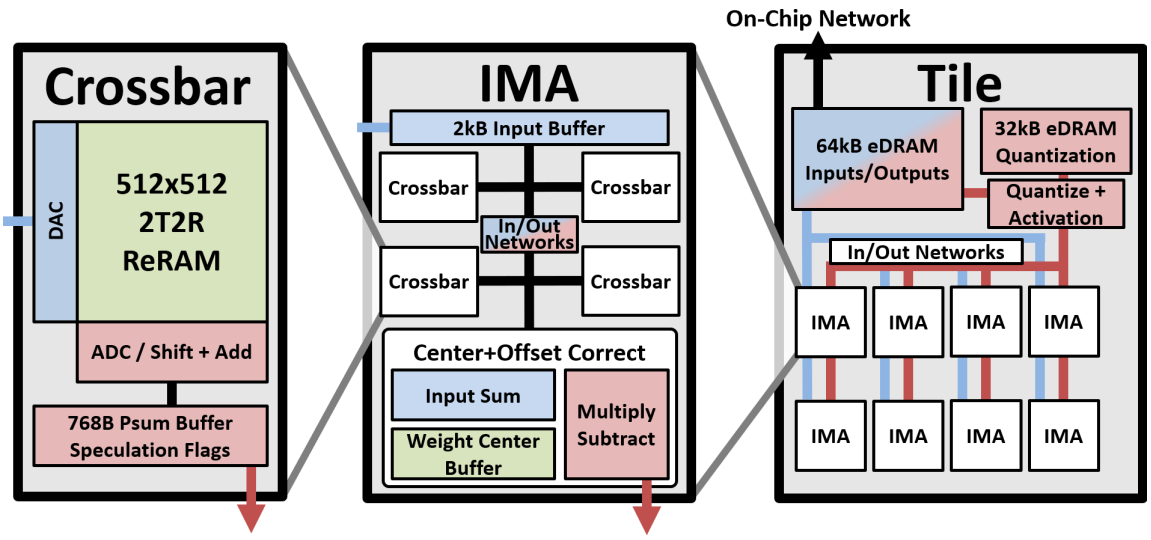

Figure 5-1: The RAELLA Architecture. **(1)** The base unit is a crossbar. **(2)** Four crossbars make up an IMA. **(3)** Eight IMAs make up a tile. Components are colored blue for input storage/processing, green for weights, and red for outputs.

5.1 Crossbar

Crossbars consist of 512×512 2T2Rs. Each crossbar is programmed with weights from one DNN layer, and each weight filter uses 2-8 crossbar columns based on the DNN layer slicing (Section 4.1).

To process inputs, we use 4b pulse-train DACs for their simple hardware [48] and superior linearity [75]. Pulse-train DACs encode the value of an N-bit input slice with a number of pulses from zero to $2^N - 1$. The DAC consists of a simple row driver to apply input pulses, a 1b flip-flop to store the current input bit, and an AND gate acting as an enable signal [48]. To output a 4b value, the most significant bit is first loaded into the flip-flop and a global clock generates eight 1ns pulses. The DAC outputs the AND'ed value of the clock and its stored value, equal to eight pulses if the bit is on and zero otherwise. Subsequent bits are loaded sequentially and run for four, two, and one pulse(s), respectively.

DACs activate the 2T2R access transistors and each 2T2R device computes a sliced product that it adds or subtracts from the column sum. Column sums appear as a current on a column, which is buffered and scaled by a current buffer [37] before being captured as capacitor voltages and held by sample+hold circuits [55].

Next, four 7b ADCs [34] convert the 512 column sums in 100ns [72]. 7b signed ADC results are summed by shift+add circuits and accumulated in 16b psum buffers [109].

With the most-dense slicing of two slices per weight, one crossbar may produce up to 256 psums, which are stored in a 256-entry psum buffer. Each entry stores a 16b psum + 8b success flags, for a 768B psum buffer total capacity.

In speculation/recovery cycles, inputs are streamed to crossbars once for each cycle. In speculation, ADCs process all columns. If an ADC saturates, the psum is not updated and the success flag is marked. In recovery, all success flags are checked. ADCs process and write results only for columns that failed speculation.

The crossbar cycle is pipelined in two stages [72]. In the first stage, the DACs supply input pulses, the crossbar computes analog column sums, and the results are latched in sample+hold circuits. 4b pulse train DACs with 1ns/1ns on/off pulse width take 30ns to send up to 15 input pulses. Crossbars settle and produce column sums in less than a nanosecond [25]. In the second stage, ADCs convert sample+hold results in 100ns [72]. The overall crossbar cycle time is 100ns from the slower-stage ADC processing.

RAELLA utilizes input bit sparsity to reduce column sum values and crossbar energy, benefiting from the high bit sparsity of unsigned inputs (Fig. 4-5). If inputs are signed, RAELLA processes positive/negative inputs in two separate cycles to generate sparsity.

5.2 In-Situ Multiply Accumulate

Four crossbars are organized into an In-Situ Multiply Accumulate (IMA) with an input buffer [72]. An input network sends input vectors to crossbars, and if all inputs are shared between two crossbars, the input vector is multicast. To exploit temporal input reuse [81, 65], the input buffer stores reused inputs between crossbar cycles. The four crossbars can process up to $4 \times 512 = 2048$ inputs across all rows, so the buffer is sized 2kB.

To support Center+Offset weights, each IMA includes a weight center buffer and digital addition circuitry to calculate input sums. A running sum is kept for each crossbar. To exploit input reuse [65], we add inputs to the sum when they are first used in crossbar columns and subtract when they are last used. If different crossbar columns use different subsets of the inputs, RAELLA adds/subtracts inputs in a streaming fashion while processing columns.

5.3 Tile

Eight IMAs are organized into a tile. Each tile includes a 64kB eDRAM buffer [72] storing 8b inputs/outputs, digital maxpool units, and quantization circuits. RAELLA digitally computes 8b per-channel quantization [109], allocating 32b per output channel to store an FP16 quantization scale and bias [109], or 32kB per tile.

5.4 Accelerator & Programming

Like ISAAC [72], every four tiles share a router enabling on-chip communication. When a tile completes a set of outputs, it sends data to the next tile via its router.

If a layer has more weights than a tile can store, its weights are split across multiple tiles.

Like other PIM accelerators [72, 37, 107, 65], RAELLA is programmed once for many inferences to mitigate high ReRAM write energy [63]. When compiling a configuration for RAELLA, we use lightweight preprocessing for Center+Offset and Adaptive Weight Slicing, as discussed in Section 4.2.2.

5.5 DNN Dataflow

Each DNN layer is mapped to one crossbar if it fits. Otherwise, it will spill over to more crossbars, IMAs, and tiles. RAELLA's interlayer dataflow follows ISAAC's [72] to minimize eDRAM footprint and inter-tile communication requirements. Fig. 5-2 shows RAELLA's dataflow. DNN layers are run in a pipeline across parallel tiles. Tiles generate one row of a layer's output tensor at a time, reusing previously used input rows and fetching only new input rows. As a tile produces rows of the output tensor from top to bottom, input rows are consumed from the previous tile in the same order. Communication and data reuse patterns are coordinated by pattern generators and fixed at program time. Below the tile level, Timeloop [58] is used to find optimal data reuse patterns.

RAELLA replicates weights to increase throughput following previous work [72, 77, 37]. If there is space, weights are replicated in-crossbar to compute multiple convolution steps using a partial Toeplitz expansion [37, 16]. Weights can be further replicated across crossbars, IMAs, or tiles. Replication follows a greedy scheme: while there are tiles left, the lowest-throughput layer is replicated.

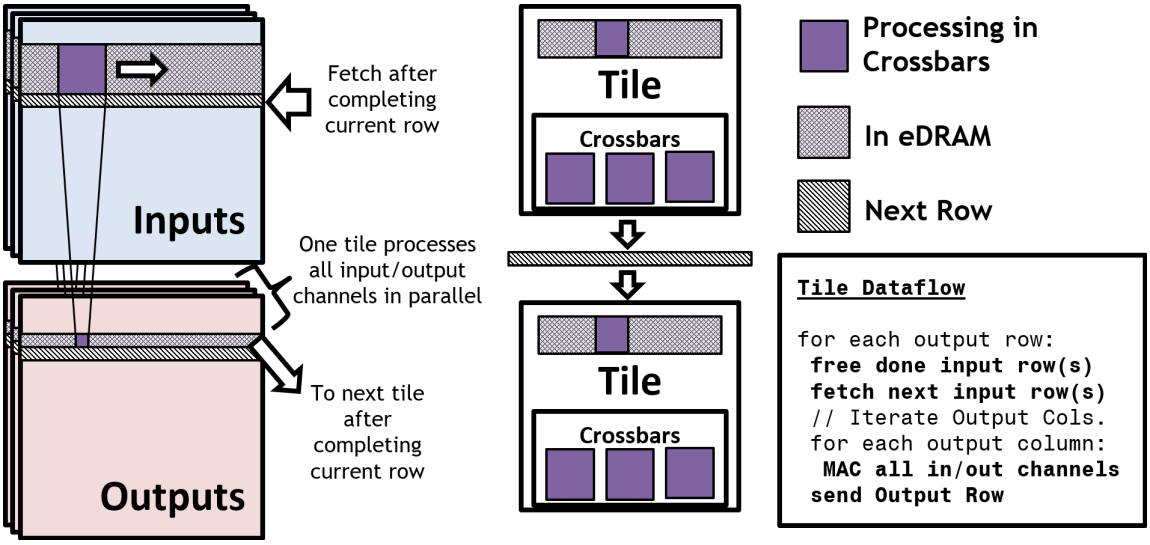

Figure 5-2: Dataflow. One row of outputs for a layer is computed at a time. Tiles receive/send inputs/outputs once.

