CiMLoop: A Flexible, Accurate, and Fast Compute-In-Memory Modeling Tool

Tanner Andrulis
MIT
Cambridge, USA
andrulis@mit.edu

Joel S. Emer
MIT, Nvidia
Cambridge, USA
jsemer@mit.edu

Vivienne Sze
MIT
Cambridge, USA
sze@mit.edu

***Abstract*—Compute-In-Memory (CiM) is a promising solution to accelerate Deep Neural Networks (DNNs) as it can avoid energy-intensive DNN weight movement and use memory arrays to perform low-energy, high-density computations. These benefits have inspired research across the CiM stack, but CiM research often focuses on only one level of the stack (*i.e.,* devices, circuits, architecture, workload, or mapping) or only one design point (*e.g.,* one fabricated chip). There is a need for a full-stack modeling tool to evaluate design decisions in the context of full systems (*e.g.,* see how a circuit impacts system energy) and to perform rapid early-stage exploration of the CiM co-design space.**

To address this need, we propose CiMLoop: an open-source tool to model diverse CiM systems and explore decisions across the CiM stack. CiMLoop introduces (1) a flexible specification that lets users describe, model, and map workloads to both circuits and architecture, (2) an accurate energy model that captures the interaction between DNN operand values, hardware data representations, and analog/digital values propagated by circuits, and (3) a fast statistical model that can explore the design space orders-of-magnitude more quickly than other high-accuracy models.

Using CiMLoop, researchers can evaluate design choices at different levels of the CiM stack, co-design across all levels, fairly compare different implementations, and rapidly explore the design space.

***Index Terms*—Compute-In-Memory, Processing-In-Memory, Analog, Deep Neural Networks, Systems, Hardware, Modeling, Open-Source**

I. Introduction

Compute-In-Memory (CiM) is a promising solution to address the high data movement energy and large number of computations required by Deep Neural Networks (DNNs). CiM systems compute directly inside memory, letting them **(1)** keep DNN weights in memory to reduce high-energy data movement, and **(2)** use large memory arrays to compute many parallel multiply-accumulate (MAC) operations with high density and low energy.

Many recent CiM implementations explore different levels of the CiM stack: *Devices* store weights in CiM arrays; *Circuits* perform computations, analog/digital conversion, data movement, and other actions; *Architecture* organizes devices, circuits, and other components into a larger system; *Workloads* are the DNNs to accelerate; and *Mapping* schedules workloads spatially and temporally on hardware.

Modeling Work	Architecture Flexibility	Circuit Flexibility	Energy Accuracy	Model Speed
NeuroSim [3]–[6]	Low	Low	High	Low
MNSim [7], [8]	Low	Low	Low	Med.
Timeloop [9]–[14]	High	Low	Low	High
This Work	**High**	**High**	**High**	**High**

TABLE I: Comparison to prior CiM modeling works.

Often, researchers explore only one of these levels [1], [2]. However, CiM stack levels interact, so it is important to have a *full-stack model*, which looks at all levels together. Full-stack modeling is essential because *evaluating a choice at one level requires the context of the other levels*. For example, when choosing circuits, we would like to know how they will affect full-system energy/throughput when running a given workload. For this reason, *we must co-optimize across the full stack to find the design that best meets desired criteria.*

To enable full-stack modeling, we need a modeling tool that can represent all levels of the CiM stack and how they interact with each other. To do so, such a tool must be *flexible* to model the space of design decisions, *accurate* to correctly compare these decisions, and *fast* to explore the design space quickly. Table I compares CiMLoop to prior modeling works in addressing the following key challenges.

A. CiM Modeling Challenges

1) Flexibility Challenge: The modeling tool must model the CiM design space, but different CiM implementations introduce different circuits and architectures [15]–[24]. To model the design space, the tool must let users easily define circuits, architectures, and how data moves between them.

This is a challenge because data movement can be different between architectures (*e.g.*, SRAM buffers in the memory hierarchy store and exchange data with each other) and circuits (*e.g.*, within an SRAM buffer, sense amplifiers read and propagate signals from the SRAM array). To address this challenge, the modeling tool must be *flexible*, meaning that it lets users easily describe a wide range of components, how they connect, and how they move data. Unfortunately, prior CiM modeling tools are either inflexible [6], [8], or lack circuit-level modeling [9], [13].

Solving this issue introduces a second challenge: Flexible modeling tools must describe many different types of components, including both an architecture hierarchy (*e.g.*, DRAM

The CiMLoop source, documentation, and tutorials are available at *https://github.com/mit-emze/cimloop*.

+ L3/L2/L1 cache) and set of circuits (*e.g.*, data converters, SRAM bitcells, addressing circuitry). To address this, the modeling tool must also let users easily describe complex designs with many different types of components.

2) Accuracy Challenge: The tool must model energy accurately to correctly inform design decisions, but device and circuit energy is *data-value-dependent*, meaning affected by the values of data that each component propagates (*e.g.*, energy of some ReRAM devices are proportional to computed MAC values [6]) Data-value-dependence can significantly affect overall system energy [17], so we need a modeling tool with high *energy accuracy*, which we define as capturing data-value-dependence. Unfortunately, some prior models may use inaccurate fixed-energy or fixed-power models [8], [9], [13] that do not model data-value-dependence.

3) Modeling Speed Challenge: The modeling tool must be fast enough to explore the large design space, but accurate data-value-dependent energy modeling depends on data values, and it is slow to simulate the many data values propagated by components in a CiM system. Unfortunately, prior CiM modeling works have not addressed this problem; accurate prior models are slow [6] and fast prior models are inaccurate [8]–[14].

B. CiMLoop

To address these challenges, we propose CiMLoop: a full-stack CiM modeling tool with flexible user-defined systems and fast, accurate statistical energy modeling. CiMLoop makes the following key contributions:

- A flexible specification to describe CiM systems with user-defined circuits and architecture. This specification includes **(1)** directives representing circuit and architecture data movement patterns, and **(2)** a representation that describes both circuits and architecture in a single hierarchy. This hierarchy lets users easily define and map workloads to systems with many different types of circuits, architectures, and data movement patterns.
- An accurate model of data-value-dependent component energy that captures the interactions between DNN operand values, data representations, and data values propagated by components. Using this interface, we develop a suite of CiM component models that can easily be used in user-defined systems.
- Fast statistical models to calculate data-value-dependent energy. These models calculate average energy once for each action by each type of component, letting CiMLoop **(1)** use constant runtime to model an arbitrary number of components/actions, and **(2)** amortize energy calculation over many mappings. As a result, CiMLoop is orders-of-magnitude faster than prior accurate modeling works.
- Case studies using four recently-published fabricated CiM designs. In these case studies, we validate CiMLoop's accuracy and show that CiMLoop can **(1)** model diverse CiM implementations, **(2)** explore tradeoffs at different levels of the CiM stack, **(3)** model full systems, and **(4)** fairly compare different CiM designs.

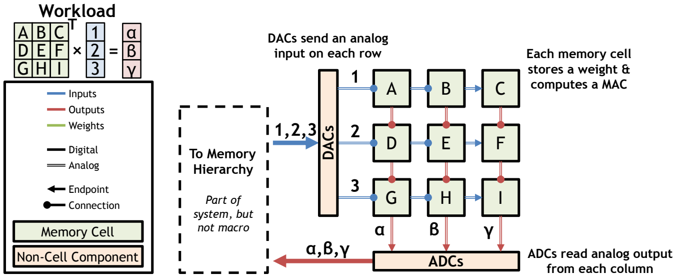


Fig. 1: CiM macro computing a matrix-vector multiplication. From a workload (top-left), the weight matrix (green) is programmed into memory cells. Input vector elements (blue) are sent on rows and outputs (red) appear on columns.

By addressing these challenges, CiMLoop lets researchers evaluate and co-design across levels, fairly compare CiM implementations, and rapidly explore the CiM design space.

II. Background/Motivation

We first give background on *Compute-In-Memory (CiM)* for accelerating DNNs and why full-stack modeling is essential for evaluating and exploring the CiM space. Following this, we discuss each of the three challenges addressed by CiMLoop.

A. Compute-In-Memory (CiM) for DNNs

Tensor operations in Deep Neural Network (DNN) layers can use large input, weight, and output tensors that are energy-intensive to move from memory. Furthermore, tensor operations may require large numbers of multiply-accumulate (MAC) operations, which can also be energy and latency-intensive to compute [25]–[27].

Data movement consumes a significant portion of DNN energy [27], [28], and CiM can reduce energy costs by computing directly within the memory arrays [29], which we define as two-dimensional grids of interconnected memory cells (*e.g.*, SRAM bitcells or RRAM devices). Most commonly, CiM systems keep DNN weights in memory because they do not change during DNN inference [2], [18], [20], [29]. Many CiM implementations further reduce energy and latency with analog domain computation, where an analog MAC operation can be completed with one or a few memory cells rather than a full digital MAC circuit.

Often, a CiM implementation is published as a *macro* [16]–[24], which we define as an array of memory cells plus the additional components needed to compute full MAC operations. This is to contrast from the *system*, which we define as one or more macros plus a memory hierarchy and the interconnects between memories in the hierarchy.

