Array 13 (2022) 100116

Contents lists available at ScienceDirect

Array

journal homepage: www.elsevier.com/locate/array

Modeling and simulating in-memory memristive deep learning systems: An overview of current efforts

Corey Lammie [a], Wei Xiang [b], Mostafa Rahimi Azghadi [a,*]

[a] College of Science and Engineering, James Cook University, Queensland 4814, Australia
[b] School of Engineering and Mathematical Sciences, La Trobe University, Victoria 3086, Australia

ARTICLE INFO

Keywords:
Device modeling
Circuit simulation
Memristors
In-Memory Computing
Deep Learning

ABSTRACT

Deep Learning (DL) systems have demonstrated unparalleled performance in many challenging engineering applications. As the complexity of these systems inevitably increase, they require increased processing capabilities and consume larger amounts of power, which are not readily available in resource-constrained processors, such as Internet of Things (IoT) edge devices. Memristive In-Memory Computing (IMC) systems for DL, entitled Memristive Deep Learning Systems (MDLSs), that perform the computation and storage of repetitive operations in the same physical location using emerging memory devices, can be used to augment the performance of traditional DL architectures; massively reducing their power consumption and latency. However, memristive devices, such as Resistive Random-Access Memory (RRAM) and Phase-Change Memory (PCM), are difficult and cost-prohibitive to fabricate in small quantities, and are prone to various device non-idealities that must be accounted for. Consequently, the popularity of simulation frameworks, used to simulate MDLS prior to circuit-level realization, is burgeoning. In this paper, we provide a survey of existing simulation frameworks and related tools used to model large-scale MDLS. Moreover, we perform direct performance comparisons of modernized open-source simulation frameworks, and provide insights into future modeling and simulation strategies and approaches. We hope that this treatise is beneficial to the large computers and electrical engineering community, and can help readers better understand available tools and techniques for MDLS development.

1. Introduction

Traditionally, Machine Learning (ML) and Deep Learning (DL) systems are trained and deployed using hardware platforms adopting the von Neumann computing architecture. While in recent years, Graphics Processor Units (GPUs) have been used to massively parallelize and accelerate the performance of these workloads [1], they are still prone to performance bottlenecks caused by the amount of data being moved back and forth between physically separated memory and processing units. IMC is a novel non-von Neumann approach, where certain computational tasks are performed in the memory itself [2], which has the potential to alleviate this bottleneck.

IMC systems can be realized by arranging memory devices in crossbar architectures, where they can be used to perform various logical and arithmetic operations [3]. These memory devices can be fabricated using legacy charge-based memory technologies, such as Static Random-Access Memory (SRAM), or emerging memristive device technologies, such as RRAM, which are introduced and discussed in Section 2. Memristive devices, in particular, have shown great promise to facilitate the acceleration and improve the power efficiency of ML and DL systems, as they can be passive, re-programmable, and non-volatile [3–8].

As depicted in Fig. 1, crossbar architectures constructed using memristive RRAM devices can be used to efficiently implement various in-memory computing operations, including Multiply-Accumulate (MAC) and VMMs operations. Previous works in the literature have exploited physical properties of memristive devices to realize a variety of commonly used operations and components of neuromorphic architectures [9–13]. Traditionally, IMC systems have been used to implement brain-inspired asynchronous neuromorphic architectures [14], realizing artificial synapses using memristive devices. However, they are also capable of accelerating VMMs, the most dominant operations in DNNs, in $\mathcal{O}(1)$, which makes them more appealing for deep learning systems [15,16].

Currently, several memristive device technologies, including RRAM and PCM, which are depicted in Fig. 2, are being actively researched [3]. However, despite continuous ongoing efforts, they are prone to various

* Corresponding author.
E-mail address: mostafa.rahimiazghadi@jcu.edu.au (M. Rahimi Azghadi).

https://doi.org/10.1016/j.array.2021.100116
Received 22 July 2021; Received in revised form 28 October 2021; Accepted 30 November 2021
Available online 20 December 2021

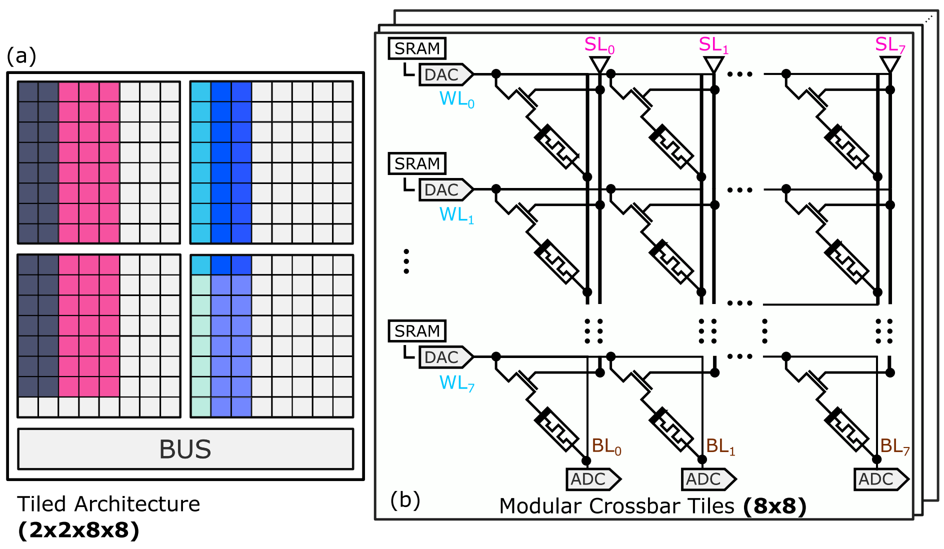

Fig. 1. (a) A modular memristive crossbar tiled architecture containing parameters from two linear and unfolded convolutional layers; both key components of traditional CNNs. Unique colors denote mapped parameters from different layers. (b) In a modular crossbar tile that is used to perform the VMM operation in-memory, SLs can be used to isolate columns of devices (BLs), in which inputs are applied to as WL voltages. BLs currents are read out using ADCs, that can be linearly related to vector–matrix product elements. *Source:* This figure is adapted from [17].

device non-idealities, which limit their accuracy and reliability to use in practical engineering settings [18]. Consequently, many large-scale simulations encompassing device and circuit non-idealities have been conducted using synaptic memristive connections for brain-inspired asynchronous neuromorphic systems [19–21] and DL systems [9]. While these simulations were traditionally performed using general purpose Simulation Program with Integrated Circuit Emphasis (SPICE)-based simulators, as the complexity of the underlying systems and neuromorphic architectures being simulated has increased, customized simulation frameworks have been developed. These frameworks are used to rapidly prototype novel network architectures as a preliminary step prior to circuit-level validation and layout using mature Computer-Aided Design (CAD) tools; for eventual circuit-level realization and large-scale fabrication.