5.6 RAELLA Reduces Analog Nonidealities

PIM crossbars can suffer from nonidealities such as IR drop and sneak current. RAELLA reduces these nonidealities relative to ISAAC.

High current traversing long crossbar columns causes IR drop, which can cause accuracy loss [12, 102]. Positive/negative 2T2R devices consume current from their neighbors, reducing IR drop [43, 108]. Furthermore, RAELLA's ADC saturates at 64, or fewer than five ReRAMs in the highest-conductance state. Therefore, RAELLA's columns must only tolerate current from five ReRAMs, compared to an ISAAC-like design that sums current for 128 ReRAMs.

Sneak current, or leakage through off ReRAMs, can cause accuracy loss [12]. Sneak current is zero in 2T2R crossbars as the leakages from positive and negative ReRAMs negate [108].

Electromigration from high current consumption causes circuit damage [85, 35].

RAELLA's 4b devices have 5× the levels as ISAAC's 2b devices (valued 0-15 vs. 0-3), so a naïve strategy could result in a 5× current increase. However, only one of two ReRAMs in a 2T2R is on, Center+Offset generates weight bit sparsity, and Adaptive Weight Slicing avoids large-valued, high-conductance weights. Overall, RAELLA's programmed conductances are 43% lower than ISAAC's on average.

5.7 Summary

In this chapter, we present the bottom-up construction of the RAELLA accelerator. RAELLA is a tiled accelerator that runs all DNN layers in parallel, passing data from one layer to the next using an on-chip pipeline. RAELLA's dataflow allows each tile to receive DNN layer inputs from the previous tile while sending outputs to the next tile, minimizing on-chip data movement and storage.

We show how RAELLA's large crossbars and IMAs are equipped with the hardware to compute Center+Offset encoding and Dynamic Input Slicing. We also show how Adaptive Weight Slicing is done in advance at compile time. Finally, we discuss analog nonidealities that challenge PIM accelerators, showing how RAELLA ameliorates these nonidealities.

Chapter 6

Evaluation

In this chapter, we will perform a comprehensive evaluation that compares RAELLA against prior PIM accelerators. We test for efficiency, throughput, and DNN accuracy in comparison to non-retraininig (non-DNN-modifying), Weight-Count-Limited, and Sum-Fidelity-Limited approaches. We use representative accelerators ISAAC [72], FORMS-8 [107], and TIMELY [37] as baselines. ISAAC does not require DNN retraining. FORMS is Weight-Count-Limited and TIMELY is Sum-Fidelity-Limited, so both retrain to recover DNN accuracy.

First, we show the efficiency and throughput gain of RAELLA in a non-retraining setting by comparing RAELLA's energy and throughput with those of ISAAC. We show that RAELLA achieves high throughput and efficiency without changing the DNN models. Next, we show competitiveness with DNN-retraining architectures by comparing RAELLA to FORMS and TIMELY. We show that RAELLA matches the efficiency/performance of these architectures without needing to retrain. Then, we show RAELLA's low accuracy loss and compare to FORMS and TIMELY. We also show the accuracy benefits of RAELLA's Center+Offset encoding.

6.1 Methodology

Models of RAELLA, ISAAC, and FORMS are created using Accelergy/Timeloop [96, 97, 58] in the 32nm technology node. The architectures are modified to support 8b DNNs as described in Section 6.1.2. Under a $600mm^2$ area budget, RAELLA fits 743 tiles while ISAAC and FORMS fit 1024 tiles each. Results for TIMELY are from the original paper [37]. To compare to TIMELY, we scale RAELLA to TIMELY's 65nm tech node and use TIMELY's analog components (TDC, IAdder, Charging+Comparator) and ReRAM devices [19] in RAELLA. RAELLA's error budget is set to 0.09 in all tests.

6.1.1 Component Models

SRAMs are modeled in CACTI [27]. Models of networks, routers, and eDRAM buffers are from ISAAC [72]. eDRAM refresh is not an issue as tiles consume data faster than a refresh period [86]. RAELLA uses the ADC [34] from ISAAC scaled to 7b following [70]. DAC, input driver, and crossbar area/energy are generated using a modified NeuroSim [7, 62]. 2T2R area is pessimistically estimated as the sum area of two ReRAMs and two min-sized transistors, ignoring potential stacking between chip layers [90, 42]. DACs use a flip-flop and an AND gate for each row to generate pulse trains [48], where each pulse is 1ns and each 4b input slice can comprise up to 15 pulses. ReRAM parameters are taken from TIMELY [37, 10], using a 0.2V read voltage and 1kΩ/20kΩ on/off resistance [19, 25]. Current buffers that capture analog column sums are taken from TIMELY [37]. Outputs are quantized with a multiply/truncate and activation functions are fused into quantization [109]. Maxpool units and sampling capacitors consume negligibly little energy/area [72, 37]. One crossbar cycle is 100ns, and crossbars produce a set of psums every 11 cycles

(three speculation input slices + eight recovery input slices) unless bottlenecked by the interlayer dataflow. Latency is doubled for signed inputs as positive/negative inputs are processed in separate cycles. With speculation disabled, crossbars require eight cycles and 800ns to produce a set of psums.

6.1.2 Models of ISAAC and FORMS

ISAAC [72] and FORMS [107] models are validated against the results presented in their papers with $< 10\%$ energy and throughput error. After validating, we model ISAAC and FORMS using the same components used in RAELLA for a fair apples-to-apples comparison. In particular, the DAC/crossbars are modeled using a modified NeuroSim [7, 62] which captures the data-dependent energy consumption of analog components. We modify both architectures and add quantization hardware to run 8b DNNs. After scaling to 8b, our ISAAC baseline has $\sim 4\times$ higher efficiency and throughput than the original ISAAC while our FORMS baseline has $\sim 2\times$ higher efficiency and throughput than the original FORMS. For FORMS, we use the highest reported pruning ratio. For a fair comparison, we modify ISAAC to support the partial-Toeplitz mappings [16, 37] that RAELLA supports, which increased the throughput of ISAAC by an additional $1-1.9\times$. These mappings were not beneficial to FORMS.

6.2 DNN Models and Test Sets

We test on seven representative DNNs. Six are CNNs from the PyTorch [59] Torchvision [47] quantized library: GoogLeNet [82], InceptionV3 [83], Resnet18 [22], ResNet50 [22], ShuffleNetV2 [46], and MobileNetV2 [71]. ShuffleNetV2 and MobileNetV2 are com-

pact with small filters, while the others are large models. We report accuracy for the ImageNet [15] validation set.

Additionally, we test a Transformer [87] BERT-Large [95] on the Stanford Question Answering Dataset [67] to show RAELLA's effectiveness on cutting-edge Transformers. For BERT-Large, we accelerate the feedforward layers. Other works explore accelerating Transformer attention [56, 80, 104]. BERT-Large shows RAELLA's performance with a non-ReLU activation and signed inputs.

6.3 Efficiency And Throughput: No Retraining

RAELLA is evaluated and compared to ISAAC running off-the-shelf models of all DNNs. Fig. 6-1 shows efficiency and throughput results. RAELLA improves energy efficiency 2.9 to 4.9× (geomean 3.9×). Efficiency gains come mainly from ADC energy reduction. RAELLA uses a 7b ADC, while ISAAC uses an 8b ADC. Furthermore, RAELLA uses larger crossbars, more bits per input slice/weight slice, and speculation to reduce ADC converts by 5 to 15×.

RAELLA's throughput benefits come from large 512×512 (versus ISAAC's 128×128) and denser 2-4 bits per weight slice (versus ISAAC's 2b per weight slice). Larger and denser weight storage and computation give RAELLA a throughput benefit of 0.7 to 3.3× (geomean 2.0×).

Without speculation, RAELLA runs recovery slices only, reducing relative efficiency benefits to 2.8× geomean due to higher ADC energy. Relative throughput increases to 2.7× geomean as crossbars do not run the three speculation slices, and psums are computed in eight crossbar cycles instead of eleven.

RAELLA is more effective with unsigned inputs and larger DNNs. Positive/negative inputs (*e.g.*, those in BERT) are processed in separate cycles, reducing through-

put gains, and small filters in ShuffleNet and MobileNet poorly utilize the large crossbars of RAELLA.