One commonly-explored macro topology [2], [29], [30] is shown in Fig. 1. In this topology, a digital-analog converter (DAC) supplies analog inputs to the rows of a memory array. Each *memory cell* in the array computes an analog MAC operation between the input and its stored weight. Analog outputs from each memory cell in a column are summed and read by an analog-digital-converter (ADC) to yield a digital output.

Researchers have explored a wide variety of other CiM macro topologies, circuits, and means of analog and/or digital computation. CiMLoop models the CiM macro space and integrates CiM macros with system modeling tools [9]–[14].

B. Full-Stack Modeling

CiM research spans each of the following levels:

- *Devices*: The components forming each memory cell. Published macros often use SRAM [17], [20], DRAM [31], [32], ReRAM [18], [30], [33], or STT-RAM [34].
- *Circuits*: The components performing computation, analog/digital conversion, storage, data movement, and other actions [17], [20], [35].
- *Architecture*: The organization of components into a larger system (*e.g.*, the number of each component and how components are connected) [2], [29], [36]–[40].
- *Workload*: The DNN to be run, which we model as a series of extended-Einsum [41] operations with tensors of varying shapes and values [13], [26].
- *Mapping*: The temporal and spatial scheduling of the workload onto the system [13].

Full-stack modeling, meaning modeling all levels together, is essential for two reasons. First, *all-level context is needed to evaluate choices at a given level.* The pitfall of modeling without considering the full stack is shown in Fig. 2a where we explore different CiM array sizes for a macro running the DNN ResNet18 [42]. The **lowest-energy macro** has a smaller array, which maintains high utilization and low energy even with small DNN tensors. The **macro that yields the lowest-energy system** uses a larger array. Though the large array is often underutilized, it stores store more weights, letting it reduce data movement to/from the memory hierarchy to reduce system energy.

If we had optimized for macro energy, we would have been misled into a higher-energy system; only by considering the full system can we make the best decisions for a level. [1]

The second reason for full-stack modeling is that *co-exploring levels can find better systems* [43]–[46]. In Fig. 2b, we start with the lowest-energy macro from Fig. 2a and measure full-system energy while varying the DAC resolution (circuits) and CiM array size (architecture). The **optimize circuits** macro used a high-resolution DAC to process more input bits at a time, reducing the number of times the array had to be activated and decreasing energy per MAC. The **optimize architecture** macro, in addition to the high-resolution DAC, used a larger array to further reduce the number of times the array had to be activated for a given number of MACs. For smaller tensors, the array was underutilized, and high-resolution high-energy DACs resulted in a greater energy efficiency loss. Finally, the **optimize both** macro used a larger array and a low-resolution DAC. The large array could compute many MACs per activation, while the low-resolution DAC maintained low energy when the large array was underutilized. Here, co-optimization helped us find design decisions that synergized to produce a lower-energy system.

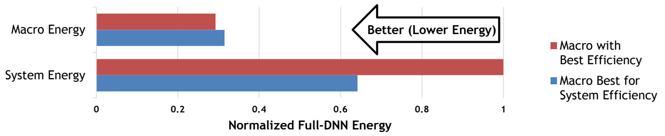


Fig. 2a: Optimizing for the lowest-energy macro while neglecting the system yields a higher-energy system overall.

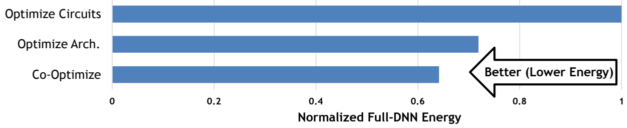


Fig. 2b: Co-optimizing circuits and architecture yields a lower-energy system than optimizing either individually.

C. CiM Circuits, Architectures, and Data Movement

A CiM macro consists of a set of components (*i.e.*, devices, circuits, and architecture) and the connections they use to move data. Component and data movement choices determine system area/energy and the computations that a system may carry out, so it is critical to represent these choices when modeling CiM macros.

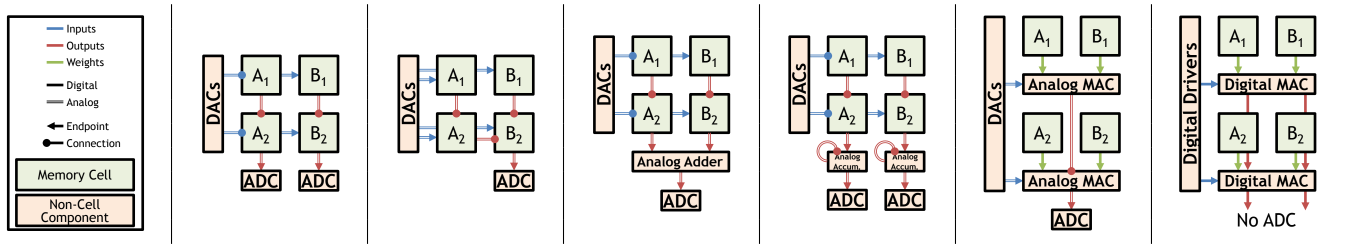
Different CiM macros use different components and data movement patterns to optimize for different design goals. A key aspect to consider when choosing components and data movement patterns is how they can leverage *data reuse* to reduce energy [28]. Reuse saves energy by reducing the number of times components are used to move and/or process data. For example, weights may be stored and reused temporally (*i.e.*, across cycles) in memory cells to reduce the number of times weights are fetched from separate memories. As another example, the macro in Fig. 1 spatially reuses analog inputs between multiple array columns, letting it reuse each analog input for multiple computations. This reduces the number of times that DACs convert inputs to analog.

There are many interesting design choices involving different components, data movement patterns, and reuse opportunities. To illustrate some of such choices, Fig. 3 shows several strategies that published macros use to reduce ADC energy. In the *Base Macro* [15], multiple array rows reuse outputs (*i.e.*, an N-row array may sum outputs from up to N MACs and read the result with one ADC convert)[2]. To further reduce ADC energy, different macros use a variety of strategies:

- *Macro A* [16] reuses analog outputs across different columns by summing them on wires.
- *Macro B* [17] reuses analog outputs across different columns by summing them with an analog adder.
- *Macro C* [18] reuses outputs across different cycles by accumulating them with an analog accumulator.

[1] Furthermore, to conduct this exploration, we had to consider the workload (tensor sizes for each DNN layer) and mapping (maximizing array utilization).

[2] Many works have introduced strategies to reduce ADC energy. For more information, see the Titanium Law [38], which breaks down the factors that contribute to ADC energy and shows how ADC energy can be reduced.



	Base Macro [15]	***A*** **[16]**	***B*** **[17]**	***C*** **[18], [19]**	***D*** **[20], [21]**	***Digital CiM*** **[22]**
From where outputs are reused (summed)	Different rows	Different rows and **columns**	Different rows and **columns**	Different rows and **cycles**	Different rows and **columns**	Different rows
How outputs are reused (summed)	Wire	Wire	Wire and **analog adder**	Wire and **analog accumulator**	Wire and **analog MAC**	**Digital MAC**
Analog or digital	Analog	Analog	Analog	Analog	Analog	**Digital**
Benefit of strategy	-	↑ Output reuse, ↓ ADC converts				Eliminate ADC
Cost of strategy	-	↓ Input reuse, ↑ DAC converts	Area and energy of extra components.			
Mapping restriction of strategy	-	A_i/B_i store bits of different weights	A_i/B_i store different bits of same weight	Different input bits in consecutive cycles	A_i/B_i store different bits of same weight	A_i/B_i store different bits of same weight

Fig. 3: ADC-energy-reducing strategies of published CiM macros [15]–[22]. Bolded items indicate changes from the base macro to implement strategies. In addition to the listed mapping restrictions, all macros are restricted to store different weights in A_1/A_2 and different weights in B_1/B_2. A flexible model is needed to explore the different components and data movement patterns that are unique to each macro. Open-source models of these macros are available at *https://github.com/mit-emze/cimloop*.

- *Macro D* [20] uses an analog MAC unit that internally reuses outputs, letting it generate a single output using different weight bits.
- *Digital CiM* [22] reuses outputs digitally to avoid the need for an ADC.

Each of these strategies has tradeoffs. Additional components consume area and energy. Components may also introduce mapping and data movement constraints (*e.g.*, for *Macro B*, adjacent columns must store different bits of same weight, and for *Macro A*, adjacent columns must store bits from different weights).

In the broad CiM design space, these are just a few sets of possible components, and reducing ADC energy is just one challenge to be addressed.

Key modeling challenge 1: A modeling tool must be *flexible*, meaning able to describe and map workloads to macros with different sets of components and different patterns of data movement/reuse. Furthermore, choices interact with the full system, which may include buffer hierarchies and interconnects. Full-system modeling necessitates a representation that makes it easy to describe complex designs with many different types of components.

To model the design space, it is critical to let users define the components in the system and how they connect. Unfortunately, prior modeling tools do not have this ability. NeuroSim [6] and MNSim [8] model only the *Base Macro* and do not let users add components or define data movement patterns. Timeloop+Accelergy [9], [10], [13] let users define architecture-level components (*e.g.*, SRAM buffers) but are not able to model circuit data movement or reuse (*e.g.*, memory cells and sense amplifiers in an SRAM buffer). For this reason, none of these works can represent $Macros\ A$, B, C, or D.