In contrast to conventional SPICE-based simulation, modern CAD simulation frameworks adopt modern software engineering methodologies. Moreover, they are able to accurately model non-ideal device characteristics, peripheral circuitry, and modular crossbar tiles while being interfaceable using high-level language APIs. We confine the scope of this paper to MDLS, i.e., memristive IMC systems for DL system deployment, and provide a survey of existing simulation frameworks and related tools used to model large-scale MDLS.

The rest of the paper is structured as follows. In Section 2, preliminaries related to modeling and simulating in-memory MDLS are presented. In Section 3, existing CAD tools for in-memory MDLS are over-viewed. In Section 4, comparisons of modern simulation frameworks for in-memory MDLS are made, and two MDLS architectures are simulated. In Section 5, we provide an outlook for MDLS simulation frameworks. Finally, in Section 6, the paper is concluded.

2. Preliminaries

Memristors, commonly referred to as the fourth fundamental circuit element, are two-terminal passive circuit elements characterized by a relationship between the charge, $q(t) \equiv \int_{-\infty}^{t} i(\tau)d\tau$ and the flux-linkage $\varphi(t) \equiv \int_{-\infty}^{t} v(\tau)d\tau$ [22]. Memristors are capable of non-volatile storage. We depict typical unipolar and bipolar switching I-V characteristics, and schematics of popular memristive device technologies in Fig. 2.

Unfolded convolutional layers and linear (dense) layers within DL systems can be implemented using a series of MAC and VMM operations, which can be computed in-memory using memristive crossbar arrays, as depicted in Fig. 1, by encoding weights as resistance/conductance values, and inputs as WL voltages. Tiled crossbar architectures contain several modular crossbar tiles connected using a shared bus. These are also connected to additional circuitry used to realize batch-normalization, pooling, activation functions, and other computations that cannot be performed, or are not efficient, in-memory. Modular crossbar tiles consist of crossbar arrays with supporting peripheral circuitry. We refer the reader to [12] for a comprehensive description and overview of IMC accelerators for DL acceleration.

In Fig. 2, typical switching modes and schematics of popular memristive device technologies are depicted. Memristors differ from electrical resistors, as they have a voltage or current-dependent resistance state, which is dependent on the electric properties of the materials using which they are constructed. As depicted in Fig. 2(c), RRAM devices are comprised of Metal–Insulator–Metal (MIM) stacks. The resistive state of RRAM devices can be modulated by creating and disrupting Conductive Filaments (CFs), used to refer to localized concentrations of defects that allow current to flow between top and bottom electrodes.

As depicted in Fig. 2(d), typical PCM devices have a mushroom shape (amorphous region), where the bottom electrode confines heat and current. By crystallizing the amorphous region, different resistive states can be obtained [3]. As shown in Fig. 2(e), CBRAM devices are comprised of a thin solid state electrolyte layer sandwiched between oxidizable and inert electrodes. The resistive state of CBRAM devices can be modulated by driving redox reactions in the filament (solid state electrolyte layer) [23]. Finally, Fig. 2(f) shows the device structure of STT-MRAM, which contains two ferromagnetic layers and one tunnel barrier layer. The resistance of STT-MRAM devices can be modulated by modifying the orientation of a magnetic layer in a magnetic tunnel junction or spin valve using a spin-polarized current [24].

As memristive devices can only be programmed to positive resistance states, weights can either be represented using a dual-array scheme, a dual row scheme, where double the number of rows are required, or a current-mirror scheme, that is capable of operation using a singular device to represent each weight [25].

As can be observed in Fig. 1, in a 1-Transistor 1-Memristor (1T1R) arrangement, SLs can be used to individually select memristive devices. After mapping and programming weights, to perform a MAC operation, inputs are scaled and encoded as voltages, prior to being presented to WLs. Currents from each BL are read-out using ADCs, either in parallel using one ADC per column, or sequentially, using time-multiplexing.

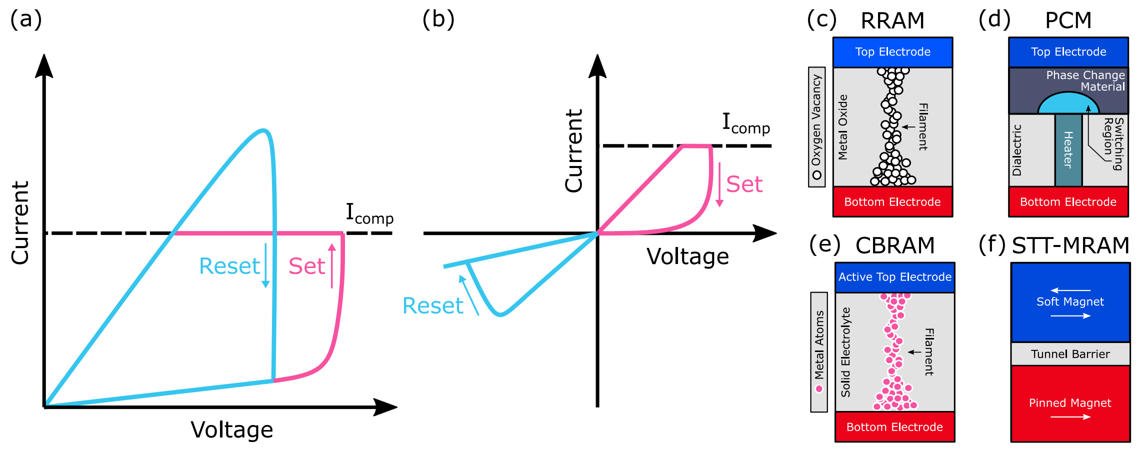

Fig. 2. Typical (a) unipolar and (b) bipolar switching modes of memristive devices and schematics of popular device technologies: (c) RRAM, (d) PCM, (e) CBRAM, and (f) STT-MRAM.

Table 1
Comparison of conventional simulation frameworks for MDLS simulation. †Not natively supported.

Simulation framework	Prog. language(s)	GPU	Pre-trained DNN conversion	TF/ PyTorch Intg.$^{\circ}$	Inference	Training	Peripheral circuitry	Supported devices	Open-source
NVMSpice [29]	Not specified (SPICE-like)				✓†	✓†		Non-volatile memories and legacy NAND flash.	
NVSim [30]	C++, C				✓†	✓†	✓	Non-volatile memories and legacy NAND flash.	✓
NVMain, NVMain 2.0 [31,32]	C++, System Verilog, Python				✓†	✓†		Non-volatile memories and hybrid non-volatile plus DRAM memory systems.	✓
MNSIM [33]	Not specified				✓†	✓†	✓	Non-volatile memories.	✓
TxSim [34]	Python	✓		✓	✓	✓	✓	Non-volatile memories and legacy NAND flash.	
PipeLayer [35]	C++	✓†	✓		✓	✓	✓	Non-volatile memories.	
Non-ideal Resistive Synaptic Device Characteristic [36]	Python	✓	✓		✓	✓	✓	Non-volatile memories and legacy NAND flash.	
Inference Accuracy Using Realistic RRAM Devices [37]	Python	✓	✓		✓			RRAM.	
RxNN [38]	C++	✓	✓		✓		✓	Non-volatile memories.	