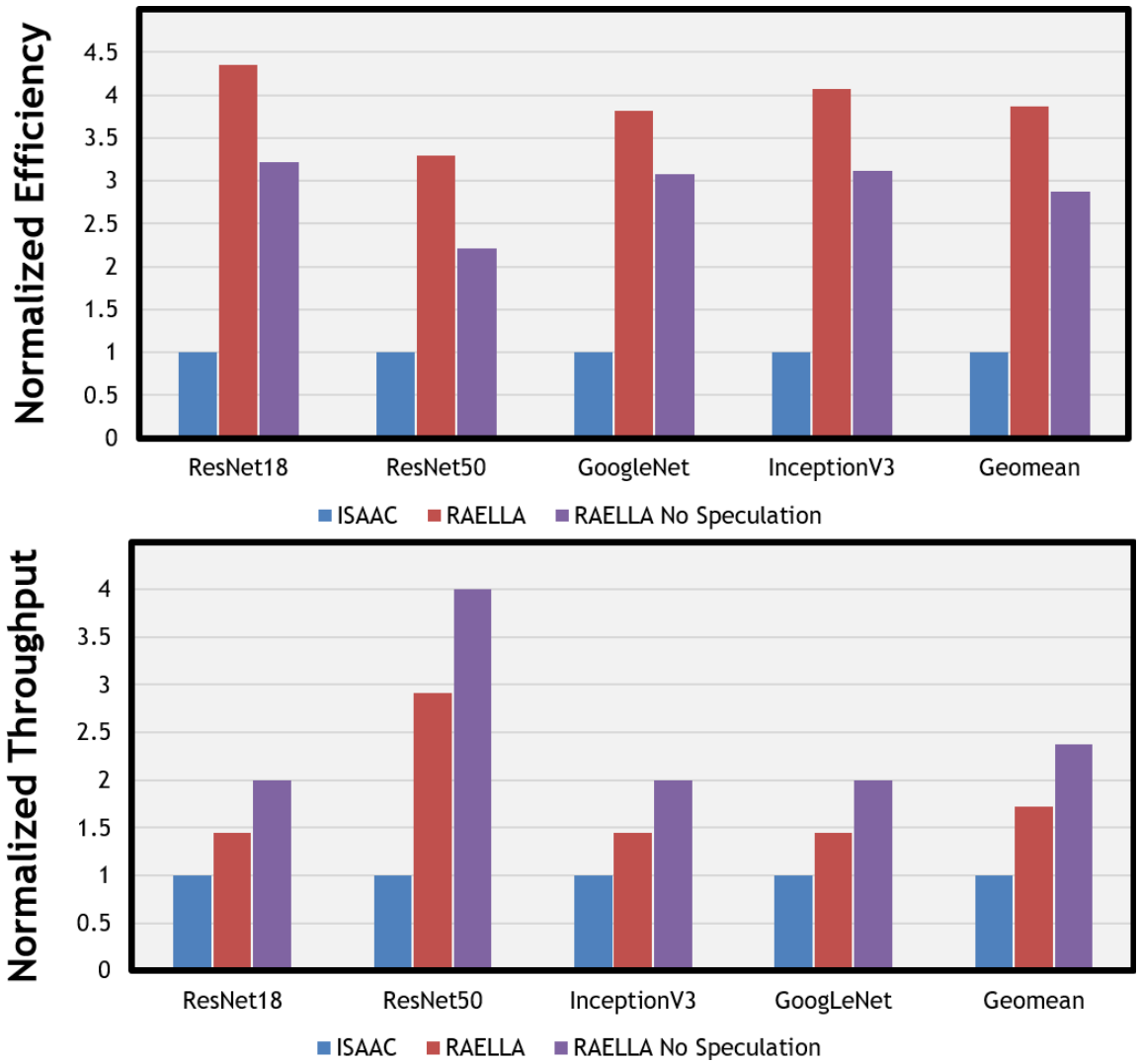

Figure 6-1: Efficiency and throughput normalized to the ISAAC architecture. Both architectures run DNNs without retraining. RAELLA with/without speculation increases efficiency 3.9 × /2.8× and throughput by 2.0 × /2.7× geomean.

6.4 Comparison with Retraining Architectures

RAELLA is compared to TIMELY and FORMS-8. We show geomean ResNet18/ResNet50 results since we have data for these DNNs on all baselines. RAELLA runs off-the-shelf models, while FORMS [107] runs pruned-retrained versions and TIMELY [37] runs requantized-retrained versions.

Fig. 6-2 compares RAELLA's efficiency/throughput to FORMS and TIMELY. RAELLA is able to match the throughput of FORMS and exceed the efficiency of both FORMS and TIMELY. In the TIMELY comparison, we find that 65nm RAELLA is more efficient without speculation. This is because 65nm-RAELLA uses TIMELY's analog components, including TIMELY's highly efficient ADC. Speculation is useful when ADC energy dominates, but the tradeoffs may not be worthwhile if the ADC is not a major contributor to overall energy.

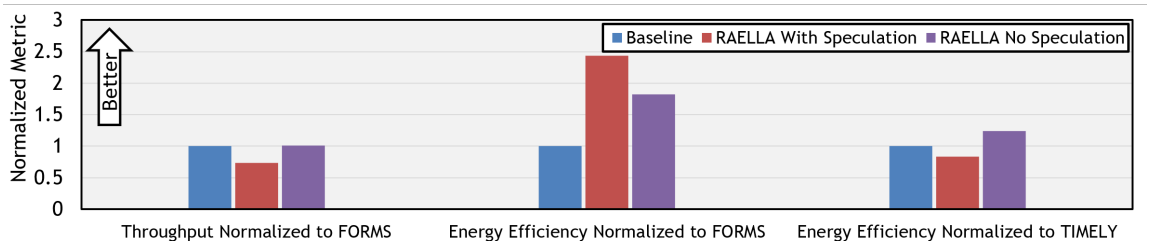

Figure 6-2: Comparison with FORMS and TIMELY. FORMS and TIMELY run retrained DNNs. RAELLA offers competitive/superior throughput/efficiency without retraining.

6.5 Accuracy Comparison

We compare RAELLA with three baselines. RAELLA Center+Offset is the standard RAELLA, configured with a 0.09 error budget. To showcase the benefits of Center+Offset, we compare it with RAELLA Zero+Offset, which implements a common-practice differential encoding (described in Section 4.1) by setting centers to zero. We use the same slicings for RAELLA Center+Offset and RAELLA Zero+Offset to match efficiency/throughput. Additionally, we show the reported accuracy of FORMS and TIMELY after retraining.

Table 6.1 shows the accuracy results. RAELLA with Center+Offest encoding causes little to no accuracy loss. Zero+Offset (differential encoding) causes substantial accuracy degradation due to high ADC saturation rates, as described in Section 4.1. Zero+Offset accuracy drop varies greatly across DNNs due to varying filter weight distributions. TIMELY and FORMS recover from accuracy loss by retraining DNNs.

	RAELLA Center+Offset	RAELLA Zero+Offset	FORMS [107]	TIMELY [37]
Retrained	**No**	**No**	**Yes**	**Yes**
Accuracy Drop %. Negative is accuracy gain.				
ResNet18	0.06	0.16	0.62	≤ 0.1
ResNet50	-0.08	0.30	0.70	≤ 0.1
MobileNetV2	0.03	10.17	-	-
ShuffleNetV2	0.14	16.36	-	-
GoogLeNet	-0.02	1.53	-	-
InceptionV3	-0.03	3.72	-	-
BERT-Large	0.12	0.46	-	-

Table 6.1: Accuracy Comparison. BERT-Large compares F1 loss, while others compare Imagenet Top-5 loss. Zero+Offset causes high accuracy loss; Center+Offset is essential to preserve accuracy. FORMS and TIMELY retrain, while RAELLA maintains low accuracy loss without retraining.

6.6 Summary

In this chapter, we compare RAELLA to existing baselines to demonstrate the benefits of RAELLA's strategies.

We compare RAELLA to non-DNN-modifying architecture ISAAC, showing that RAELLA can improve energy efficiency by up to 4.9× (geomean 3.9×) and throughput by up to 3.3× (geomean 2.0×) without changing DNNs. We then compare RAELLA to Weight-Count-Limited and Sum-Fidelity-Limited accelerators. In comparison to these architectures that require DNN retraining to recover accuracy, RAELLA is shown to provide similar efficiency and throughput while avoiding expensive DNN retraining. Finally, we demonstrate the low accuracy loss DNNs running on RAELLA for a set of representative DNNs and show the accuracy benefits of RAELLA's Center+Offset compared to a common-practice differential encoding.

Chapter 7

Ablation Studies

In this chapter, we isolate the effects of each of RAELLA's strategies through ablation studies. In the energy ablation, we test the efficiency benefits of each of RAELLA's strategies. In the accuracy ablation, we test RAELLA's strategies against increasing analog noise.