D. Data-Value-Dependent Energy

Device/circuit energy is *data-value-dependent*, meaning affected by data values propagated by components in the system. We can break data-value-dependence into three components:

1) **DNN Workload:** The system processes DNN operands, which vary between DNN layers/tensors in their distributions, signedness, and amount of sparsity [11], [38].
2) **Representation:** How data values appear is determined by how the hardware represents operands [47]. First, operands are *encoded*, meaning represented as bits. Next, they are *sliced*, meaning bits are partitioned across different devices and circuits. Representations may change for different tensors and different components in the system.
3) **Circuits:** Different circuits spend different amounts of energy to propagate different data values. For example, some ADCs [35], [48] and DACs [17], [20] spend less energy to convert small values.

Fig. 4 shows examples of each of these three components affecting DAC energy. Data-value-dependence can have a 2.5× effect on the energy of the DAC type shown.

Key Modeling Challenge 2: To model accurately, a modeling tool must model the data-value-dependent interactions between workload, data representations, and circuit energy. To explore the space of interactions possible, the tool must do this flexibly (*i.e.*, for heterogeneous user-defined circuits).

Unfortunately, Timeloop+Accelergy and MNSim use inaccurate fixed-energy [9], [13] or fixed-power [8] models that do not model data-value-dependence.

E. The Need for Fast Modeling

Co-exploring multiple levels opens a large design space, and the number of design points grows exponentially with the number of decisions explored. Furthermore, for each point,

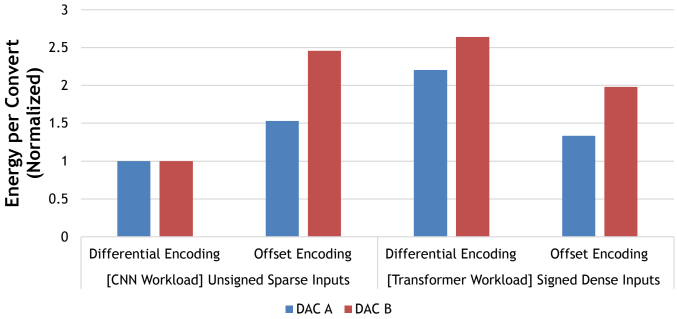


Fig. 4: Data-value-dependence can affect circuit energy by $> 2.5\times$, and its effect is different for each DAC, encoding, and layer. The best encoding is different for each layer.

we may map a DNN with hundreds of layers [49] and search thousands of mappings for each layer [13].

Key Modeling Challenge 3: Fast modeling is essential to explore the large design space. This goal is in tension, however, with accurate modeling because accurate modeling requires modeling the many data values that circuits propagate.

Unfortunately, this tension has led prior work to be either accurate or fast, but not both. NeuroSim [6] uses an accurate data-value-dependent model, but it simulates every data value and is therefore slow. Timeloop+Accelergy [9], [13] and MNSim [8] use faster fixed-energy and fixed-power models, but these models do not capture data-value-dependence.

III. CiMLoop

In this section, we describe CiMLoop and how it addresses the three key challenges described previously. We begin with an overview of the CiMLoop infrastructure. We then describe CiMLoop's flexible per-component data reuse models and CiMLoop's system representation. Following that, we describe CiMLoop's data-value-dependent energy model and provided component models. Finally, we discuss why CiMLoop is fast.

A. Infrastructure Overview

CiMLoop is built on the Timeloop+Accelergy [9], [13] infrastructure. Users describe systems with a collection of YAML files (architecture, workload, components, other configurations) and run the infrastructure with a Python interface. Accelergy performs area/energy estimations for each component in the system. Timeloop uses these estimates to perform mapping (*i.e.*, spatial & temporal tiling) searches and model full-system energy, throughput, and area for systems running DNN workloads. CiMLoop modifies both Accelergy and Timeloop to support CiM features, introducing a component modeling interface that supports data-value-dependent modeling and circuit-level data movement/reuse/mapping support. CiMLoop also introduces a new flexible architecture specification and fast modeling pipeline.

B. Flexible Circuit and Architecture Modeling

CiMLoop's flexible specification lets users specify components, where they are in the system, and how they can move and reuse data. We define *components* as anything that may move or reuse data. Components may be fine-grained (*e.g.*, an SRAM bitcell) or coarse-grained (*e.g.*, an SRAM buffer).

1) Per-Component Data Movement and Reuse: In a data movement hierarchy, component X *reuses* a piece of data if X uses the data multiple times with one access to the parent of X. Reuse can only occur if reuse is supported by hardware, present in the workload (*i.e.*, multiple computations use the same data), and present in the mapping (*i.e.*, computations are mapped to adjacent spatial components or timesteps such that reuse can happen between the components or timesteps).

For each component and each tensor (*i.e.*, inputs, outputs, weights), users may set supported reuse independently. *Temporal-Reuse* can store data between cycles, *e.g.*, buffers temporally reuse data. *No-Temporal-Reuse* does not allow reuse between cycles. No-temporal-reuse components may or may not *coalesce*, meaning change multiple accesses of the same value into one access of backing storage. For example, when an adder sums several values, it coalesces them into one output (*i.e.*, reusing the output for multiple additions). On the other hand, a DAC may not coalesce; if the same piece of data is propagated through a DAC multiple times, it must be fetched from backing storage multiple times. Temporal-reuse components can always coalesce if given the opportunity. Additionally, data may *bypass* data around a component without activating that component (*e.g.*, inputs bypass a weight buffer).

Spatially between components, data may be reused (multicast or reduced between components) or not reused (unicast to each component individually). For example, in the *Base* macro topology, outputs are reused between rows (*i.e.*, outputs from multiple rows are summed and read once by the ADC) but not columns (*i.e.*, outputs from each column are read individually by the ADC). We show each of these reuse options when describing the macro in the following section.

2) Representing Systems with Container-Hierarchy: CiMLoop uses a container-hierarchy representation to scalably represent circuits and architectures. A *container* is a grouping of components and/or (sub)containers, and a *container-hierarchy* is a series of containers where each contains all subsequent components/containers. Container-hierarchies are useful for describing CiM systems because:

- Each container isolates local design decisions while abstracting other containers. This permits easy description of complex designs with many different types of components.
- Container hierarchies express circuits and architecture in one hierarchy, so they are compatible with tools and abstractions for memory hierarchies [13].
- Container-hierarchies can be nested to arbitrary depths (*e.g.*, can describe memory cells, circuits, architecture, and multi-chip data movement at the same time).
- Multiple container-hierarchies can be mixed and matched to aid design space exploration (*e.g.*, a user may create one macro and test that macro in multiple systems).

To show how to use container-hierarchy, we use it to describe the CiM system in Fig. 5a. This system is similar to the macro shown in Fig. 1, but we add a digital adder to reuse (sum) outputs after the ADC and we use a single buffer as the memory hierarchy. In our description, each component is underlined and each **reuse option is bolded**. The macro can be described as:

1) A container abstracts the macro, separating it from the rest of the system. The macro communicates with a buffer that **temporally reuses** inputs and outputs (*i.e.*, stores them over time).
2) In the macro, inputs pass through a **no-temporal-reuse-no-coalesce** bank of DACs to be converted to analog. Meanwhile, a **no-temporal-reuse** adder sums values from columns and **coalesces** them into one output. Finally, we abstract the array as two column containers. Inputs, but not outputs, are **reused spatially** between columns (*i.e.*, all columns receive the same inputs, but each column produces an independent output).
3) In each column, two memory cells **temporally reuse** (*i.e.*, store over time) weights. Outputs from memory cells are **reused (summed) spatially** then pass through a **no-temporal-reuse-no-coalesce** ADC to be converted to digital.

Fig. 5b shows a simplified YAML specification for this CiM macro. Given this specification and a workload, CiMLoop maps (*i.e.*, spatially/temporally schedules) the workload onto the system and generates area/energy/throughput estimates. In the full (non-simplified) specification, each component includes a class (*e.g.*, a buffer is SRAM), attributes (*e.g.*, ADC resolution), and optional constraints/heuristics for the mapping search. CiMLoop tutorials include the full syntax.

C. Accurate Data-Value-Dependent Modeling

1) Data-Value-Dependent Pipeline: To model data-value-dependent energy, CiMLoop needs to know what data values each component propagates and how these impact energy. This procedure is broken into three steps:

a) Workload Operand Distributions: For each workload tensor (*i.e.*, inputs, outputs, weights), CiMLoop uses a distribution of values that operands in this tensor take. Distributions, rather than full tensors, are fast to model as described in Section III-D.

b) Encoding and Slicing: CiMLoop calculates the representation of elements in each tensor for each component (representations may change as data moves through the system). Representation first depends on *encoding*, meaning representing operands as binary values. CiMLoop supports encoding and slicing functions from CiM implementations, including offset [2], differential [38], XNOR [16], and magnitude-only [44]. Other encoding schemes can be defined or imported. After encoding, binary values are *sliced*, meaning their bits are partitioned across hardware components. Computations across multiple slices are exposed to the Timeloop [13] mapper, letting CiMLoop tile and spatially/temporally map the bits of each tensor.

c) Component Energy Modeling: At this point, CiMLoop has the distribution of encoded and sliced data values propagated by each component. Per-component models use these distributions to calculate energy. Each component may use distributions differently (*e.g.*, resistor energy increases with the duration of applied voltages, while capacitor energy increases with the amount of switching of applied voltages).