Finally, BL currents can be correlated with desired deterministic output elements using linear regression. By time multiplexing the presentation of inputs, or duplicating modular crossbar tiles, VMMs operations can be performed in $\mathcal{O}(n)$, or $\mathcal{O}(1)$, respectively.

CAD tools can be used to convert traditional DNNs to equivalent representations using modular tiled architectures. These tools can be used to simulate the inference and training of MDLS, and to estimate power/area/latency of end-to-end implementations when various memristive devices are integrated within Complementary Metal–Oxide–Semiconductor (CMOS) processes. Models are used to simulate the behavior of peripheral circuitry and memristive devices, which can be broadly categorized as empirical or analytical (functional). Empirical models are based on, concerned with, or verified by experimental data, whereas analytical models are based on analysis or logic derived from fundamental physics of the device. In this paper, we do not emphasize specific memristive device and crossbar circuit models, as these have previously been surveyed in other works [26–28].

3. Overview of existing CAD tools

In Tables 1 and 2, we present an overview of existing *conventional* and *modernized* simulation frameworks that can be used to simulate MDLS and IMC systems utilizing non-volatile memory and legacy NAND flash devices for comparison. We categorize modernized simulation frameworks as those that support pre-trained DNN conversion and TF and/or PyTorch integration. General SPICE [39] simulation tools, such as PSPICE and LTSPICE, are not compared. Although they are the most commonly used tools for analog circuit simulation [40], they are difficult to parallelize and prohibitively slow; even when simulating large crossbar arrays using significant approximation methodologies [41,42]. Consequently, specialized and/or parallelizable CAD tools with direct integration with modern ML frameworks, such as PyTorch [43] and Tensorflow [44], are more commonly used to simulate MDLS.

Tables 1 and 2 demonstrate that while most mature conventional SPICE-based simulation frameworks, such as NVMSpice, NVSim, and NVMain, are Central Processor Unit (CPU) bound, and do not natively support pre-trained DNN conversion, inference, and training modeling, they do support a large variety of device types. In addition, they are primarily focused on the high-precision and high-speed simulation of non-volatile memories and legacy NAND flash devices. In contrast, modernized recently developed frameworks, such as DNN + NeuroSIM, MemTorch, and the IBM Analog Hardware Acceleration Kit, abstract performance-critical operations on GPUs, integrate directly with popular ML frameworks, and have well documented APIs. Moreover, they

Table 2
Comparison of modernized simulation frameworks for MDLS simulation. ‡Models are shared using Google Drive without APIs. ∘TF/PyTorch integration.

Simulation framework	Prog. language(s)	GPU	Pre-trained DNN conversion	TF/ PyTorch Intg.∘	Inference	Training	Peripheral circuitry	Supported devices	Open-source
RAPIDNN [45]	C++, SPICE		✓	✓	✓		✓	Single-level memristive devices.	
PUMA [46]	C++		✓	✓	✓		✓	Non-volatile memories and legacy NAND flash.	
DL-RSIM [47]	Python	✓	✓	✓	✓		✓	Non-volatile memories.	
Tiny but Accurate [48]	MATLAB		✓	✓	✓		✓	Non-volatile memories.	✓‡
Ultra-Efficient Memristor-Based DNN [49]	C++, MATLAB		✓	✓	✓		✓	Non-volatile memories	✓‡
MemTorch [50,51]	Python, C++, CUDA	✓	✓	✓	✓		✓	Non-volatile memories and legacy NAND flash.	✓
NeuroSim and derivatives [52–55]	C++, Python	✓	✓	✓	✓	✓	✓	Non-volatile memories and legacy NAND flash.	✓
IBM Analog Hardware Acceleration Kit [56]	C++, Python, CUDA	✓	✓	✓	✓	✓	✓	Non-volatile memories.	✓

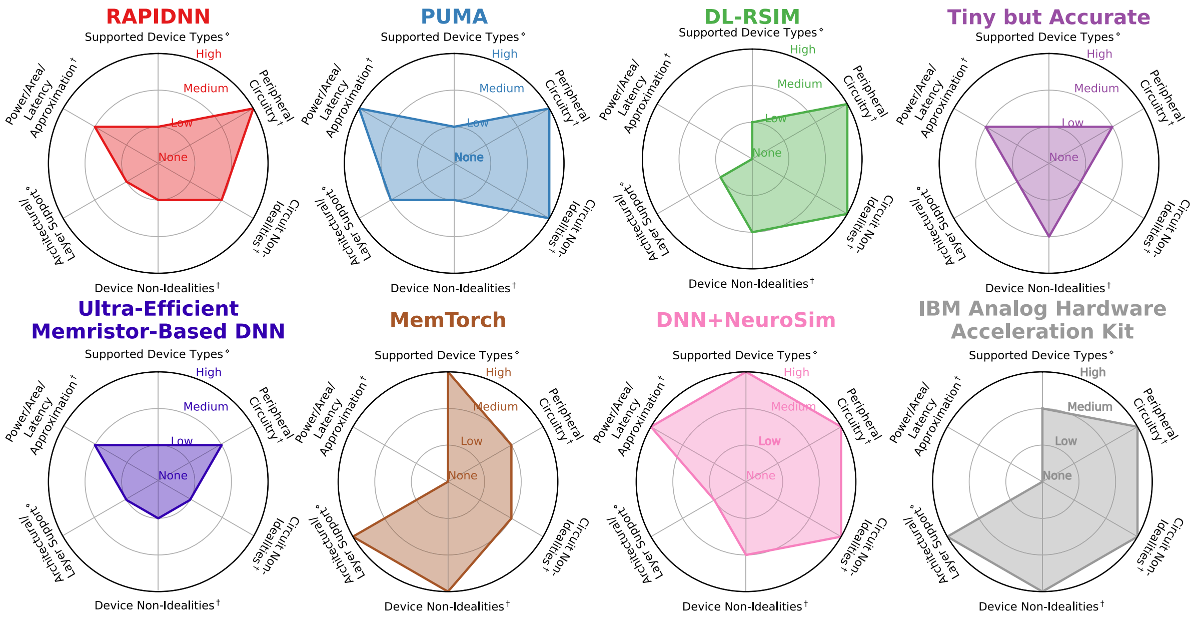

Fig. 3. Comparison of modern simulation frameworks that support pre-trained DNN conversion and TF/PyTorch integration. †Support and Accuracy. ∘Degree of Coverage.

adopt modern software engineering methodologies, and are able to accurately model non-ideal device and circuit characteristics, peripheral circuitry, and crossbar tiles. They are also directly interfaceable with other tools using accessible, general-purpose high-level programming languages; a paradigm shift from conventional SPICE-based simulation.

4. Comparison of modern simulation frameworks

While modernized simulation frameworks superficially appear similar, upon closer inspection, they are complimentary in nature. To make this clearer, in Fig. 3, we compare modern simulation frameworks, i.e., those that support pre-trained DNN conversion and TF/PyTorch integration in more detail, using radar charts. As it is shown, there is not a large overlap amongst the simulation frameworks which have been compared: RAPIDNN, PUMA, DL-RSIM, Tiny but Accurate, Ultra-Efficient Memristor-Based DNN, MemTorch, DNN + NeuroSIM, and the IBM Analog Hardware Acceleration Kit.