7.1 Ablation Setups

We begin with an ISAAC architecture and apply strategies sequentially. All test setups maintain high fidelity. The four test setups are the following:

- ISAAC: an 8b ISAAC. 128×128 crossbars, unsigned arithmetic. Four 2b weight slices, eight 1b input slices. 8b ADC.
- Center+Offset: previous setup, plus crossbar size increased to 512×512 2T2R with Center+Offset arithmetic. ADC resolution is reduced to 7b.
- Center+Offset, Adaptive Weight Slicing: previous setup, plus weight slicings

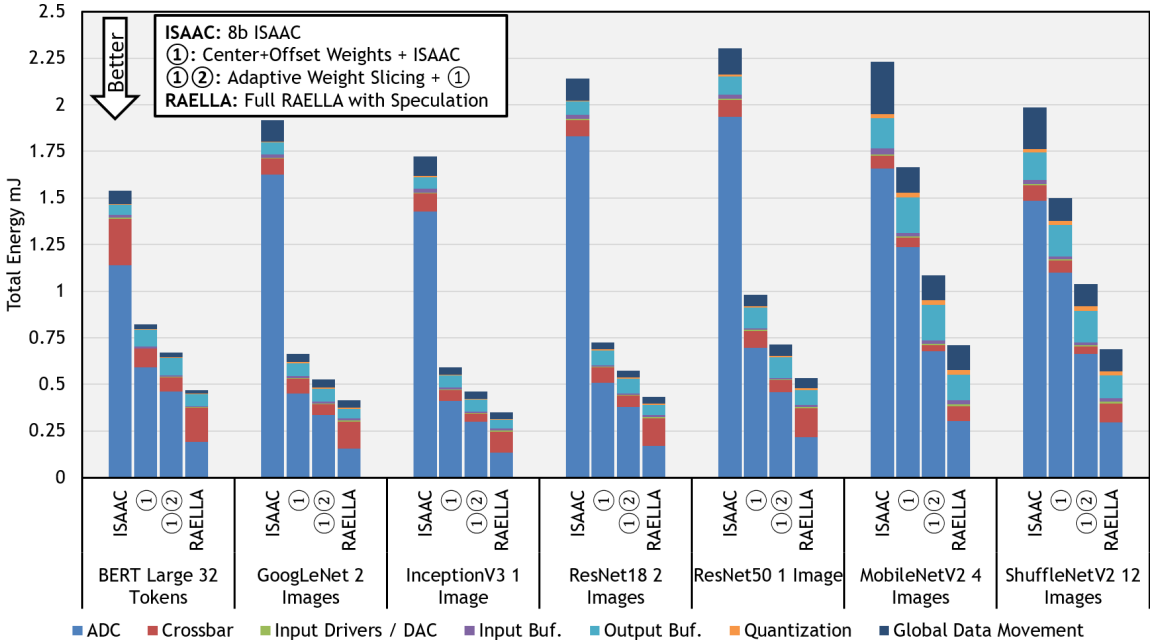

Figure 7-1: Energy Ablation. Each of RAELLA's strategies increases PIM architecture efficiency. Batch size is varied across DNNs to keep overall energy in the same range.

are chosen per-layer following Section 4.2.2. Most layers use three weight slices in a 4b-2b-2b pattern.

- RAELLA: previous setup, plus Dynamic Input Slicing and speculation enabled. RAELLA's registers/networks are added. RAELLA runs a 2-4 bit speculation input slice followed by 2-4 one-bit recovery input slices. In recovery cycles, ADCs do not convert columns where speculation succeeded.

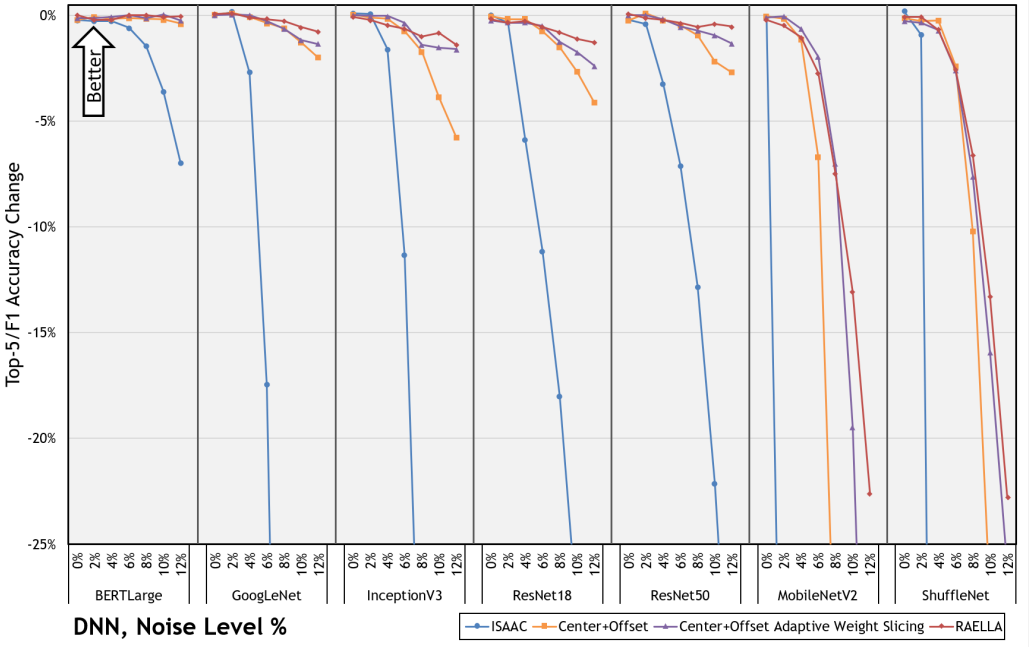

Figure 7-2: Accuracy drop at increasing analog noise. Center+Offset and Adaptive Weight Slicing increase noise tolerance. Dynamic Input Slicing maintains accuracy despite speculation failures; recovery prevents accuracy loss.

7.2 Energy Ablation

Fig. 7-1 shows the following results:

- ISAAC: ADCs dominate overall energy. *Converts/MAC* is 0.25. Per-component energy breakdown varies depending on DNN input/weight values, crossbar utilization, and digital data movement requirements.

- Center+Offset: enables a 4× scale-up in crossbar rows/columns and reduces ADC resolution. Center+Offset bit sparsity lowers crossbar energy. Large crossbars decrease data movement energy and reduce *Converts/MAC* from 0.25 to 0.063. Digital center processing, which requires one input addition and one multiply/subtract per several hundred MACs, contributes negligible energy.

- Adaptive Weight Slicing: reduces ADC energy $\sim 25\%$ as most layers use three weight slices instead of four. *Converts/MAC* is reduced to 0.047.

- Speculation: reduces ADC energy by 60%. Increases crossbar/DAC energy due to speculation cycles. Increases the input buffer energy due to 2× fetches. Usually decreases output buffer energy due to fewer psum writebacks. *Converts/MAC* is 0.018.

7.3 Accuracy and In-Crossbar Noise Ablation

Using the same four ablation setups, we evaluate DNN accuracy running on RAELLA with varying levels of noise. All PIM architectures suffer from analog variation and noise, but RAELLA tolerates noise with lower accuracy loss.

We model variation and noise as a Gaussian distribution that we add to column sums [25]. Given positive/negative sliced product sums N_+ and N_-, we model the column sum as $\mathcal{N}(\mu, \sigma^2)$. For noise level E, we set the mean μ to the ideal column sum $(N_+ - N_-)$ and the standard deviation $\sigma = E\sqrt{N_+ + N_-}$. After calculating a column sum, it is sent to an ADC and will be saturated at $[-64, 64)$ if out of range. Noise is additive across positive/negative sliced products. We test with an error of up to (12%), or $\sigma \approx 4$ for 512 $2b \times 2b$ MACs.

We make two changes to ISAAC to improve noise tolerance for a fair comparison. ISAAC's encoding strategy relies on an analog circuit that sums crossbar inputs [72]. This component has been shown to degrade accuracy under noise [99], so we replace it with a digital equivalent. For the accuracy of the BERT, we give ISAAC two cycles to process positive/negative inputs, matching RAELLA. This provides additional noise resistance. Fig. 7-2 shows the following:

- ISAAC: all DNNs suffer high accuracy loss for noise $> 4\%$. ISAAC uses unsigned weights, which have dense high-order bits (Fig. 4-5). Dense bits generate larger and higher noise values, and noise in high-order slices creates large errors in the results.

- Center+Offset: this is critical. Offset encoding provides noise resistance [99], and Center+Offset increases bit sparsity and decreases noise. Intuitively, digital center processing moves much of the computation out of the noisy analog domain.

- Adaptive Weight Slicing: accuracy is further improved. RAELLA's empirical slicing strategy is noise-aware, allowing RAELLA to adapt slicing to varying levels of noise. As noise increases, Adaptive Weight Slicing uses fewer bits per

weight slice to reduce error, with five weight slices for most layers at the highest tested noise.

- RAELLA: with speculation, RAELLA maintains accuracy similar to that of a no-speculation strategy. The recovery step prevents accuracy drop due to failed speculations.

We find that RAELLA can maintain DNN accuracy at higher noise levels, while on ISAAC, all DNNs suffer sharp accuracy loss at lower noise levels. Compact DNNs suffer higher accuracy degradation from errors compared to larger DNNs [101]. BERT uniquely benefits from the sparsity generated by two-cycle positive/negative inputs. This, along with BERT's large size, allows BERT to maintain better accuracy at high noise levels.

RAELLA can adapt to varying noise; adaptive weight slicing automatically trades off storage density and efficiency for correctness by using fewer bits per slice in higher-noise scenarios. This lets RAELLA maintain accuracy without retraining while extracting as much efficiency as possible under noise constraints.