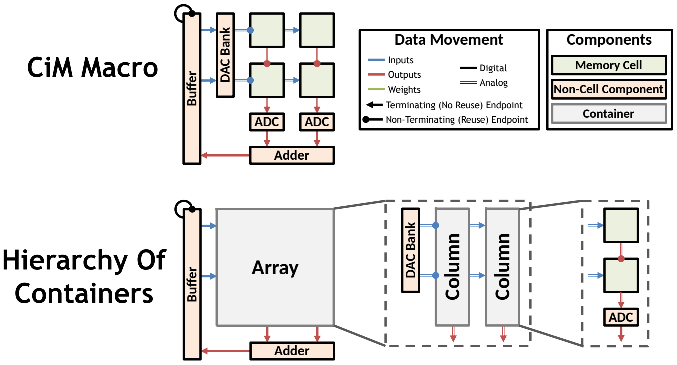


Fig. 5a: Container-hierarchy partitions systems into simple containers.

```
!Component   # Buffer stores inputs & outputs.
name: buffer
temporal_reuse: [Inputs, Outputs] # Bypass weights

!Container  # Container includes everything declared in
name: macro # following lines

# =================== Now inside Macro ====================
!Component  # Adder sums values and coalesces them into
name: adder # one output.
coalesce: [Outputs] # Bypasses inputs/weights

!Component # Inputs pass through DACs, convert to analog.
name: DAC_bank # DACs can not coalesce.
no_coalesce: [Inputs] # Bypass outputs/weights

!Container   # Inputs are spatially reused between columns,
name: column # while outputs/weights are not.
spatial: {meshX: 2} # 2 columns in X dimension
spatial_reuse: [Inputs] # Reuse inputs, not outputs/weights

# =================== Now inside Column ===================
!Component # Outputs pass through ADC, convert to digital
name: ADC
no_coalesce: [Outputs] # Bypass inputs/weights

!Component # Memory cells store & temporally reuse weights.
name: memory_cell # Memory cells spatially reuse outputs.
spatial: {meshY: 2} # 2 cells in Y dimension
temporal_reuse: [Weights] # Bypass inputs/outputs
spatial_reuse: [Outputs] # Reuse outputs not inputs/weights
```

Fig. 5b: A YAML container-hierarchy describes the system in Fig. 5a. #-marked comments explain the syntax. `!Component` / `!Container` tags declare components/containers, `temporal_reuse` / `coalesce` / `no_coalesce` directives describe data reuse, and `spatial` describes the number of each component in the X/Y dimensions. If a particular tensor is not listed for a component, then it bypasses the component.

2) Provided Models: CiMLoop includes a suite of provided plug-ins that let users model components in their own systems. CiMLoop uses the Accelergy [9] plug-in suite, including the CACTI [50] plug-in modeling buffers and the Aladdin [51] plug-in modeling digital components. CiMLoop also includes a simple plug-in interface that lets users define new data-value-dependent energy models. To model CiM components, CiMLoop includes the following additional plug-ins.

The ADC Plug-In [52] uses regression-based models over published ADCs [53]–[56] to predict the area and energy of an ADC (or bank of ADCs) that meets a user-defined throughput, resolution, and number of ADCs.

The NeuroSim Plug-In uses NeuroSim [6] to model array row/column drivers and ADCs; memory cells; and digital components such as adders, multiplexers, and logic gates. To enable flexibility, CiMLoop separates NeuroSim components from one another so they can be reassembled into user-defined systems. To enable fast modeling, CiMLoop connects the NeuroSim plug-in to the fast modeling pipeline described in Section III-D. These changes maintain high accuracy, which we quantify in Section IV-A. We also connect the NeuroSim plug-in to memory cells in the NVMExplorer [1] memory cell exploration tool to let users flexibly swap device models.

The Library Plug-In models a library of components used in various CiM works [2], [6], [18], [29], [37], [38], [40], [44], [51], [57]. The Library plug-in can be used to quickly create new systems by leveraging off-the-shelf component models [58], or it can be used to fairly compare different architectures while using a common set of components.

D. Fast Modeling

To model quickly, CiMLoop calculates the average energy for each action by each type of component. This average energy can then be applied to any number of actions by that type of component, letting CiMLoop model an arbitrary number of components/operations with constant runtime. This is particularly helpful for quickly modeling CiM systems, which may have many components in parallel (*e.g.*, thousands of memory cells in an array).

1) Modeling DNN Operand Values with Distributions: CiMLoop leverages the distributions (*i.e.*, a probability mass function for each tensor) of DNN operand values [38] as an input to its statistical model. As described in Section III-C, CiMLoop will use these distributions and their hardware representations to derive the data values that each type of component will propagate.

CiMLoop decouples the gathering of DNN operand distributions from system modeling. This is for two reasons.

1) It allows CiMLoop design space exploration to be much faster, as the DNN does not need to be run to evaluate each CiM design.
2) It permits multi-fidelity modeling by letting users trade off the fidelity of distributions with the amount of user effort required to obtain distributions. Users may provide CiMLoop with distributions that are **easy-to-obtain, yet low-fidelity** (*e.g.*, model ReRAM device input voltage as a uniform distribution from 0V to 1V); **moderately-easy-to-obtain and moderate-fidelity** (*e.g.*, two's-complement encode a DNN input tensor and use the resulting distribution as ReRAM device input voltages); or **difficult-to-obtain, yet high-fidelity** (*e.g.*, simulate a DAC generating analog voltages and use the resulting distribution as ReRAM device input voltages).

CiMLoop assumes that the distributions of values in separate tensors are independent. Leveraging this assumption, CiMLoop stores an independent distribution for each tensor. Independent distributions, rather than joint distributions, are faster to record and use in energy estimations because for N-point probability density functions and T tensors, the number of points that must be stored scales with $O(NT)$ for independent distributions and $O(N^T)$ for joint distributions. We note that this restriction is not fundamental to CiMLoop, and user-defined models may use joint distributions. However, using a joint distribution will make CiMLoop slower, and we found that independent distributions are sufficient to get high accuracy (see Section III-C).

2) Per-Layer Model: For each DNN layer, plug-in models (described in Section III-C) for each component receive distributions of data values (one distribution for each tensor) and calculate average per-action energy. Different layers require different per-action energies because data value distributions can change between layers and tensors within layers [38].

3) Mapping-Invariant Energy: CiMLoop assumes that the energy of each action by each component is mapping-invariant (*i.e.*, it does not change across different mappings). Note that overall component energy is not mapping-invariant, as the number of actions by each component depends on the mapping (*i.e.*, the energy per read of a buffer must be the same across mappings, but the number of reads, and thus overall energy, may be different).

This assumption is valid if the mapping does not affect the distribution of values propagated for any particular tensor. Generally, for regular mappings (*i.e.*, those that can be represented by a loop nest [13], [14]), this is the case because mappings affect tensor elements equally (*e.g.*, if a mapping results in DACs propagating inputs twice as often, then each input element is propagated twice as often and the distribution does not change). This assumption is violated for sparse systems that may skip zero elements [11], [12], though CiMLoop focuses on dense CiM system modeling.

Based on this assumption, CiMLoop pre-calculates the average energy for each action by each component, amortizing calculation time over many mappings.

E. Example Data-Value-Dependent Calculation

As an example, we will calculate the data-value-dependent energy consumed by ReRAM device reads whose energy E is the product of the weight conductance G, the square of the applied input voltage V, and the read duration T_{read}.

Algorithm 1 shows the calculation as it would be performed in a design space exploration over architectures. For each DNN layer (line 2), we profile the DNN to get probability mass functions for input and weight values (line 3). Then, for each architecture in the exploration (line 4), lines 5 and 6 calculate the average squared voltage and average conductance that the architecture will use to represent inputs and weights, respectively. This uses the architecture's slicing and encoding functions as defined in Section II-D (*e.g.*, an architecture may encode an operand as a two's complement value and represent this value as a voltage between 0V and 1V). For simplicity, we do not show slicing (*i.e.*, partitioning bits across multiple components and/or timesteps). With slicing, then Algorithm 1 would find the average power across all slices.

Finally, we calculate average ReRAM read energy in line 7. Next, lines 8 and 9 iterate over different mappings and calculate the number of times each mapping reads ReRAM devices. Line 10 multiplies the number of reads by the average read energy to get total ReRAM read energy.

Amortization of data-value-dependent calculation time is key to CiMLoop's speed. Calculating data-value-dependent energy and profiling DNNs consume negligible overhead because data-value-dependent calculations (lines 5-7) are amortized over the innermost loop (line 8), which may run for thousands of mappings, while profiling the DNN (line 3) is amortized over both the innermost (line 4) and intermediate (line 8) loops. The per-mapping evaluation (line 9), is non-data-value-dependent (based on the assumption in Secton III-D3) and dominates runtime in most explorations.