Although many of these simulation frameworks are still under active development, and are not fully mature, they clearly adopt different design and usability approaches. For instance, both Tiny but Accurate and Ultra-Efficient Memristor-Based DNN are built upon NVSim, whereas all other simulation frameworks are either written from scratch in lower level languages, or extend upon existing high-level GPU-accelerated computing libraries to abstract performance critical operations. Moreover, while RAPIDNN, PUMA, Tiny but Accurate, Ultra-Efficient Memristor-Based DNN and DNN + NeuroSIM can be used to generate power/area/latency reports, MemTorch and the IBM Analog Hardware Acceleration Kit support a large number of different layer types, and can be used to accurately model device non-idealities in a robust and modular manner. By adopting different design and usability approaches, all simulation frameworks can be beneficial and complement each other to be used by a variety of users with different requirements.

To determine the usability and performance of each modernized simulation framework, when possible, we used each framework to simulate the training routine of the VGG-8 [57] network architecture, and the inference routine of the GoogLeNet [58] network architecture. Both training and inference routines were evaluated using the CIFAR-10 dataset. Two separate network architectures were used for evaluation, as larger and more complex networks could not be reliably trained using existing simulation frameworks with Compute Unified Device Architecture (CUDA) support when utilizing a single GPU, even with 32 GB of Video Random-Access Memory (VRAM). Moreover, not all simulation frameworks supported convolutional layers with non-zero groups (connections between inputs and outputs), meaning that many ResNet-based architectures could not be implemented.

When possible, weights from linear and convolutional layers were mapped onto modular 1T1R crossbar tiles of size (16 × 16) using a differential weight mapping scheme, and device-to-device variability was modeled by sampling R_{ON} and R_{OFF} from normal distributions with mean values of 10 kΩ and 100 kΩ, and standard deviation values of 1000 and 10,000, respectively, i.e., $\bar{R}_{\text{ON}} = 10$ kΩ, and $\bar{R}_{\text{OFF}} = 100$ kΩ. Devices were assumed to have a finite number (6) of conductance states, and ADCs were assumed to operate at a 6-bit resolution. For inference routine simulations, 10 runs were conducted, and mean and standard deviation values were reported across all runs. For training routine simulations, mean and standard deviation values were reported across all training epochs. All codes used to perform comparisons are made publicly-accessible,[1] and can be modified to perform comparisons using different hardware technologies, network architectures, and hyper-parameters.

The RAPIDNN, PUMA, and DL-RSIM simulation frameworks are not open-source, so they could not be evaluated and directly compared in more detail. Similarly, while full precision and quantized trained models are available for the DL-RSIM and Tiny but Accurate frameworks, codes used to simulate inference routines are not. Consequently, in Fig. 4, training routines of DNN + NeuroSim and the IBM Analog Hardware Acceleration Kit are compared, and in Fig. 5, inference routines of MemTorch, DNN + NeuroSim, and the IBM Analog Hardware Acceleration Kit, are compared.

4.1. Simulation configurations

All simulations were conducted using a High Performance Computing (HPC) cluster with the following run-time hardware configuration set using the Simple Linux Utility for Resource Management (SLURM) workload manger: 1 node and 8 CPU cores (Intel Xeon 6132 series CPU sockets), 100 GB DDR4 3200 MHz Random-Access Memory (RAM), and one PCI-E 32 GB Volta V100 GPU. `torch.cuda.Event` and `timer.time()` were used to determine the execution time of various simulation components. We reiterate that all scripts provided in [1] can be used to benchmark all simulation frameworks using different software, hardware, and environmental configurations.

4.1.1. MemTorch

Using `MemTorch`,[2] modular crossbars tiles of (16 × 16) generic RRAM devices arranged using a differential weight mapping scheme were simulated. For each device, device-to-device variability was modeled by sampling R_{ON} and R_{OFF} from normal distributions with mean values of 10 kΩ and 100 kΩ, and standard deviation values of 1000 and 10,000, respectively. Devices were assumed to have a finite number (6) of evenly-spaced conductance states. The operating resolution of ADCs was set to 6-bits.

4.1.2. NeuroSim

Using `DNN_NeuroSim_V2.1`,[3] modular crossbars tiles of (16 × 16) generic RRAM devices arranged using a differential weight mapping scheme were simulated. Each device was set to have an $\bar{R}_{\text{ON}}/\bar{R}_{\text{OFF}}$ ratio of 10, with a device-to-device variation of 10%. This was done, as NeuroSim did not have the functionality to directly set $\bar{R}_{\text{ON}}$ and $\bar{R}_{\text{OFF}}$ values. The weight precision of each device and operating resolution of ADCs were set to 6-bits.

4.1.3. IBM analog hardware acceleration kit

Using the IBM Analog Hardware Acceleration Kit (denoted using `aihwkit`[4] in short-form), modular crossbar tiles could not be simulated, as they were not supported. Instead, singular tiles arranged using a differential weight mapping scheme were used to map weights of linear and convolutional layers. In lieu of support for generic RRAM device modeling with arbitrary $\bar{R}_{\text{ON}}$ and $\bar{R}_{\text{OFF}}$ values and $\bar{R}_{\text{ON}}/\bar{R}_{\text{OFF}}$ ratios, devices characterized in [59] were simulated with a device-to-device variation of 10%. The weight precision of each device could not be directly set. The operating resolution of ADCs was set to 6-bits.

4.1.4. Baseline

In addition to simulating training and inference routines using MemTorch, DNN_NeuroSim_V2.1, and the IBM Analog Hardware Acceleration Kit, baseline training and inference routines were simulated using the native PyTorch ML library for comparison. For all baseline implementations, the exact same hyper-parameters were used. `torch.cuda.amp` was used to quantize all network parameters to 16-bits to improve performance.

4.2. Training routine comparison

In Fig. 4, the performance of training routines for the VGG-8 network architecture using the CIFAR-10 dataset are compared. For NeuroSim and the IBM Analog Hardware Acceleration Kit, default nonlinear weight update parameters were used. All networks were trained for 256 epochs with a batch size of 128 using Stochastic Gradient Descent (SGD) with momentum and cross-entropy loss. An initial learning rate of 0.1 was used with fixed momentum value of 0.9. Optimizers that support adaptive learning rates were not used, as these were not supported by DNN_NeuroSim_V2.1. Instead, during training, the learning rate was decayed by one order of magnitude at epochs 100, 200, and 250 (these schedules were determined empirically), to prevent stagnation.

The functionality of each simulation framework has previously been investigated and validated [51,55,56]. Consequently, training and test set losses and accuracies were not reported or compared, as they have no bearing on the performance of each simulation framework. As can be seen in Fig. 4, the IBM Analog Hardware Acceleration Kit consumed the most RAM and GPU VRAM. While DNN_NeuroSim_V2.1 consumed more RAM than the baseline implementation, interestingly, it consumed notability less VRAM. This can be largely attributed to the large number of operations being performed on CPU and/or sequentially on GPU, rather than in parallel, and can be used to explain the relatively large elapsed time per training epoch reported by DNN_NeuroSim_V2.1, as depicted in Fig. 4(c).