7.4 Summary

In this chapter, we demonstrate the benefits of each of RAELLA's strategies. We present an ablation study that progressively applies RAELLA's strategies to ISAAC, a non-retraining architecture. We first show energy results, exploring how the energy consumption of each component changes for each optimization and DNN and showing the system-level energy benefits of each of RAELLA's strategies.

Next, we show accuracy results, which again applies each of RAELLA's strategies to ISAAC, testing DNN accuracy against increasing analog noise. We show that

RAELLA's optimizations protect DNNs from analog noise, permitting high accuracy and significantly increased noise resistance relative to ISAAC.

Chapter 8

Conclusion

PIM accelerators have the potential to efficiently compute DNNs, but many PIM accelerators are limited by the high energy and area costs of ADCs. Published architectures have often relied on Weight-Count-Limited or Sum-Fidelity-Limited approaches to reduce ADC area and energy, which modify DNNs. Modifying the DNN may cause accuracy loss or require retraining to recover accuracy. In this thesis, we explore an alternative by introducing retraining-free approaches to reduce ADC area/energy and gain efficiency without modifying the DNNs. The main contributions of this thesis are:

- **An analysis and breakdown of ADC energy factors via the Titanium Law:** ADCs consume a significant portion of the energy of PIM accelerators. To enable efficient PIM inference, it is essential to understand the available tradeoffs and how they affect ADC energy. In this thesis, we have presented the Titanium Law, which breaks down ADC energy into its constituent terms and provides guidance on how to modify each of these terms. Through the lens of the Titanium Law, we look at prior work to see why some works have paid

high ADC energy and why other works have turned to modifying the DNN and retraining to reduce this ADC energy.

- **Center+Offset encoding to reduce the resolution of accumulated analog values:** PIM crossbars can accumulate many values in the analog domain, but accumulating more values comes at a cost; accumulating more values increases the resolution of the resulting sums, thus increasing the required ADC resolution and ADC area/energy needed to read those sums. Center+Offset works to eliminate this effect by partitioning computation between the analog and digital domains. In the analog domain, it makes accumulated values near-zero on average. This allows us to sum many values and keep sums small and low resolution. To generate near-zero-average values, Center+Offset shifts the values of DNN weights in each filter in each crossbar, attempting to equalize the average magnitude of positive and negative weight slices. This creates near-zero-average weight slices, leading to near-zero-average sliced products and near-zero-average column sums. Meanwhile, in the digital domain, Center+Offset uses a low-overhead strategy to process high-resolution values. In this thesis, we have shown how to shift weights to implement Center+Offset encoding, how to implement Center+Offset arithmetic in hardware, and how to calculate optimal centers such that the average positive and negative weight slice magnitudes are balanced. We have compared Center+Offset to common-practice differential encodings, showing a significant DNN accuracy benefit from Center+Offset. Finally, we have broken down the energy and DNN accuracy impacts of Center+Offset, showing that Center+Offset can increase the efficiency of PIM accelerators and increase the noise tolerance of DNNs running on PIM accelerators.

- **Adaptive Weight Slicing to use the most efficient and highest density weight slicing for each DNN layer:** DNN layers can vary in many aspects: the number of weights in a filter, input distributions, output distributions, quantization parameters, and other factors. Furthermore, individual bit indices within each DNN layer have different proportions of one or zero bits, meaning that even within the same DNN layer, some vectors of slices may have different value distributions than others. Because of this, some layers can use fewer (denser, more efficient) weight slices while maintaining high fidelity, whereas other layers require more (less dense, less efficient) weight slices. Adaptive Weight Slicing allows architectures to use the most efficient, highest-density weight slicing possible for each DNN layer. In this thesis, we have shown how to find the best slicing for each DNN layer by testing each possible slicing with a small set of ten test inputs. We have shown that, for most DNN layers, it is optimal to use three slices to store 8b weights. We have demonstrated that Adaptive Weight Slicing can increase the efficiency of PIM accelerators by finding more efficient slicings, and it can also improve noise tolerance by using more slices to counteract the effects of analog noise.

- **Dynamic Input Slicing to balance the efficiency of more bits per input slice and the small, low-resolution values of few bits per input slice:** Lastly, we present a method to dynamically slice DNN inputs. We demonstrate a speculation and recovery strategy. Speculation uses an aggressive-efficient slicing with more bits per input slice but a higher chance of ADC saturation. Recovery uses a conservative-correct slicing with fewer bits per input slice and a low chance of ADC saturation. With Dynamic Input Slicing, we run the efficient speculative slicing, then recover with the less-efficient slicing. To save

energy, we run the ADC only for columns that failed speculation. In this thesis, we have shown the hardware support required to run speculation and recovery slicings and discussed the tradeoffs involved in running Dynamic Input Slicing. We have shown that on various DNNs, 98% of the column sums are captured correctly and efficiently in speculation, while incorrect column sums are recovered. We have provided a breakdown of accelerator energy, showing that Dynamic Input Slicing decreases ADC energy by 60% on various DNNs and can increase overall PIM accelerator efficiency. Finally, we have demonstrated that recovery can effectively protect from incorrectly-speculated results, allowing Dynamic Input Slicing to preserve high fidelity and high DNN accuracy.

Using the trade-offs and optimizations explored in this thesis, we create the RAELLA architecture. RAELLA reduces the area and energy of ADCs without retraining or modifying the DNNs. This allows RAELLA to achieve significantly higher efficiency and throughput than non-DNN-retraining architectures. Additionally, RAELLA achieves similar efficiency and throughput to Sum-Fidelity-Limited and Weight-Count-Limited architectures while avoiding expensive DNN modification or retraining.

8.1 Future Work

To build upon the work in this thesis, we plan to:

- **Evaluate the effects of different Sum-Fidelity-Limited approaches on DNNs.** Sum-Fidelity-Limited approaches make decisions that restrict the resolution and/or quantization of column sums, which will ultimately have an effect on DNN accuracy. However, DNN accuracy loss due to these approaches

is often not reported in published designs. There is a need for a clear understanding of the tradeoffs between Sum-Fidelity-Limited approaches and DNN accuracy. It would be valuable to explore these tradeoffs to develop a systemic understanding and taxonomy of Sum-Fidelity-Limited approaches to guide future hardware design choices.

- **Test RAELLA's strategies in combination with retraining-oriented strategies.** RAELLA's strategies do not require retraining themselves, though they may synergize with the retraining-requiring strategies employed by Sum-Fidelity-Limited and Weight-Count-Limited architectures. It would be interesting to see how these strategies can combine to potentially yield even greater benefits in efficiency and throughput.

- **Fold Center+Offset Digital Arithmetic into Quantization.** Center+Offset shifts weights by some value, using digital circuitry to correct the computed results. Interestingly, the arithmetic it performs is identical to that of some linear quantization strategies [109] that use a calculated *quantization zero* to shift DNN weights. It may be possible to incorporate RAELLA's Center+Offset arithmetic into this quantization zero calculation, enabling Center+Offset to operate without any extra digital arithmetic.

8.2 Conclusion

It is commonly assumed in PIM accelerator designs that DNN workloads will work well given the computation executed by the PIM accelerator. When the computation executed by the PIM accelerator is Weight-Count-Limited or Sum-Fidelity-Limited, the assumption is that the DNN will be modified to work well with the given compu-

tation. With Weight-Count-Limited architectures, this can mean pruning the DNN, and with Sum-Fidelity-Limited architectures, this can mean requantizing the DNN. Both can involve retraining.

RAELLA presents an alternative. Rather than modifying the DNN to run well on a PIM accelerator, RAELLA adapts the computation performed by the accelerator to gain more efficiency and throughput from the given DNN. RAELLA's optimizations exploit simple characteristics of DNN operand value distributions. These characteristics are found in a variety of DNNs, allowing RAELLA to gain efficiency and throughput while running these DNNs off-the-shelf.

As an analogy, we can think of scratchpads versus caches in computer architecture. Scratchpads are memory units that are explicitly managed by the software, while caches are memory units that are managed in the background by the hardware.

Like scratchpads, Weight-Count-Limited/Sum-Fidelity-Limited architectures can require explicit modification of software to gain efficiency, and each hardware comes with different setups that may or may not be compatible with any given workload.

Like caches, RAELLA offers its improvements flexibly. Caches, based on the almost-always-true observation that software follows locality, can improve performance off-the-shelf for most workloads. If workloads do not satisfy the locality assumption, cache efficiency may decrease, but caches continue to preserve correct execution.

The takeaway of this thesis is not that we should avoid modifying DNNs. Modifying DNNs can be a valuable step in codesign, just as scratchpads are essential to maximize the performance of some programs. Rather, the takeaway of this thesis is that by thinking about our computations, architecture, and workload together, we can develop strategies that can increase efficiency while easing restrictions on the workloads that PIM accelerators run. To this end, this thesis explores the following

key insights.