Algorithm 1: Calculate Data-Value-Dependent ReRAM energy in a design space exploration. Energy GV^2T_{Read} is the product of conductance, squared voltage, and read duration. $P_I(x)$ and $P_W(y)$ are the probability mass functions of inputs and weights. $V_I(Arch, x)$ and $G_W(Arch, y)$ encode input and weight values as voltages and conductances, respectively.

```
1  Func CalcReRAMEnergy (Architectures, Layers)
       /* Requires a set of architectures to
          explore and a set of DNN layers to
          run                                        */
2      for $Layer \in DNN\ Layers$ do
           /* Run the layer and record
              input/weight distributions             */
3          $\mathrm{P}_I(x), P_W(x) = RecordOperandPMFs(Layer)$
4          for $Arch \in Architectures$ do
               /* Calculate average data-value-
                  dependent ReRAM read energy        */
5              $\mathrm{V}_{Avg}^2 = \sum_x P_I(x) \times V_I(Arch, x)^2$
6              $\mathrm{G}_{Avg} = \sum_y P_W(y) \times G_W(Arch, y)$
7              $\mathrm{E}_{Avg} = V_{Avg}^2 \times G_{Avg} \times T_{read}$
8              for $Mapping \in GetMappings(Arch, Layer)$
                do
                   /* Mapping evaluation
                      dominates runtime                  */
9                  #Reads = $Evaluate(Mapping,\ Arch)$
10                 Mapping.$\mathrm{E}_{ReRAM} = E_{Avg} \times \#Reads$
```

IV. Accuracy and Model Speed Evaluation

In this section, we evaluate CiMLoop accuracy and speed relative to prior CiM modeling works. We will use NeuroSim [59] as a baseline. Note that CiMLoop uses the NeuroSim plug-in, so any accuracy or speed differences are due to CiMLoop's data-value-dependent and fast modeling pipelines. We test using ResNet18 [42] and ImageNet [60]. Other NeuroSim-provided models use the CIFAR-10 [61] dataset which is less applicable to modern large-scale DNNs.

A. CiMLoop Accuracy

To evaluate the accuracy of CiMLoop's statistical data-value-dependent model, we model the NeuroSim [6] macro with default parameters. NeuroSim, which calculates every data value propagated by every modeled component, is used as a ground truth. We compare the following:

- **CiMLoop** uses an input, output, and weight distribution for each DNN layer as described in Section III-C.
- **Non-Data-Value-Dependent** is a fixed-energy model [9]. We optimistically assume that energy is calculated using data values averaged over all layers. In general, the fixed-energy model would not incorporate any knowledge of the DNN.

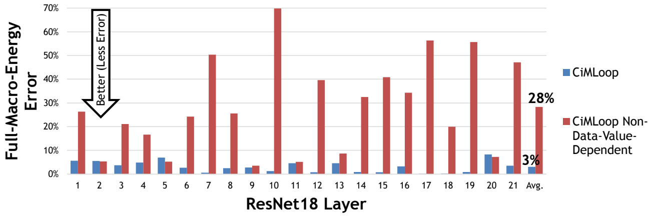


Fig. 6: CiMLoop's data-value-dependent model is significantly more accurate than non-data-value-dependent models.

Fig. 6 compares the accuracy of these two setups when measuring full-macro energy across different layers of ResNet18 [42]. Relative to the NeuroSim ground-truth estimates, CiMloop's data-value-dependent model achieves an average/max error of 3%/7% while the fixed-energy model has an average/max error of 28%/70%. The fixed-energy model has high error because it does not model the distributions of inputs, outputs, and weights, which vary across DNN layers and significantly affect analog component energy. CiMLoop's data-value-dependent model accounts for these distributions, reducing error. CiMLoop's remaining error is due to statistical model error from representing the values of tensors in each DNN layer using independent distributions.

B. CiMLoop Speed

We compare the modeling speed of CiMLoop and NeuroSim when running the default NeuroSim macro running ResNet18/ImageNet [42], [60] one input image on an Intel Xeon Gold 6444Y processor. For CiMLoop we show both single-core and 16-core performance running one or 5000 mappings. NeuroSim does not support multithreading or mapping exploration at the time of testing, so it is tested with one core and one mapping.

Table II shows that CiMLoop improves modeling speed over NeuroSim by several orders of magnitude. Time per mapping decreases for many mappings because CiMLoop can amortize one-time startup costs of library invocation, intermediate file generation, and data-value-dependent energy modeling. In most explorations, we test thousands of mappings per system and startup consumes negligible runtime.

CiMLoop's speed benefits would increase with larger systems and/or larger DNN workload tensors because NeuroSim would simulate the additional components and/or operations, while the runtime of CiMLoop's statistical model would not increase (see Section III-D). CiMLoop also optimizes

Model	**# Cores**	**(Mappings×Layers) / Second**	
		1 Mapping	5000 Mappings
NeuroSim [6]	1	0.07	-
CiMLoop	1	0.28	83
CiMLoop	16	2.25	1076

TABLE II: CiMLoop is orders-of-magnitude faster than prior accurate modeling works. CiMLoop is faster for more mappings because it amortizes mapping-invariant calculation and up-front invocation time.

Macro	Node (nm)	Device	Input Bits	Weight Bits	Array Rows×Cols	ADC Bits
A [16]	65	SRAM	1-8	1-8	768×768	8
B [17]	7	SRAM	4	4	64×64	4
C [18], [19]	130	ReRAM	1-8	Analog	256×256	1-10
D [20], [21]	22	SRAM	8	8	512×128*	8

*Activates a 64×128 subset of the array at once.

TABLE III: Parameterized attributes of *Macros A-D*.

Timeloop+Accelergy [9], [13] modeling speed, making CiMLoop the fastest modeling tool in Table I.

V. Case Studies: Published CiM Macros

To validate CiMLoop and provide examples of some of the decisions that CiMLoop can explore, we present models of four recently-published fabricated CiM macros [16]–[21]. Each macro includes multiple contributions, but due to space limitations, we only discuss those necessary to validate CiMLoop's results. We encourage the reader to see the authors' publications and our open-sourced models for each macro.

For each macro, we model the array and row/column drivers using the NeuroSim [6] plug-in. We create memory cell models and calibrate the area/energy of each component to match published values. Unless otherwise stated, all dataflows are weight-stationary with weights pre-loaded into the macro's CiM array, inputs sent to the CiM array, and outputs read from the CiM array. Table III shows parameterized attributes of each of the CiM macros. CiMLoop can model systems with varied technology nodes, bit precisions, and device technologies. For inputs to DNN workloads, we use ImageNet [60] inputs and Wikipedia [62] for image and language models, respectively.

A. Validating CiMLoop

In this section, we validate CiMLoop models against published *Macros A-D*. We compare CiMLoop-modeled results with the published data for each macro, which include both simulated and silicon-measured results. For each of the following plots, we include macros for which we could find published data (*e.g.*, the papers for *Macros A/B/D*, but not *C*, have voltage sweep results). For ease of visualization, plots show a representative subset of validation data. We report average percent error measurements using all validation data, including data that we do not show in plots.

1) Energy/Throughput and Supply Voltage: Fig. 7 validates CiMLoop modeling *Macros A/B/D* energy and throughput across varying supply voltages. *Macro B* energy is data-value-dependent, so we show results for small and large data values. For supply voltage sweeps, CiMLoop's average energy efficiency and throughput errors are 7% and 2%.

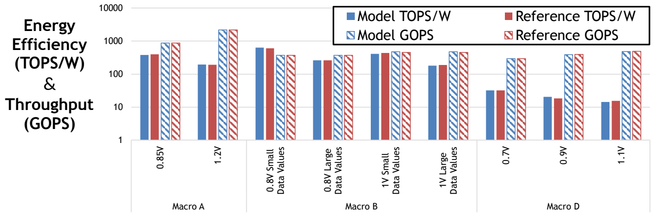


Fig. 7: Validating energy/throughput for varied supply voltage.

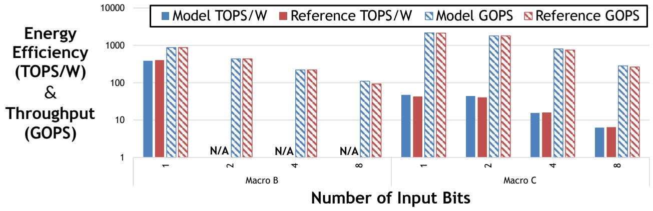


Fig. 8: Validating energy/throughput for varied # of input bits.

2) Energy/Throughput and Input Encoding: Fig. 8 validates CiMLoop modeling *Macros B/C* energy and throughput for varying numbers of input bits. For these sweeps, CiMLoop's energy efficiency and throughput errors are 6% and 5%.

3) Energy Breakdown: Fig. 9 validates CiMLoop modeling the energy breakdowns of *Macros C and D*. For *Macro C*, we report energy for 1b, 2b, and 8b inputs to show that CiMLoop can capture how the energy of each component scales with the number of input bits. Modeled *Macro D* miscellaneous energy is lower than the reference due to components we did not model; accuracy could be improved by adding more components into our model (recall that models are user-defined and components can be added easily). For discrete components, CiMLoop's average energy error is 4%.

4) Area Breakdown: Fig. 10 validates CiMLoop modeling the area breakdowns of *Macros A/B/C/D*. As with the energy breakdown, modeled *Macro D* miscellaneous area could be made more accurate by modeling miscellaneous components. For discrete components, CiMLoop's average area error is 8%.