To quantify the performance trade-off between GPU VRAM usage and training time, Fig. 4(f) was constructed. The baseline training routine clearly exhibits the best performance trade-off. Our findings suggest that DNN_NeuroSim_V2.1 is capable of simulating the training routine of larger and more complex network architectures, however, it does not fully utilize CUDA, and is much slower than other simulation frameworks. In contrast, the IBM Analog Hardware Acceleration Kit

[1] https://github.com/coreylammie/Modeling-and-Simulating-In-Memory-Memristive-Deep-Learning-Systems.
[2] https://github.com/coreylammie/MemTorch.
[3] https://github.com/neurosim/DNN_NeuroSim_V2.1.
[4] https://github.com/IBM/aihwkit.

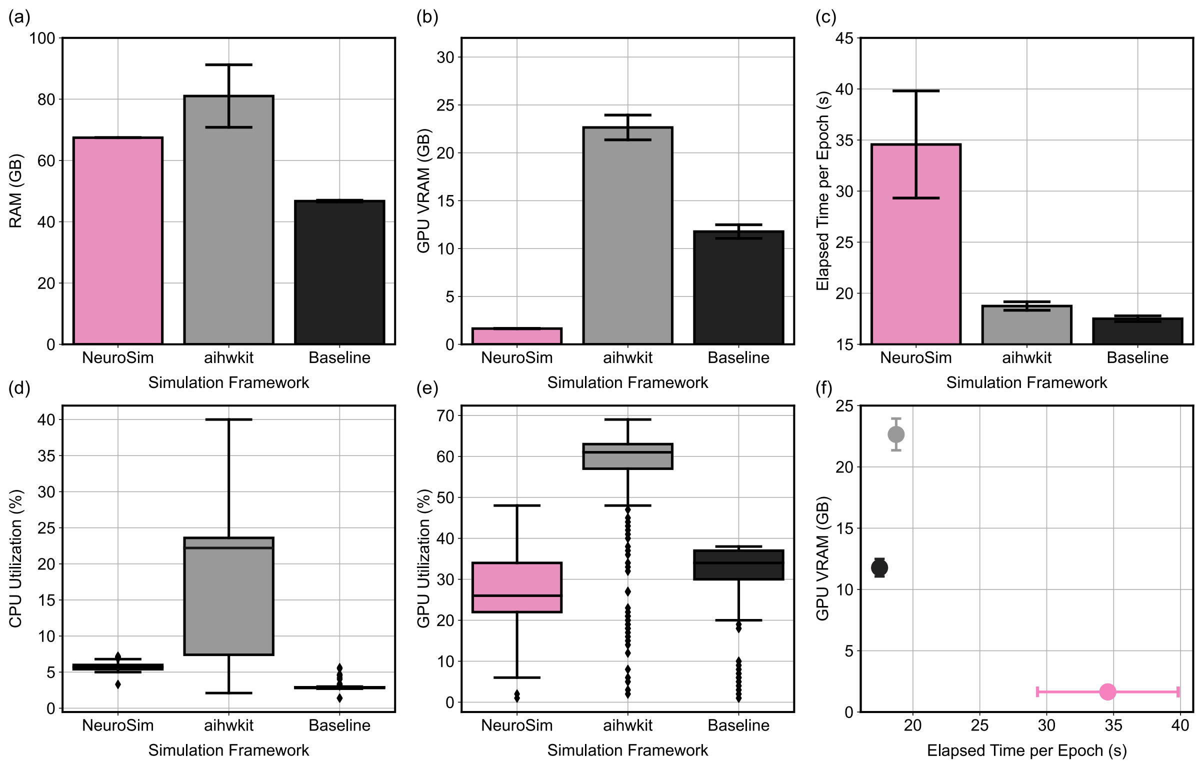

Fig. 4. Comparison of training routines of DNN + NeuroSim and the IBM Analog Hardware Acceleration Kit, for the VGG-8 network architecture, using the CIFAR-10 dataset.

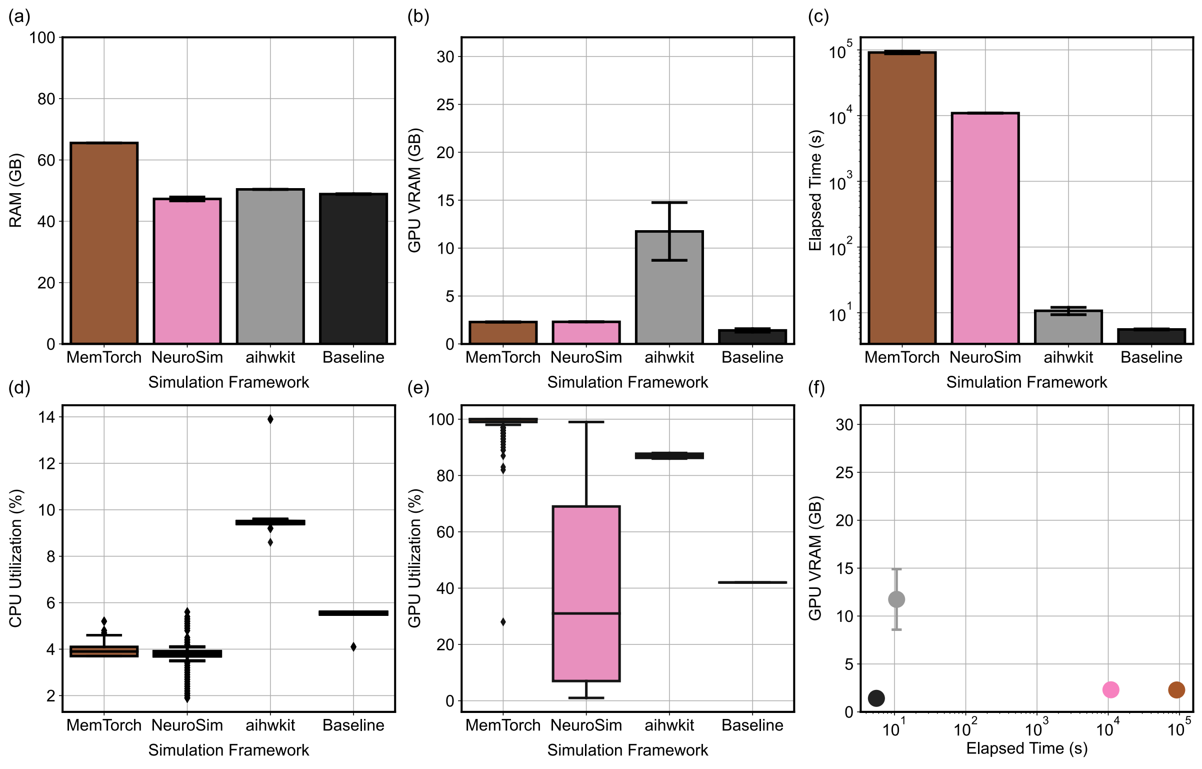

Fig. 5. Comparison of inference routines of MemTorch, DNN + NeuroSim, and the IBM Analog Hardware Acceleration Kit, for the VGG-8 network architecture, using the CIFAR-10 dataset.

fully utilizes CUDA, and is comparable in performance to the native `torch` library. However, the IBM Analog Hardware Acceleration Kit consumes a large amount of VRAM, is unable to simulate modular crossbar tiles, and is consequently unable to simulating the training routine of larger and more complex network architectures.