The actual values that operands take on are more important than the maximum resolution of those operands. In the analog domain, where all values are represented as continuous variables, an 8b value 00000000 is identical to a 1b value 0. By thinking about the computation in terms of values instead of maximum resolution, we no longer must budget our hardware to handle the worst-case values generated by our computation. Instead, we budget hardware for the values actually generated. For example, based on maximum resolution, we would think that the product of two 4b numbers is an 8b number, and the sum of 512 of these 8b numbers is up to 17b. However, RAELLA often performs 512-element vector-vector multiplications with 4b operands and converts results with a 7b ADC. This is possible because the values generated are small, despite their resolution. As another example, thinking about operand values permits the lightweight adaptability employed in Adaptive Weight Slicing and Dynamic Input Slicing. Both of these strategies vary the bits per slice, but they do so through values, recognizing that a 2b slice can be seen as a 4b slice with the two most significant bits set to zero. This thinking allows for lightweight adaptability without changing the resolution of analog hardware.

The values generated by computations are tied to how the computation is performed. As architecture designers, we can choose how accelerators perform computation, and therefore we have the opportunity to affect the values that computations generate. In this thesis, we consider multiple ways of modifying the computation. We consider encoding strategies such as Center+Offset and differential encodings that change the representation of DNN weights. We also explore slicing strategies that divide large computations into multiple smaller parts. We have shown how these factors modify computation both adaptively and dynamically, allowing us to affect the values generated by analog computations. Given that we will ultimately need to

pay for an ADC to read the computed analog values, how to perform computation should be among the most important decisions to make when designing any PIM accelerator.

If operands follow known distributions, we can gain efficiency by taking advantage of these distributions. In this thesis, we take advantage of the distributions of DNN weights, inputs, and generated column sums. Center+Offset exploits the DNN weight distribution, which is usually symmetric, to better balance the average positive and negative weight slice magnitudes. Adaptive Weight Slicing also exploits DNN weight distribution, where weights are usually small, using small values to generate more efficient slicings. Dynamic Input Slicing exploits the DNN input distribution, where DNN inputs are usually small, to use more bits in speculation slices. Finally, all optimizations rely on the column sum distribution. When the column sum values are tightly distributed in a narrow range, we can represent column sums well with a low-resolution ADC. In addition to the opportunities we explore, there remain many interesting opportunities in value distributions. For example, after RAELLA's strategies, column sums are tightly distributed around zero, and only a small percentage of column sums approach the bounds of the ADC range. Can we design an ADC that leverages this knowledge to convert smaller column sums more efficiently?

Adaptable computation can extract more efficiency from workloads while also preserving correct computation. In this thesis, we place correct computation as a first-order goal, designing RAELLA to gain efficiency only when it can do so while maintaining high fidelity and correctly-computed results. To achieve this goal, we introduce adaptive and dynamic strategies. Adaptive Weight Slicing measures error, using the most efficient slicing possible that does not exceed a given error budget. Dynamic Input Slicing uses a speculate-and-recover strategy that speculates efficiently

and recovers erroneously computed values. In both of these strategies, RAELLA is able to modify its computation to avoid loss of fidelity.

These insights can be combined to inform architecture design. We know that values matter. We also know that values depend on both the data with which we compute and the computations themselves. We can measure the distributions of the data with which we compute, and we can choose how we perform the computations. It follows that if we would like to achieve outcomes with our computed values, such as high fidelity and high efficiency, we can measure the distributions of data with which we compute and then adapt our computation to best achieve our goals.

This line of thinking led to the RAELLA architecture. There are many more opportunities to improve on any or all of these aspects. New workloads may introduce different value distributions or computations, while new hardware or new encoding strategies may express existing computations in different ways. New synergies between these aspects may lead to novel accelerator architectures.

We hope that this thesis will encourage future designers to incorporate these key insights into PIM accelerators. We also hope that this thesis will inspire future work to build upon the presented strategies and extract even more efficiency and throughput from PIM workloads.

Bibliography

[1] Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, and Reetuparna Das. Compute caches. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 481–492, 2017.

[2] Fabien Alibart, Ligang Gao, Brian D Hoskins, and Dmitri B Strukov. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology*, 23(7):075201, jan 2012.

[3] Tanner Andrulis, Joel S. Emer, and Vivienne Sze. Raella: Reforming the arithmetic for efficient, low-resolution, and low-loss analog pim: No retraining required! In *2023 ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2023.

[4] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R. Stanley Williams, Paolo Faraboschi, Wen-mei W Hwu, John Paul Strachan, Kaushik Roy, and Dejan S. Milojicic. 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 '19, page 715–731, New York, NY, USA, 2019. Association for Computing Machinery.

[5] Han Bao, Yifan Qin, Jia Chen, Ling Yang, Jiancong Li, Houji Zhou, Yi Li, and Xiangshui Miao. Quantization and sparsity-aware processing for energy-efficient nvm-based convolutional neural networks. In *Frontiers in Electronics*, 2022.

[6] Sathwika Bavikadi, Purab Ranjan Sutradhar, Khaled N. Khasawneh, Amlan Ganguly, and Sai Manoj Pudukotai Dinakarrao. A review of in-memory computing architectures for machine learning applications. In *Proceedings of the*

2020 on Great Lakes Symposium on VLSI, GLSVLSI '20, page 89–94, New York, NY, USA, 2020. Association for Computing Machinery.

[7] Pai-Yu Chen, Xiaochen Peng, and Shimeng Yu. Neurosim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures. In *2017 IEEE International Electron Devices Meeting (IEDM)*, pages 6.1.1–6.1.4, 2017.

[8] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622, 2014.

[9] Yuzong Chen, Lu Lu, Bongjin Kim, and Tony Tae-Hyoung Kim. Reconfigurable 2t2r reram architecture for versatile data storage and computing in-memory. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(12):2636–2649, 2020.

[10] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 27–39, 2016.

[11] Jungwook Choi, Swagath Venkataramani, Vijayalakshmi (Viji) Srinivasan, Kailash Gopalakrishnan, Zhuo Wang, and Pierce Chuang. Accurate and efficient 2-bit quantized neural networks. In A. Talwalkar, V. Smith, and M. Zaharia, editors, *Proceedings of Machine Learning and Systems*, volume 1, pages 348–359, 2019.

[12] Teyuh Chou, Wei Tang, Jacob Botimer, and Zhengya Zhang. Cascade: Connecting rrams to extend analog dataflow in an end-to-end in-memory processing paradigm. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, page 114–125, New York, NY, USA, 2019. Association for Computing Machinery.

[13] Chaoqun Chu, Yanzhi Wang, Yilong Zhao, Xiaolong Ma, Shaokai Ye, Yunyan Hong, Xiaoyao Liang, Yinhe Han, and Li Jiang. Pim-prune: Fine-grain dcnn pruning for crossbar-based process-in-memory architecture. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[14] Philip Colangelo, Nasibeh Nasiri, Eriko Nurvitadhi, Asit Mishra, Martin Margala, and Kevin Nealis. Exploration of low numeric precision deep learning inference using intel® fpgas. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 73–80, 2018.

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[16] Lei Deng, Ling Liang, Guanrui Wang, Liang Chang, Xing Hu, Xin Ma, Liu Liu, Jing Pei, Guoqi Li, and Yuan Xie. Semimap: A semi-folded convolution mapping for speed-overhead balance on crossbars. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(1):117–130, 2020.

[17] Amin Farmahini-Farahani, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. Nda: Near-dram acceleration architecture leveraging commodity dram devices and standard memory modules. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 283–295, 2015.

[18] Andrea Fasoli, Chia-Yu Chen, Mauricio Serrano, Xiao Sun, Naigang Wang, Swagath Venkataramani, George Saon, Xiaodong Cui, Brian Kingsbury, Wei Zhang, Zoltán Tüske, and Kailash Gopalakrishnan. 4-Bit Quantization of LSTM-Based Speech Recognition Models. In *Proc. Interspeech 2021*, pages 2586–2590, 2021.

[19] Ligang Gao, Fabien Alibart, and Dmitri B. Strukov. A high resolution nonvolatile analog memory ionic devices. 2013.

[20] Sujan K. Gonugondla, Charbel Sakr, Hassan Dbouk, and Naresh R. Shanbhag. Fundamental limits on the precision of in-memory architectures. In *Proceedings of the 39th International Conference on Computer-Aided Design*, ICCAD '20, New York, NY, USA, 2020. Association for Computing Machinery.

[21] Mengyu Guo, Zihan Zhang, Jianfei Jiang, Qin Wang, and Naifeng Jing. Boosting reram-based dnn by row activation oversubscription. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 604–609, 2022.

[22] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[23] Mingxuan He, Choungki Song, Ilkon Kim, Chunseok Jeong, Seho Kim, Il Park, Mithuna Thottethodi, and T. N. Vijaykumar. Newton: A dram-maker's accelerator-in-memory (aim) architecture for machine learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 372–385, 2020.

[24] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14, 2014.

[25] Miao Hu, John Paul Strachan, Zhiyong Li, Emmanuelle M. Grafals, Noraica Davila, Catherine Graves, Sity Lam, Ning Ge, Jianhua Joshua Yang, and R. Stanley Williams. Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2016.

[26] Houxiang Ji, Linghao Song, Li Jiang, Hai Li, and Yiran Chen. Recom: An efficient resistive accelerator for compressed deep neural networks. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 237–240, 2018.