5) Data-Value-Dependent Energy: Fig. 11 validates CiMLoop modeling the data-value-dependent energy of *Macro B*. As average MAC value increases, *Macro B*'s DAC switches more often to supply larger input values and its analog adder

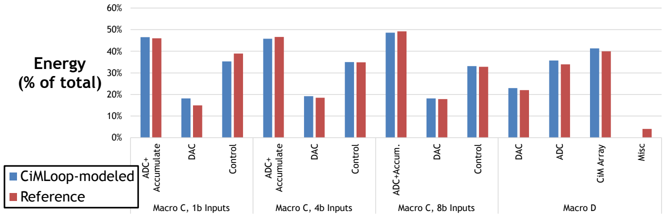


Fig. 9: Validating CiMLoop-modeled energy breakdown.

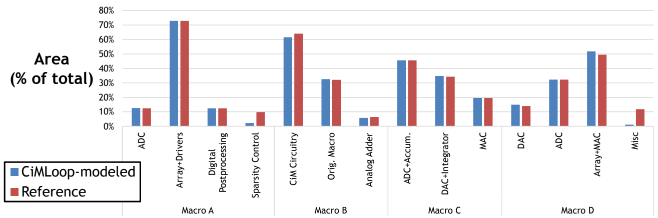


Fig. 10: Validating CiMLoop-modeled area breakdown.

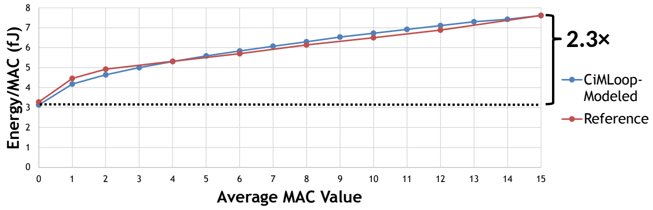


Fig. 11: Validating CiMLoop data-value-dependent energy.

charges/discharges with larger analog values. These data-value-dependent effects can increase macro energy by 2.3×.

B. Exploring with CiMLoop

In this section, we show the usefulness of CiMLoop. We explore one level of the CiM stack using each of *Macros A-C* and model a full system using *Macro D*. Finally, we perform a cross-macro comparison.

1) Mapping: Macro A [16] reuses outputs, rather than inputs, between adjacent CiM array columns (Shown in Fig. 3). Reusing outputs, and not inputs, between every N array columns increases output reuse $N\times$ but decreases input reuse $N\times$. This affects DAC/ADC energy; more output reuse can decrease ADC converts and ADC energy, but trading off input reuse increases DAC converts and DAC energy. This decision also changes available mappings; if adjacent array columns reuse outputs, then we must map workload operations to share outputs between those columns (*i.e.*, map one output channel to those columns). Otherwise, the array is underutilized.

In Fig. 12 we explore *Macro A* configurations that reuse outputs between different numbers of columns. We test configurations when running a maximum-utilization (matrix-vector multiply with dimensions matching the array) and variable-utilization (ResNet18 [42]) workloads. We report DAC, ADC, and other energy. For the maximum-utilization workload, increasing output reuse reduces lower ADC energy but trades off higher DAC energy. For the variable-utilization workload, the three-column-reuse configuration is uniquely lower energy than the other configurations. This is because ResNet18 [42] uses many 3×3 convolutional kernels, and they were able to achieve high-utilization mappings on the three-column-reuse macro. This is one of the reasons why Jia et al. use three-column-reuse in their fabricated chip [16].

2) Circuits: Macro B [17] uses an analog adder circuit that sums analog outputs. The ADC reads the resulting sum, rather than each output individually, so the adder can decrease the number of times the ADC is used. This adder comes with the constraint that its inputs must come from different bits of same weights. In Fig. 13 we explore different widths of analog adders (1-, 2-, 4-, and 8-operand) and workloads with different numbers of bits per weight. Adders that sum more analog operands can increase throughput-per-area because they reduce the number of ADCs required to read outputs. However, they become underutilized when there are fewer bits per weight. Larger adders also consume more chip area; for this reason, the macro with the 8-operand adder never has the highest throughput-per-area.

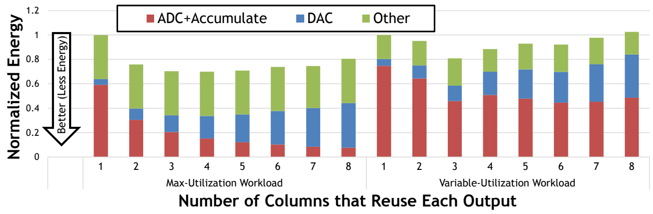


Fig. 12: ***Macro A* + Mapping**: Output reuse between columns decreases ADC energy but trades off lower input reuse and higher DAC energy. The three-column-reuse case had the best-utilization mappings for the variable-utilization workload.

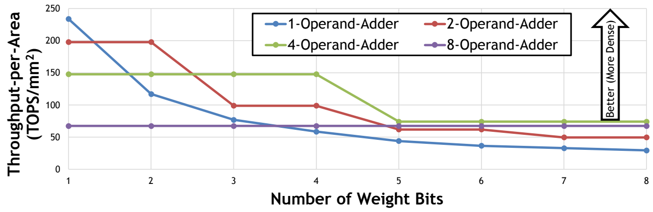


Fig. 13: ***Macro B* + Circuits**: An analog adder trades off flexibility for compute density. Fewer-column adders can flexibly leverage fewer-bit weights, but more-column adders achieve higher compute density with more-bit weights.

3) Architecture: Using *Macro C* [18], [19], we conduct an architectural exploration over varying array sizes. We set the number of array rows and columns to 64, 128, 256, 512, or 1024 and test maximum-utilization (matrix-vector multiplication), large-tensor-size (Vision Transformer ViT [63]), medium-tensor-size (ResNet18 [42]), and small-tensor-size (MobileNetV3 [64]) workloads.

Fig. 14 shows that as array size increases, energy decreases due to additional MACs amortizing ADC and digital output sum energy. These effects are strongest for the maximum-utilization and large-tensor-size workloads. For the medium-tensor-size workload, effects saturate as the array grows larger and becomes underutilized for smaller layers. For the small-tensor-size workload, underutilization increased energy for all array sizes and a smaller array was the lowest-energy choice.

4) Full-System: CiM systems reduce energy by reducing off-chip data movement and by using CiM arrays to execute many MACs in parallel. To explore the full-system effects of these contributions, we put *Macro D* [20], [21] in a full system. The system includes a DRAM backing storage [50] and a chip that has parallel macros with input/output buffers, routers [2],

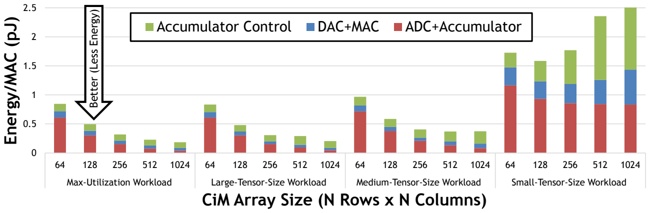


Fig. 14: ***Macro C* + Architecture**: Larger arrays can reduce energy if workload tensors are large enough to utilize them.

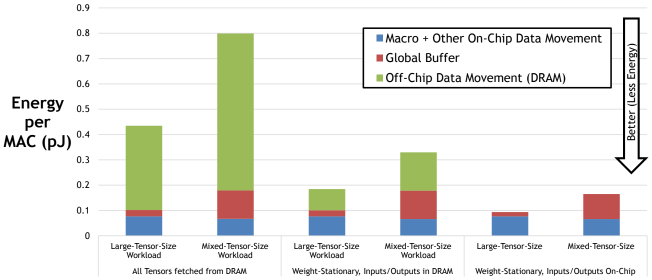


Fig. 15: ***Macro D* + Full-System**: Weight-stationary CiM saves significant energy, but benefits are limited by off-chip movement of DNN inputs/outputs between each layer. CiM will be most effective when combined with layer fusion to keep all tensors on-chip.

and a global buffer. The on-chip memory can fit any tested layer, so inputs/outputs/weights will be transferred no more than once to/from DRAM for each layer.

Tensor size influences the number of MACs performed, so we test large-tensor (large language model GPT-2 [49]) and mixed-size-tensor (ResNet18 [42]) workloads. We compare three scenarios: **(1)** inputs/outputs/weights stored off-chip in DRAM and fetched for each layer; **(2)** inputs/outputs fetched from DRAM, weights stationary (pre-loaded for each layer); and **(3)** weights stationary, inputs/outputs kept on-chip in the global buffer between layers [65], [66].

Fig. 15 shows the system energy breakdown for on-chip data movement, on-chip global buffer, and off-chip data movement. We see a significant reduction in overall energy when going from off-chip weights to weight-stationary due to fewer weight fetches. Benefits are limited, however, by input/output fetch from off-chip. Recall that this system only transfers inputs/outputs to/from DRAM once per layer; to see further benefits, it is necessary to avoid transferring inputs/outputs off-chip between layers [65], [66]. We note that weight-stationary CiM requires sufficient memory to keep all DNN weights on-chip. To store large DNNs, this may require a multi-chip pipeline [2], [67] or dense storage technologies [1].