4.3. *Inference routine comparison*

In Fig. 5, the performance of inference routines for the GoogLeNet network architecture using the CIFAR-10 dataset are compared. Inference was performed using a batch size of 128. As can be seen in

Fig. 5(c), the IBM Analog Hardware Acceleration Kit is capable of simulating inference routines significantly faster than the MemTorch and DNN_NeuroSim_V2.1 simulation frameworks. This is while consuming more VRAM and approximately the same amount of RAM. We largely attribute this to the fact that the IBM Analog Hardware Acceleration Kit is unable to simulate modular crossbar tiles, which are difficult to parallelize using CUDA. When modular crossbar tiles are not simulated, when sufficiently small WL voltages are used to encode inputs, conventional VMMs can be used to determine output currents when 1T1R crossbars are modeled.

MemTorch and DNN_NeuroSim_V2.1 consume a similar amount of RAM and VRAM, however, MemTorch is approximately one order of magnitude slower than DNN_NeuroSim_V2.1, despite having a higher GPU utilization. We believe this is largely attributed to MemTorch's inefficient default weight-mapping scheme, as depicted in Fig. 5(c) and (d). This is especially evident when simulating large CNNs with many small convolutional layers, such as GoogLeNet. MemTorch stores convolutional kernels in a staggered arrangement, and does not share adjacent modular crossbar tiles between layers. DNN_NeuroSim_V2.1 utilizes proprietary weight mapping and data flow schemes [60], which significantly improves performance. We note that both DNN_NeuroSim_V2.1 and MemTorch under-utilize VRAM during inference, and both perform some operations sequentially and/or on CPU.

As can be seen in Fig. 5(d), our findings suggest that the IBM Analog Hardware Acceleration Kit is able to utilize VRAM to the greatest extent, however, it is unable to simulate modular crossbar tiles. DNN_NeuroSim_V2.1 is able to simulate inference routines significantly faster than MemTorch, however, it is not as customizable, as it utilizes proprietary weight mapping and data flow schemes, which cannot be easily modified.

5. Outlook

It is evident that MDLS and memristive simulation frameworks are becoming increasingly useful and popular. While the reliable, large-scale operation of reconfigurable MDLS is still arguably an open problem [61], modernized simulation frameworks and tools enable researchers from a variety of disciplines to rapidly and accurately model the behavior and operation of MDLS without specialized circuit-level SPICE simulation expertise. This is in addition to the ability to work in tandem with existing modernized ML libraries. As these simulation frameworks and the models used to simulate non-ideal circuit and device characteristics mature and grow in popularity, the development cycle and production of innovative device technologies and MDLS architectures will also continue. These new devices and architectures can be conveniently integrated into the existing tools, facilitating their quick large-scale adoption.

An increasing number of simulation frameworks have been improved using measurements from experimental data, validating their reliable and accurate operation. In future, we expect CAD tools to (i) support the end-to-end characterization of memristive devices, (ii) be natively integrated within more mature and standardized MDLS design-flows, and (iii) be capable of programming future physical re-programmable memristive circuits [62–64]. Such IMC simulation frameworks will be instrumental to the design of next generation of Artificial Intelligence (AI) hardware [56].

6. Conclusion

In this paper, we presented a survey of current simulation frameworks and related tools to model and simulate IMC MDLS. In addition, we presented a detailed comparison of modern simulation frameworks that support pre-trained DNN conversion and TF/PyTorch integration. This was performed by directly comparing the training and inference routines of two popular CNN architectures using open-source modernized simulation frameworks. Furthermore, we provided an outlook/perspective into the future of CAD tools for modeling and simulating MDLS. We demonstrated that modern simulation frameworks are complimentary in nature, and can be used by a variety of users with different requirements to facilitate current research efforts in the domains of IMC and unconventional computing.

CRediT authorship contribution statement

Corey Lammie: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization, Funding acquisition. **Wei Xiang:** Writing – review & editing, Supervision. **Mostafa Rahimi Azghadi:** Conceptualization, Methodology, Validation, Writing – review & editing, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Corey Lammie acknowledges the James Cook University (JCU) DRTPS and an IBM PhD Fellowship. Mostafa Rahimi Azghadi acknowledges a JCU Rising Star ECR Fellowship.

References

[1] Chellappa R, Theodoridis S, van Schaik A. Advances in machine learning and deep neural networks. Proc IEEE 2021;109:607–11.

[2] Verma N, Jia H, Valavi H, Tang Y, Ozatay M, Chen L-Y, Zhang B, Deaville P. In-memory computing: Advances and prospects. IEEE Solid-State Circuits Mag 2019;11:43–55.

[3] Sebastian A, Le Gallo M, Khaddam-Aljameh R, Eleftheriou E. Memory devices and applications for in-memory computing. Nature Nanotechnol 2020;15:529–44.

[4] Joshi V, Le Gallo M, Haefeli S, Boybat I, Nandakumar SR, Piveteau C, Dazzi M, Rajendran B, Sebastian A, Eleftheriou E. Accurate deep neural network inference using computational phase-change memory. Nature Commun 2020;11:2473.

[5] Hu M, Graves CE, Li C, Li Y, Ge N, Montgomery E, Davila N, Jiang H, Williams RS, Yang JJ, Xia Q, Strachan JP. Memristor-based analog computation and neural network classification with a dot product engine. Adv Mater 2018;30:1705914.

[6] Azghadi MR, Lammie C, Eshraghian JK, Payvand M, Donati E, Linares-Barranco B, Indiveri G. Hardware implementation of deep network accelerators towards healthcare and biomedical applications. IEEE Trans Biomed Circuits Syst 2020;14:1138–59.

[7] Ielmini D, Wong H-SP. In-memory computing with resistive switching devices. Nat Electron 2018;1:333–43.

[8] Rahimi Azghadi M, Chen Y, Eshraghian J, Chen J, Lin C, Amirsoleimani A, Mehonic A, Kenyon A, Fowler B, Lee J, Chang Y. Complementary metal-oxide semiconductor and memristive hardware for neuromorphic computing. Adv Intell Syst 2020;2:1900189.

[9] Liu X, Zeng Z. Memristor crossbar architectures for implementing deep neural networks. Complex Intell Syst 2021.

[10] Shahsavari M. Unconventional computation from digital to brain-like neuromorphic: Memristive computing. Éditions universitaires européennes; 2017.