[27] Norman P. Jouppi, Andrew B. Kahng, Naveen Muralimanohar, and Vaishnav Srinivas. Cacti-io: Cacti with off-chip power-area-timing models. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(7):1254–1267, 2015.

[28] Tinu Theckel Joy, Santu Rana, Sunil Gupta, and Svetha Venkatesh. Hyperparameter tuning for big data using bayesian optimisation. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2574–2579, 2016.

[29] Jin Hyun Kim, Shin-haeng Kang, Sukhan Lee, Hyeonsu Kim, Woongjae Song, Yuhwan Ro, Seungwon Lee, David Wang, Hyunsung Shin, Bengseng Phuah, Jihyun Choi, Jinin So, YeonGon Cho, JoonHo Song, Jangseok Choi, Jeonghyeon Cho, Kyomin Sohn, Youngsoo Sohn, Kwangil Park, and Nam Sung Kim.

Aquabolt-xl: Samsung hbm2-pim with in-memory processing for ml accelerators and beyond. In *2021 IEEE Hot Chips 33 Symposium (HCS)*, pages 1–26, 2021.

[30] Sangjin Kim, Zhiyong Li, Soyeon Um, Wooyoung Jo, Sangwoo Ha, Juhyoung Lee, Sangyeob Kim, Donghyeon Han, and Hoi-Jun Yoo. 16.5 dynaplasia: An edram in-memory-computing-based reconfigurable spatial accelerator with triple-mode cell for dynamic resource switching. In *2023 IEEE International Solid- State Circuits Conference (ISSCC)*, pages 256–258, 2023.

[31] Sangyeob Kim, Sangjin Kim, Soyeon Um, Soyeon Kim, Kwantae Kim, and Hoi-Jun Yoo. Neuro-cim: A 310.4 tops/w neuromorphic computing-in-memory processor with low wl/bl activity and digital-analog mixed-mode neuron firing. In *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pages 38–39, 2022.

[32] Michael Klachko, Mohammad Reza Mahmoodi, and Dmitri Strukov. Improving noise tolerance of mixed-signal neural networks. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[33] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *CoRR*, abs/1806.08342, 2018.

[34] Lukas Kull, Thomas Toifl, Martin Schmatz, Pier Andrea Francese, Christian Menolfi, Matthias Braendli, Marcel Kossel, Thomas Morf, Toke Meyer Andersen, and Yusuf Leblebici. A 3.1mw 8b 1.2gs/s single-channel asynchronous sar adc with alternate comparators for enhanced speed in 32nm digital soi cmos. In *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pages 468–469, 2013.

[35] Hunjun Lee, Minseop Kim, Dongmoon Min, Joonsung Kim, Jongwon Back, Honam Yoo, Jong-Ho Lee, and Jangwoo Kim. 3d-fpim: An extreme energy-efficient dnn acceleration system using 3d nand flash-based in-situ pim unit. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1359–1376, 2022.

[36] Boxun Li, Lixue Xia, Peng Gu, Yu Wang, and Huazhong Yang. Merging the interface: Power, area and accuracy co-optimization for rram crossbar-based mixed-signal computing system. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2015.

[37] Weitao Li, Pengfei Xu, Yang Zhao, Haitong Li, Yuan Xie, and Yingyan Lin. Timely: Pushing data movements and interfaces in pim accelerators towards local and in time domain. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, ISCA '20, page 832–845. IEEE Press, 2020.

[38] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021.

[39] Jie Lin and Jiann-Shiun Yuan. Analysis and simulation of capacitor-less reram-based stochastic neurons for the in-memory spiking neural network. *IEEE Transactions on Biomedical Circuits and Systems*, 12(5):1004–1017, 2018.

[40] Jilan Lin, Zhenhua Zhu, Yu Wang, and Yuan Xie. Learning the sparsity for reram: Mapping and pruning sparse neural network for reram based accelerator. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ASPDAC '19, page 639–644, New York, NY, USA, 2019. Association for Computing Machinery.

[41] Yu-Hsuan Lin, Chao-Hung Wang, Ming-Hsiu Lee, Dai-Ying Lee, Yu-Yu Lin, Feng-Min Lee, Hsiang-Lan Lung, Keh-Chung Wang, Tseung-Yuen Tseng, and Chih-Yuan Lu. Performance impacts of analog reram non-ideality on neuromorphic computing. *IEEE Transactions on Electron Devices*, 66(3):1289–1295, 2019.

[42] Eike Linn, Roland Rosezin, Carsten Kügeler, and Rainer Waser. Complementary resistive switches for passive nanocrossbar memories. *Nature Materials*, 9(5):403–406, May 2010.

[43] Qi Liu, Bin Gao, Peng Yao, Dong Wu, Junren Chen, Yachuan Pang, Wenqiang Zhang, Yan Liao, Cheng-Xin Xue, Wei-Hao Chen, Jianshi Tang, Yu Wang, Meng-Fan Chang, He Qian, and Huaqiang Wu. 33.2 a fully integrated analog reram based 78.4tops/w compute-in-memory chip with fully parallel mac computing. In *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 500–502, 2020.

[44] Xiaoxiao Liu, Mengjie Mao, Beiye Liu, Hai Li, Yiran Chen, Boxun Li, Yu Wang, Hao Jiang, Mark Barnell, Qing Wu, and Jianhua Yang. RENO: A high-efficient reconfigurable neuromorphic computing accelerator design. In

2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), pages 1–6, June 2015. ISSN: 0738-100X.

[45] Anni Lu, Xiaochen Peng, Wantong Li, Hongwu Jiang, and Shimeng Yu. Neurosim validation with 40nm rram compute-in-memory macro. In *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 1–4, 2021.

[46] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[47] Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, page 1485–1488, New York, NY, USA, 2010. Association for Computing Machinery.

[48] Matthew J. Marinella, Sapan Agarwal, Alexander Hsia, Isaac Richter, Robin Jacobs-Gedrim, John Niroula, Steven J. Plimpton, Engin Ipek, and Conrad D. James. Multiscale co-design analysis of energy, latency, area, and accuracy of a reram analog neural training accelerator. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(1):86–101, 2018.

[49] Bradley McDanel, Sai Qian Zhang, and H. T. Kung. Saturation rram leveraging bit-level sparsity resulting from term quantization. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[50] Sparsh Mittal. A survey of reram-based architectures for processing-in-memory and neural networks. *Machine Learning and Knowledge Extraction*, 1(1):75–114, 2019.

[51] Sparsh Mittal, Gaurav Verma, Brajesh Kaushik, and Farooq A Khanday. A survey of sram-based in-memory computing techniques and applications. *Journal of Systems Architecture*, 119:102276, 2021.

[52] Boris Murmann. Energy limits in a/d converters. In *2013 IEEE Faible Tension Faible Consommation*, pages 1–4, 2013.

[53] Anirban Nag, Rajeev Balasubramonian, Vivek Srikumar, Ross Walker, Ali Shafiee, John Paul Strachan, and Naveen Muralimanohar. Newton: Gravitating towards the physical limits of crossbar acceleration. *IEEE Micro*, 38(5):41–49, 2018.

[54] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. 2019.

[55] M. O'Halloran and R. Sarpeshkar. A 10-nw 12-bit accurate analog storage cell with 10-aa leakage. *IEEE Journal of Solid-State Circuits*, 39(11):1985–1996, 2004.

[56] Atsuya Okazaki, Pritish Narayanan, Stefano Ambrogio, Kohji Hosokawa, Hsinyu Tsai, Akiyo Nomura, Takeo Yasuda, Charles Mackin, Alexander Friz, Masatoshi Ishii, Yasuteru Kohda, Katie Spoon, An Chen, Andrea Fasoli, Malte J. Rasch, and Geoffrey W. Burr. Analog-memory-based 14nm hardware accelerator for dense deep neural networks including transformers. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 3319–3323, 2022.

[57] Shunsuke Okumura, Makoto Yabuuchi, Kenichiro Hijioka, and Koichi Nose. A ternary based bit scalable, 8.80 tops/w cnn accelerator with many-core processing-in-memory architecture with 896k synapses/mm2. In *2019 Symposium on VLSI Technology*, pages C248–C249, 2019.

[58] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 304–315, 2019.

[59] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[60] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training, 2021.

[61] J. Thomas Pawlowski. Hybrid memory cube (hmc). In *2011 IEEE Hot Chips 23 Symposium (HCS)*, pages 1–24, 2011.

[62] Xiaochen Peng, Shanshi Huang, Hongwu Jiang, Anni Lu, and Shimeng Yu. Dnn+neurosim v2.0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(11):2306–2319, 2021.

[63] Lillian Pentecost, Alexander Hankin, Marco Donato, Mark Hempstead, Gu-Yeon Wei, and David Brooks. Nvmexplorer: A framework for cross-stack comparisons of embedded non-volatile memories. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 938–956, 2022.

[64] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization, 2018.

[65] Ximing Qiao, Xiong Cao, Huanrui Yang, Linghao Song, and Hai Li. Atomlayer: A universal reram-based cnn accelerator with atomic layer computation. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6, 2018.