5) Cross-Macro: CiMLoop can be used to project how a macro will scale to a new technology node and to fairly compare across different macros. We compare the three SRAM-based *Macros A/B/D*, scaling all macros to 7nm, using *Macro B* memory cells and using an 8b ADC.

Fig. 16 compares the energy efficiency of all three macros for different numbers of input and weight bits. *Macro A* computes analog MACs with 1b inputs/weights and accumulates results digitally. This lets *Macro A* flexibly leverage few-bit inputs/weights to increase energy efficiency, but the strategy is less efficient with more-bit inputs/weights. *Macros B/D* use 4b/8b analog components (shown in Fig. 3) that can increase output reuse and reduce ADC energy. However, these components are underutilized with few-bit inputs/weights, so *Macros B/D* gain little energy efficiency from few-bit operands.

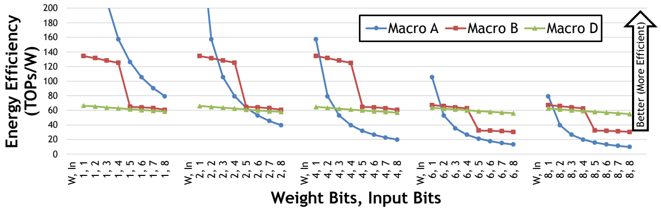


Fig. 16: ***Macros A, B, D* + Cross-Macro**: CiMLoop can fairly compare CiM implementations. The lowest-energy macro choice depends on input/weight bit precisions.

VI. Related Works

In addition to the discussed works [3]–[13], many works have explored different parts of CiM. While CiMLoop models area/energy/throughput, IBM AI Hardware Kit [68], CrossSim [69], and MemTorch [70], model DNN accuracy. Eva-CiM [71] models the CPU/CiM interface but not CiM macros. Simeuro [72] and SuperNeuro [73] model spiking (rather than deep) neural network systems. PUMA [74] provides a detailed model of a particular DNN system but does not explore the design space. Sparseloop [11], [12], like CiMLoop, uses statistical analytical models, applying them to model sparse DNN [75] systems.

VII. Conclusion

In this paper, we presented CiMLoop: a flexible, accurate, and fast model that connects all levels of the CiM stack. CiMLoop lets researchers evaluate design decisions at each level, co-design across levels, and fairly compare CiM implementations. By bringing all levels together in one model, CiMLoop can bridge the device, circuits, and architecture research communities. CiMLoop can even be used beyond CiM to model traditional [13] accelerators and those that use other paradigms such as photonics [76]. We hope that researchers will use CiMLoop to share their work, publish open-source models, and reveal new insights and co-design opportunities that leverage the contributions of all communities.

VIII. Acknowledgments

We thank the authors of the presented macros Hongyang Jia, Hyunjoon Kim, Mahmut Sinangil, Weier Wan, and Hechen Wang for providing invaluable feedback and advice as we modeled their works. We also thank Anni Lu for her guidance as we integrated NeuroSim as a plug-in to CiMLoop. This work was funded in part by Ericsson, TSMC, the MIT AI Hardware Program, and MIT Quest.

References

[1] L. Pentecost, A. Hankin, M. Donato, M. Hempstead, G.-Y. Wei, and D. Brooks, “NVMExplorer: A framework for cross-stack comparisons of embedded non-volatile memories,” in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 938–956.

[2] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, “ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars,” in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 14–26.

[3] X. Peng, S. Huang, Y. Luo, X. Sun, and S. Yu, “DNN+NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies,” in *2019 IEEE International Electron Devices Meeting (IEDM)*, 2019, pp. 32.5.1–32.5.4.

[4] P.-Y. Chen, X. Peng, and S. Yu, “NeuroSim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures,” in *2017 IEEE International Electron Devices Meeting (IEDM)*, 2017, pp. 6.1.1–6.1.4.

[5] A. Lu, X. Peng, W. Li, H. Jiang, and S. Yu, “NeuroSim validation with 40nm RRAM compute-in-memory macro,” in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2021, pp. 1–4.

[6] X. Peng, S. Huang, H. Jiang, A. Lu, and S. 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*, vol. 40, no. 11, pp. 2306–2319, 2021.

[7] L. Xia, B. Li, T. Tang, P. Gu, X. Yin, W. Huangfu, P.-Y. Chen, S. Yu, Y. Cao, Y. Wang, Y. Xie, and H. Yang, “MNSIM: Simulation platform for memristor-based neuromorphic computing system,” in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 469–474.

[8] Z. Zhu, H. Sun, T. Xie, Y. Zhu, G. Dai, L. Xia, D. Niu, X. Chen, X. S. Hu, Y. Cao, Y. Xie, H. Yang, and Y. Wang, “MNSIM 2.0: A behavior-level modeling tool for processing-in-memory architectures,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 4112–4125, 2023.

[9] Y. N. Wu, J. S. Emer, and V. Sze, “Accelergy: An architecture-level energy estimation methodology for accelerator designs,” in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[10] Y. N. Wu, V. Sze, and J. 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)*, 2020, pp. 116–118.

[11] Y. N. Wu, P.-A. Tsai, A. Parashar, V. Sze, and J. S. Emer, “Sparseloop: An analytical, energy-focused design space exploration methodology for sparse tensor accelerators,” in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2021, pp. 232–234.

[12] ——, “Sparseloop: An analytical approach to sparse tensor accelerator modeling,” in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 1377–1395.

[13] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, “Timeloop: A systematic approach to DNN accelerator evaluation,” in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 304–315.

[14] M. Horeni, P. Taheri, P. Tsai, A. Parashar, J. Emer, and S. Joshi, “Ruby: Improving hardware efficiency for tensor algebra accelerators through imperfect factorization,” in *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2022, pp. 254–266. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ISPASS55109.2022.00039

[15] A. Lu, X. Peng, W. Li, H. Jiang, and S. Yu, “NeuroSim validation with 40nm RRAM compute-in-memory macro,” in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2021, pp. 1–4.

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

[17] M. E. Sinangil, B. Erbagci, R. Naous, K. Akarvardar, D. Sun, W.-S. Khwa, H.-J. Liao, Y. Wang, and J. 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*, vol. 56, no. 1, pp. 188–198, 2021.

[18] W. Wan, R. Kubendran, S. B. Eryilmaz, W. Zhang, Y. Liao, D. Wu, S. Deiss, B. Gao, P. Raina, S. Joshi, H. Wu, G. Cauwenberghs, and H.-S. P. Wong, “33.1 a 74 TMACS/W CMOS-RRAM neurosynaptic core with dynamically reconfigurable dataflow and in-situ transposable weights for probabilistic graphical models,” in *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2020, pp. 498–500.

[19] W. Wan, R. Kubendran, C. Schaefer, S. B. Eryilmaz, W. Zhang, D. Wu, S. Deiss, P. Raina, H. Qian, B. Gao, S. Joshi, H. Wu, H.-S. P. Wong, and G. Cauwenberghs, “A compute-in-memory chip based on resistive random-access memory,” *Nature*, vol. 608, no. 7923, pp. 504–512, Aug. 2022, number: 7923 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/s41586-022-04992-8

[20] H. Wang, R. Liu, R. Dorrance, D. Dasalukunte, D. Lake, and B. Carlton, “A charge domain SRAM compute-in-memory macro with C-2C ladder-based 8-bit MAC unit in 22-nm FinFET process for edge inference,” *IEEE Journal of Solid-State Circuits*, vol. 58, no. 4, pp. 1037–1050, 2023.

[21] H. Wang, R. Liu, R. Dorrance, D. Dasalukunte, X. Liu, D. Lake, B. Carlton, and M. 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)*, 2022, pp. 36–37.

[22] H. Kim, T. Yoo, T. T.-H. Kim, and B. Kim, “Colonnade: A reconfigurable SRAM-based digital bit-serial compute-in-memory macro for processing neural networks,” *IEEE Journal of Solid-State Circuits*, vol. 56, no. 7, pp. 2221–2233, 2021.

[23] Q. Dong, M. E. Sinangil, B. Erbagci, D. Sun, W.-S. Khwa, H.-J. Liao, Y. Wang, and J. Chang, “15.3 a 351 TOPS/W and 372.4 GOPS compute-in-memory SRAM macro in 7nm FinFET CMOS for machine-learning applications,” in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2020, pp. 242–244.

[24] C.-X. Xue, T.-Y. Huang, J.-S. Liu, T.-W. Chang, H.-Y. Kao, J.-H. Wang, T.-W. Liu, S.-Y. Wei, S.-P. Huang, W.-C. Wei *et al.*, “15.4 a 22nm 2Mb ReRAM compute-in-memory macro with 121-28TOPS/W for multibit mac computing for tiny AI edge devices,” in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2020, pp. 244–246.

[25] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi, “Scaling for edge inference of deep neural networks,” *Nature Electronics*, vol. 1, no. 4, pp. 216–222, 2018.

[26] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, *Efficient Processing of Deep Neural Networks*. Springer International Publishing, 2020. [Online]. Available: https://doi.org/10.1007/978-3-031-01766-7

[27] M. 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)*, 2014, pp. 10–14.