[11] Azghadi MR, Linares-Barranco B, Abbott D, Leong PHW. A hybrid cmos-memristor neuromorphic synapse. IEEE Trans Biomed Circuits Syst 2017;11:434–45.

[12] Mehonic A, Sebastian A, Rajendran B, Simeone O, Vasilaki E, Kenyon AJ. Memristors—From in-memory computing, deep learning acceleration, and spiking neural networks to the future of neuromorphic and bio-inspired computing. Adv Intell Syst 2020;2:2000085.

[13] Lammie C, Eshraghian JK, Lu WD, Azghadi MR. Memristive stochastic computing for deep learning parameter optimization. IEEE Trans Circuits Syst II 2021;68:1650–4.

[14] Zidan MA, Chen A, Indiveri G, Lu WD. Memristive computing devices and applications. Cham: Springer International Publishing; 2022, p. 5–32. http://dx.doi.org/10.1007/978-3-030-42424-4_2.

[15] Lammie C, Krestinskaya O, James A, Azghadi MR. Variation-aware binarized memristive networks. In: 2019 26th IEEE international conference on electronics, circuits and systems (ICECS). 2019, p. 490–3. http://dx.doi.org/10.1109/ICECS46596.2019.8964998.

[16] Sun Z, Huang R. Time complexity of in memory matrix vector multiplication. IEEE Trans Circuits Syst II 2021.

[17] Lammie C, Rahimi Azghadi M, Ielmini D. Empirical metal-oxide RRAM device endurance and retention model for deep learning simulations. Semicond Sci Technol 2021;36:065003.

[18] Zidan MA, Strachan JP, Lu WD. The future of electronics based on memristive systems. Nat Electron 2018;1:22–9.

[19] Bichler O, Roclin D, Gamrat C, Querlioz D. Design exploration methodology for memristor-based spiking neuromorphic architectures with the xnet event-driven simulator. In: 2013 IEEE/ACM international symposium on nanoscale architectures (NANOARCH). 2013, p. 7–12. http://dx.doi.org/10.1109/NanoArch.2013.6623029.

[20] Demirag Y, Frenkel C, Payvand M, Indiveri G. Online training of spiking recurrent neural networks with phase-change memory synapses. 2021, CoRR abs/2108.01804.

[21] Boulet P, Devienne P, Falez P, Polito G, Shahsavari M, Tirilly P. N2s3, an open-source scalable spiking neuromorphic hardware simulator. 2017.

[22] Chua L. Memristor-The missing circuit element. IEEE Trans Circuit Theory 1971;18:507–19.

[23] Kund M, Beitel G, Pinnow C-U, Rohr T, Schumann J, Symanczyk R, Ufert K, Muller G. Conductive bridging ram (cbram): an emerging non-volatile memory technology scalable to sub 20nm. In: IEEE internationalelectron devices meeting, 2005. IEDM technical digest. 2005, p. 754–7. http://dx.doi.org/10.1109/IEDM.2005.1609463.

[24] Khvalkovskiy AV, Apalkov D, Watts S, Chepulskii R, Beach RS, Ong A, Tang X, Driskill-Smith A, Butler WH, Visscher PB, Lottis D, Chen E, Nikitin V, Krounbi M. Basic principles of STT-MRAM cell operation in memory arrays. 2013;46:074001.

[25] Wang Q, Wang X, Lee SH, Meng F-H, Lu WD. A deep neural network accelerator based on tiled RRAM architecture. In: 2019 IEEE international electron devices meeting (IEDM). 2019, p. 14.4.1–4. http://dx.doi.org/10.1109/IEDM19573.2019.8993641.

[26] Khalid M. Review on various memristor models, characteristics, potential applications, and future works. Trans Electr Electron Mater 2019;20:289–98.

[27] Woods W, Taha MMA, Dat Tran SJ, Bürger J, Teuscher C. Memristor panic — A survey of different device models in crossbar architectures. In: Proceedings of the 2015 IEEE/ACM international symposium on nanoscale architectures (NANOARCH´ 15). 2015, p. 106–11. http://dx.doi.org/10.1109/NANOARCH.2015.7180595.

[28] Mohammad B, Homouz D, Elgabra H. Robust hybrid memristor-CMOS memory: Modeling and design. IEEE Trans Very Large Scale Integr (VLSI) Syst 2013;21:2069–79.

[29] Fei W, Yu H, Zhang W, Yeo KS. Design exploration of hybrid CMOS and memristor circuit by new modified nodal analysis. IEEE Trans Very Large Scale Integr (VLSI) Syst 2012;20:1012–25.

[30] Dong X, Xu C, Xie Y, Jouppi NP. NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. IEEE Trans Comput-Aided Des Integr Circuits Syst 2012;31:994–1007.

[31] Poremba M, Xie Y. NVMain: An architectural-level main memory simulator for emerging non-volatile memories. In: 2012 IEEE computer society annual symposium on VLSI. 2012, p. 392–7. http://dx.doi.org/10.1109/ISVLSI.2012.82.

[32] Poremba M, Zhang T, Xie Y. NVMain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems. IEEE Comput Archit Lett 2015;14:140–3.

[33] Xia L, Li B, Tang T, Gu P, Yin X, Huangfu W, Chen P-Y, Yu S, Cao Y, Wang Y, Xie Y, Yang H. MNSIM: Simulation platform for memristor-based neuromorphic computing system. In: 2016 design, automation test in europe conference exhibition (DATE). 2016, p. 469–74.

[34] Roy S, Sridharan S, Jain S, Raghunathan A. TxSim:Modeling training of deep neural networks on resistive crossbar systems. 2021, arXiv:2002.11151 [cs, eess, stat].

[35] Song L, Qian X, Li H, Chen Y. PipeLayer: A pipelined reram-based accelerator for deep learning. In: 2017 IEEE international symposium on high performance computer architecture (HPCA). 2017, p. 541–52. http://dx.doi.org/10.1109/HPCA.2017.55.

[36] Sun X, Yu S. Impact of non-ideal characteristics of resistive synaptic devices on implementing convolutional neural networks. IEEE J Emerg Sel Top Circuits Syst 2019;9:570–9.

[37] Mehonic A, Joksas D, Ng WH, Buckwell M, Kenyon AJ. Simulation of inference accuracy using realistic RRAM devices. Front Neurosci 2019;13.

[38] Jain S, Sengupta A, Roy K, Raghunathan A. RxNN: A framework for evaluating deep neural networks on resistive crossbars. IEEE Trans Comput-Aided Des Integr Circuits Syst 2021;40:326–38.

[39] Nagel LW, Pederson D. SPICE (Simulation program with integrated circuit emphasis). Technical Report UCB/ERL M382, Berkeley: EECS Department, University of California; 1973.

[40] Gielen G, Rutenbar R. Computer-aided design of analog and mixed-signal integrated circuits. Proc IEEE 2000;88:1825–54.

[41] Song L, Zhang J, Chen A, Wu H, Qian H, Yu Z. An efficient method for evaluating RRAM crossbar array performance. Solid-State Electron 2016;120:32–40.