[66] Songyun Qu, Bing Li, Ying Wang, and Lei Zhang. Asbp: Automatic structured bit-pruning for rram-based nn accelerator. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 745–750, 2021.

[67] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics.

[68] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022.

[69] Babak Rokh, Ali Azarpeyvand, and Ali Reza Khanteymoori. A comprehensive survey on model quantization for deep neural networks. *ArXiv*, abs/2205.07877, 2022.

[70] Mehdi Saberi, Reza Lotfi, Khalil Mafinezhad, and Wouter A. Serdijn. Analysis of power consumption and linearity in capacitive digital-to-analog converters used in successive approximation adcs. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 58(8):1736–1748, 2011.

[71] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. pages 4510–4520, 06 2018.

[72] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 14–26, 2016.

[73] Naresh R. Shanbhag and Saion K. Roy. Benchmarking in-memory computing architectures. *IEEE Open Journal of the Solid-State Circuits Society*, 2:288–300, 2022.

[74] Xueyuan She, Yun Long, and Saibal Mukhopadhyay. Improving robustness of reram-based spiking neural network accelerator with stochastic spike-timing-dependent-plasticity. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[75] Mahmut E. Sinangil, Burak Erbagci, Rawan Naous, Kerem Akarvardar, Dar Sun, Win-San Khwa, Hung-Jen Liao, Yih Wang, and Jonathan Chang. A 7-nm compute-in-memory sram macro supporting multi-bit input, weight and output and achieving 351 tops/w and 372.4 gops. *IEEE Journal of Solid-State Circuits*, 56(1):188–198, 2021.

[76] Gagandeep Singh, Lorenzo Chelini, Stefano Corda, Ahsan Javed Awan, Sander Stuijk, Roel Jordans, Henk Corporaal, and Albert-Jan Boonstra. Near-memory computing: Past, present, and future. *Microprocessors and Microsystems*, 71:102868, 2019.

[77] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. Pipelayer: A pipelined reram-based accelerator for deep learning. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 541–552, 2017.

[78] Tao Song, Xiaoming Chen, Xiaoyu Zhang, and Yinhe Han. Brahms: Beyond conventional rram-based neural network accelerators using hybrid analog memory system. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 1033–1038, 2021.

[79] Zhuoran Song, Dongyue Li, Zhezhi He, Xiaoyao Liang, and Li Jiang. Reram-sharing: Fine-grained weight sharing for reram-based deep neural network accelerator. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[80] Katie Spoon, Hsinyu Tsai, An Chen, Malte J. Rasch, Stefano Ambrogio, Charles Mackin, Andrea Fasoli, Alexander M. Friz, Pritish Narayanan, Milos Stanisavljevic, and Geoffrey W. Burr. Toward software-equivalent accuracy on transformer-based deep neural networks with analog memory devices. *Frontiers in Computational Neuroscience*, 15, 2021.

[81] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. *Efficient Processing of Deep Neural Networks*. Springer International Publishing, 2020.

[82] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[83] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[84] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014.

[85] Minh S. Q. Truong, Eric Chen, Deanyone Su, Liting Shen, Alexander Glass, L. Richard Carley, James A. Bain, and Saugata Ghose. Racer: Bit-pipelined processing using resistive memory. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 100–116, New York, NY, USA, 2021. Association for Computing Machinery.

[86] Fengbin Tu, Weiwei Wu, Shouyi Yin, Leibo Liu, and Shaojun Wei. Rana: Towards efficient neural acceleration with refresh-optimized embedded dram. *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 340–352, 2018.

[87] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[88] Marian Verhelst and Boris Murmann. Area scaling analysis of cmos adcs. *Electronics Letters*, 48:314–315, 2012.

[89] R.H. Walden. Analog-to-digital converter survey and analysis. *IEEE Journal on Selected Areas in Communications*, 17(4):539–550, 1999.

[90] Ching-Hua Wang, Yi-Hung Tsai, Kai-Chun Lin, Meng-Fan Chang, Ya-Chin King, Chrong-Jung Lin, Shyh-Shyuan Sheu, Yu-Sheng Chen, Heng-Yuan Lee, Frederick T. Chen, and Ming-Jinn Tsai. Three-dimensional 4f2 reram cell with cmos logic compatible process. In *2010 International Electron Devices Meeting*, pages 29.6.1–29.6.4, 2010.

[91] Hechen Wang, Renzhi Liu, Richard Dorrance, Deepak Dasalukunte, Xiaosen Liu, Dan Lake, Brent Carlton, and May Wu. A 32.2 tops/w sram compute-in-memory macro employing a linear 8-bit c-2c ladder for charge domain computation in 22nm for edge inference. In *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pages 36–37, 2022.

[92] Linfang Wang, Wang Ye, Chunmeng Dou, Xin Si, Xiaoxin Xu, Jing Liu, Dashan Shang, Jianfeng Gao, Feng Zhang, Yongpan Liu, Meng-Fan Chang, and Qi Liu. Efficient and robust nonvolatile computing-in-memory based on voltage division in 2t2r rram with input-dependent sensing control. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(5):1640–1644, 2021.

[93] Wikipedia. Iron law of processor performance — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Iron%20law%20of%20processor%20performance&oldid=1112639388`, 2022. [Online; accessed 22-November-2022].

[94] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation. *ArXiv*, abs/2004.09602, 2020.

[95] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation, 04 2020.

[96] Yannan Nellie Wu, Joel S. Emer, and Vivienne Sze. Accelergy: An architecture-level energy estimation methodology for accelerator designs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2019.

[97] Yannan Nellie Wu, Vivienne Sze, and Joel S. Emer. An architecture-level energy and area estimator for processing-in-memory accelerator designs. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 116–118, 2020.

[98] T. Patrick Xiao, Christopher H. Bennett, Ben Feinberg, Sapan Agarwal, and Matthew J. Marinella. Analog architectures for neural network acceleration based on non-volatile memory. *Applied Physics Reviews*, 7(3), 07 2020. 031301.

[99] T. Patrick Xiao, Ben Feinberg, Christopher H. Bennett, Venkatraman Prabhakar, Prashant Saxena, Vineet Agrawal, Sapan Agarwal, and Matthew J. Marinella. On the accuracy of analog neural network inference accelerators [feature]. *IEEE Circuits and Systems Magazine*, 22:26–48, 2021.

[100] J. Joshua Yang, Dmitri B. Strukov, and Duncan R. Stewart. Memristive devices for computing. *Nature Nanotechnology*, 8(1):13–24, Jan 2013.

[101] Tien-Ju Yang and Vivienne Sze. Design considerations for efficient deep neural networks on processing-in-memory accelerators. pages 22.1.1–22.1.4, 12 2019.

[102] Tzu-Hsien Yang, Hsiang-Yun Cheng, Chia-Lin Yang, I-Ching Tseng, Han-Wen Hu, Hung-Sheng Chang, and Hsiang-Pang Li. Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 236–249, 2019.

[103] Xiaoxuan Yang, Syrine Belakaria, Biresh Kumar Joardar, Huanrui Yang, Janardhan Rao Doppa, Partha Pratim Pande, Krishnendu Chakrabarty, and Hai Helen Li. Multi-objective optimization of reram crossbars for robust dnn inferencing under stochastic noise. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2021.

[104] Amir Yazdanbakhsh, Ashkan Moradifirouzabadi, Zheng Li, and Mingu Kang. Sparse attention acceleration with synergistic in-memory pruning and on-chip recomputation, 2022.

[105] Shihui Yin, Zhewei Jiang, Jae-Sun Seo, and Mingoo Seok. Xnor-sram: In-memory computing sram macro for binary/ternary deep neural networks. *IEEE Journal of Solid-State Circuits*, 55(6):1733–1743, 2020.

[106] Geng Yuan, Payman Behnam, Yuxuan Cai, Ali Shafiee, Jingyan Fu, Zhiheng Liao, Zhengang Li, Xiaolong Ma, Jieren Deng, Jinhui Wang, Mahdi Bojnordi, Yanzhi Wang, and Caiwen Ding. Tinyadc: Peripheral circuit-aware weight pruning framework for mixed-signal dnn accelerators. In *2021 Design, Automation And Test in Europe Conference And Exhibition (DATE)*, pages 926–931, 2021.

[107] Geng Yuan, Payman Behnam, Zhengang Li, Ali Shafiee, Sheng Lin, Xiaolong Ma, Hang Liu, Xuehai Qian, Mahdi Nazm Bojnordi, Yanzhi Wang, and Caiwen Ding. Forms: Fine-grained polarized reram-based in-situ computation for mixed-signal dnn accelerator. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 265–278, 2021.

[108] Jinshan Yue, Yongpan Liu, Fang Su, Shuangchen Li, Zhe Yuan, Zhibo Wang, Wenyu Sun, Xueqing Li, and Huazhong Yang. Aeris: Area/energy-efficient 1t2r reram based processing-in-memory neural network system-on-a-chip. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ASPDAC '19, page 146–151, New York, NY, USA, 2019. Association for Computing Machinery.

[109] Xiandong Zhao, Ying Wang, Xuyi Cai, Chuanming Liu, and Lei Zhang. Linear symmetric quantization of neural networks for low-precision integer hardware. In *ICLR*, 2020.