[28] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[29] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, “PRIME: A novel processing-in-memory architecture for neural network computation in reram-based main memory,” in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 27–39.

[30] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. 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)*, 2016, pp. 1–6.

[31] M. He, C. Song, I. Kim, C. Jeong, S. Kim, I. Park, M. 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)*, 2020, pp. 372–385.

[32] S. Xie, C. Ni, A. Sayal, P. Jain, F. Hamzaoglu, and J. P. Kulkarni, “16.2 eDRAM-CIM: Compute-in-memory design with reconfigurable embedded-dynamic-memory array realizing adaptive data converters

and charge-domain computing," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 248–250.

[33] P. Narayanan, S. Ambrogio, A. Okazaki, K. Hosokawa, H. Tsai, A. Nomura, T. Yasuda, C. Mackin, S. C. Lewis, A. Friz, M. Ishii, Y. Kohda, H. Mori, K. Spoon, R. Khaddam-Aljameh, N. Saulnier, M. Bergendahl, J. Demarest, K. W. Brew, V. Chan, S. Choi, I. Ok, I. Ahsan, F. L. Lie, W. Haensch, V. Narayanan, and G. W. Burr, "Fully on-chip MAC at 14 nm enabled by accurate row-wise programming of PCM-based weights and parallel vector-transport in duration-format," *IEEE Transactions on Electron Devices*, vol. 68, no. 12, pp. 6629–6636, 2021.

[34] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan, "Computing in memory with spin-transfer torque magnetic RAM," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, pp. 470–483, 2018.

[35] R. Chen, H. Kung, A. Chandrakasan, and H.-S. Lee, "A bit-level sparsity-aware SAR ADC with direct hybrid encoding for signed expressions for AIoT applications," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2022, pp. 1–6.

[36] T. Chou, W. Tang, J. Botimer, and Z. 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*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 114–125. [Online]. Available: https://doi.org/10.1145/3352460.3358328

[37] X. Qiao, X. Cao, H. Yang, L. Song, and H. Li, "AtomLayer: A universal ReRAM-based CNN accelerator with atomic layer computation," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[38] T. Andrulis, J. S. Emer, and V. Sze, "RAELLA: Reforming the arithmetic for efficient, low-resolution, and low-loss analog PIM: No retraining required!" in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589062

[39] J. Yue, Y. Liu, F. Su, S. Li, Z. Yuan, Z. Wang, W. Sun, X. Li, and H. 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*, ser. ASPDAC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 146–151. [Online]. Available: https://doi.org/10.1145/3287624.3287635

[40] W. Li, P. Xu, Y. Zhao, H. Li, Y. Xie, and Y. 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*, ser. ISCA '20. IEEE Press, 2020, p. 832–845. [Online]. Available: https://doi.org/10.1109/ISCA45697.2020.00073

[41] N. Nayak, T. O. Odemuyiwa, S. Ugare, C. W. Fletcher, M. Pellauer, and J. S. Emer, "TeAAL: A declarative framework for modeling sparse tensor accelerators," 2023.

[42] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[43] S.-H. Sie, J.-L. Lee, Y.-R. Chen, Z.-W. Yeh, Z. Li, C.-C. Lu, C.-C. Hsieh, M.-F. Chang, and K.-T. Tang, "MARS: Multimacro architecture SRAM CIM-based accelerator with co-designed compressed neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 5, pp. 1550–1562, 2022.

[44] G. Yuan, P. Behnam, Z. Li, A. Shafiee, S. Lin, X. Ma, H. Liu, X. Qian, M. N. Bojnordi, Y. Wang, and C. 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)*, 2021, pp. 265–278.

[45] C. Chu, Y. Wang, Y. Zhao, X. Ma, S. Ye, Y. Hong, X. Liang, Y. Han, and L. Jiang, "PIM-prune: Fine-grain DCNN pruning for crossbar-based process-in-memory architecture," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[46] J. Lin, Z. Zhu, Y. Wang, and Y. 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*, ser. ASPDAC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 639–644. [Online]. Available: https://doi.org/10.1145/3287624.3287715

[47] T. P. Xiao, B. Feinberg, C. H. Bennett, V. Prabhakar, P. Saxena, V. Agrawal, S. Agarwal, and M. J. Marinella, "On the accuracy of analog neural network inference accelerators [feature]," *IEEE Circuits and Systems Magazine*, vol. 22, pp. 26–48, 2021.

[48] H. Yun, H. Shin, M. Kang, and L.-S. Kim, "Optimizing ADC utilization through value-aware bypass in ReRAM-based DNN accelerator," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 1087–1092.

[49] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[50] N. P. Jouppi, A. B. Kahng, N. Muralimanohar, and V. Srinivas, "CACTI-IO: CACTI with off-chip power-area-timing models," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 7, pp. 1254–1267, 2015.

[51] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "The aladdin approach to accelerator design and modeling," *IEEE Micro*, vol. 35, no. 3, pp. 58–70, 2015.

[52] T. Andrulis, R. Chen, H.-S. Lee, J. S. Emer, and V. Sze, "Modeling analog-digital-converter energy and area for compute-in-memory accelerator design," 2024.

[53] B. Murmann, "ADC Performance Survey 1997-2023," [Online]. Available: https://github.com/bmurmann/ADC-survey.

[54] M. Saberi, R. Lotfi, K. Mafinezhad, and W. 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*, vol. 58, no. 8, pp. 1736–1748, 2011.

[55] M. Verhelst and B. Murmann, "Area scaling analysis of CMOS ADCs," *Electronics Letters*, vol. 48, pp. 314–315, 2012.

[56] R. Schreier, J. Silva, J. Steensgaard, and G. Temes, "Design-oriented estimation of thermal noise in switched-capacitor circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 11, pp. 2358–2368, 2005.

[57] T. Song, X. Chen, X. Zhang, and Y. Han, "BRAHMS: Beyond conventional RRAM-based neural network accelerators using hybrid analog memory system," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 1033–1038.

[58] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm," *Integration*, vol. 58, pp. 74–81, 2017.

[59] P.-Y. Chen, X. Peng, and S. Yu, "NeuroSim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures," in *2017 IEEE International Electron Devices Meeting (IEDM)*, 2017, pp. 6.1.1–6.1.4.

[60] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[61] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[62] Wikipedia, "Bread — Wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=Bread&oldid=1189236003, 2023, [Online; accessed 16-December-2023].

[63] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021.

[64] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for MobileNetV3," 2019.

[65] M. Gilbert, Y. N. Wu, A. Parashar, V. Sze, and J. S. Emer, "LoopTree: Enabling exploration of fused-layer dataflow accelerators," in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2023, pp. 316–318.

[66] W. Niu, J. Guan, Y. Wang, G. Agrawal, and B. Ren, "DNNFusion: Accelerating deep neural networks execution with advanced operator fusion," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, ser. PLDI 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 883–898. [Online]. Available: https://doi.org/10.1145/3453483.3454083

[67] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *2017 IEEE International Sym-*

posium on High Performance Computer Architecture (HPCA), 2017, pp. 541–552.

[68] M. J. Rasch, D. Moreda, T. Gokmen, M. Le Gallo, F. Carta, C. Goldberg, K. El Maghraoui, A. Sebastian, and V. Narayanan, "A flexible and fast PyTorch toolkit for simulating training and inference on analog crossbar arrays," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2021, pp. 1–4.

[69] T. P. Xiao, C. H. Bennett, B. Feinberg, M. J. Marinella, and S. Agarwal, "CrossSim: accuracy simulation of analog in-memory computing." [Online]. Available: https://github.com/sandialabs/cross-sim

[70] C. Lammie and M. R. Azghadi, "MemTorch: A simulation framework for deep memristive cross-bar architectures," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[71] D. Gao, D. Reis, X. S. Hu, and C. Zhuo, "Eva-CiM: A system-level performance and energy evaluation framework for computing-in-memory architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 5011–5024, 2020.

[72] H. Zhang, N.-M. Ho, D. Y. Polat, P. Chen, M. Wahib, T. T. Nguyen, J. Meng, R. S. M. Goh, S. Matsuoka, T. Luo, and W.-F. Wong, "Simeuro: A hybrid CPU-GPU parallel simulator for neuromorphic computing chips," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 10, pp. 2767–2782, 2023.

[73] P. Date, C. Gunaratne, S. R. Kulkarni, R. Patton, M. Coletti, and T. Potok, "SuperNeuro: A fast and scalable simulator for neuromorphic computing," in *Proceedings of the 2023 International Conference on Neuromorphic Systems*, ser. ICONS '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3589737.3606000

[74] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-m. W. Hwu, J. P. Strachan, K. Roy, and D. 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*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 715–731. [Online]. Available: https://doi.org/10.1145/3297858.3304049

[75] C. Guo, B. Y. Hsueh, J. Leng, Y. Qiu, Y. Guan, Z. Wang, X. Jia, X. Li, M. Guo, and Y. Zhu, "Accelerating sparse DNN models without hardware-support via tile-wise sparsity," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20. IEEE Press, 2020.

[76] T. Andrulis, G. I. Chaudhry, V. M. Suriyakumar, J. S. Emer, and V. Sze, "Architecture-level modeling of photonic deep neural network accelerators," in *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2024.