[42] Uppala R, Yakopcic C, Taha TM. Methods for reducing memristor crossbar simulation time. In: 2015 national aerospace and electronics conference (NAECON). 2015, p. 312–9. http://dx.doi.org/10.1109/NAECON.2015.7443089.

[43] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S. PyTorch: An imperative style, high-performance deep learning library. In: Wallach H, Larochelle H, Beygelzimer A, dAlché Buc F, Fox E, Garnett R, editors. Advances in neural information processing systems 32. Curran Associates, Inc.; 2019, p. 8024–35.

[44] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015.

[45] Imani M, Samragh Razlighi M, Kim Y, Gupta S, Koushanfar F, Rosing T. Deep learning acceleration with neuron-to-memory transformation. In: 2020 IEEE international symposium on high performance computer architecture (HPCA). 2020, p. 1–14. http://dx.doi.org/10.1109/HPCA47549.2020.00011.

[46] Ankit A, Hajj IE, Chalamalasetti SR, Ndu G, Foltin M, Williams RS, Faraboschi P, Hwu W-mW, Strachan JP, Roy K, Milojicic DS. 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. New York, NY, USA: Association for Computing Machinery; 2019, p. 715–31. http://dx.doi.org/10.1145/3297858.3304049.

[47] Lin M-Y, Cheng H-Y, Lin W-T, Yang T-H, Tseng I-C, Yang C-L, Hu H-W, Chang H-S, Li H-P, Chang M-F. DL-RSIM: A simulation framework to enable reliable ReRAM-based accelerators for deep learning. In: Proceedings of the international conference on computer-aided design, ICCAD '18. New York, NY, USA: Association for Computing Machinery; 2018, p. 1–8. http://dx.doi.org/10.1145/3240765.3240800.

[48] Ma X, Yuan G, Lin S, Ding C, Yu F, Liu T, Wen W, Chen X, Wang Y. Tiny but accurate: A pruned, quantized and optimized memristor crossbar framework for ultra efficient DNN implementation. In: 2020 25th Asia and South Pacific design automation conference (ASP-DAC). 2020, p. 301–6. http://dx.doi.org/10.1109/ASP-DAC47756.2020.9045658.

[49] Yuan G, Ma X, Ding C, Lin S, Zhang T, Jalali ZS, Zhao Y, Jiang L, Soundarajan S, Wang Y. An ultra-efficient memristor-based DNN framework with structured weight pruning and quantization using ADMM. In: 2019 IEEE/ACM international symposium on low power electronics and design (ISLPED). 2019, p. 1–6. http://dx.doi.org/10.1109/ISLPED.2019.8824944.

[50] Lammie C, Azghadi MR. MemTorch: A simulation framework for deep memristive cross-bar architectures. In: 2020 IEEE international symposium on circuits and systems (ISCAS). 2020, p. 1–5. http://dx.doi.org/10.1109/ISCAS45731.2020.9180810.

[51] Lammie C, Xiang W, Linares-Barranco B, Azghadi MR. MemTorch: An open-source simulation framework for memristive deep learning systems. 2021, arXiv:2004.10971 [cs].

[52] Chen P-Y, Peng X, Yu S. NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning. IEEE Trans Comput-Aided Des Integr Circuits Syst 2018;37:3067–80.

[53] Peng X, Huang S, Luo Y, Sun X, Yu S. 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, p. 32.5.1–4. http://dx.doi.org/10.1109/IEDM19573.2019.8993491.

[54] Peng X, Huang S, Jiang H, Lu A, Yu S. DNN+NeuroSim V2.0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training. IEEE Trans Comput-Aided Des Integr Circuits Syst 2020;1.

[55] Lu A, Peng X, Li W, Jiang H, Yu S. NeuroSim simulator for compute-in-memory hardware accelerator: Validation and benchmark. Front Artif Intell 2021;4.

[56] Rasch MJ, Moreda D, Gokmen T, Le Gallo M, Carta F, Goldberg C, El Maghraoui K, Sebastian A, Narayanan V. 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, http://dx.doi.org/10.1109/AICAS51828.2021.9458494.

[57] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. In: International conference on learning representations. 2015.

[58] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A. Going deeper with convolutions. In: 2015 IEEE conference on computer vision and pattern recognition (CVPR). 2015, p. 1–9. http://dx.doi.org/10.1109/CVPR.2015.7298594.

[59] Gong N, Idé T, Kim S, Boybat I, Sebastian A, Narayanan V, Ando T. Signal and noise extraction from analog memory elements for neuromorphic computing. Nature Commun 2018;9:2102.

[60] Peng X, Liu R, Yu S. Optimizing weight mapping and data flow for convolutional neural networks on processing-in-memory architectures. IEEE Trans Circuits Syst I Regul Pap 2020;67:1333–43.

[61] Qin Y-F, Bao H, Wang F, Chen J, Li Y, Miao X-S. Recent progress on memristive convolutional neural networks for edge intelligence. Adv Intell Syst 2020;2:2000114.

[62] Cong J, Xiao B. mrFPGA: A novel FPGA architecture with memristor-based reconfiguration. In: 2011 IEEE/ACM international symposium on nanoscale architectures. 2011, p. 1–8. http://dx.doi.org/10.1109/NANOARCH.2011.5941476.

[63] Ho PWC, Almurib HAF, Kumar TN. Configurable memristive logic block for memristive-based FPGA architectures. Integration 2017;56:61–9.

[64] Tolba MF, Fouda ME, Hezayyin HG, Madian AH, Radwan AG. Memristor FPGA IP core implementation for analog and digital applications. IEEE Trans Circuits Syst II 2019;66:1381–5.

Corey Lammie is currently pursuing a Ph.D. in Computer Engineering at James Cook University (JCU), where he completed his undergraduate degrees in Electrical Engineering (Honors) and Information Technology in 2018. His main research interests include brain-inspired computing, and the simulation and hardware implementation of Spiking Neural Networks (SNNs) and Artificial Neural Networks (ANNs) using ReRAM devices and FPGAs. He has received several awards and fellowships including the intensely competitive 2020–2021 IBM international Ph.D. Fellowship, a Domestic Prestige Research Training Program Scholarship (the highest paid Ph.D. scholarship in Australia), the 2020 Circuits and Systems (CAS) Society Pre-Doctoral Grant, and the 2017 Engineers Australia CN Barton Medal awarded to the best undergraduate engineering thesis at JCU. Corey has served as a reviewer for several journals and conferences including the IEEE Internet of Things Journal and the IEEE International Symposium on Circuits and Systems (ISCAS).

Wei Xiang is currently Cisco Chair of AI and Internet of Things and Founding Director of Cisco-La Trobe Centre for AI and Internet of Things. He is also Director of Higher Degree Research Program at SmartSat CRC. Prior to joining in La Trobe, he was Foundation Chair and Head of Discipline of Internet of Things Engineering at James Cook University, Cairns, Australia. Due to his instrumental leadership in establishing Australia's first accredited Internet of Things Engineering degree program, he was selected into Pearcy Foundation's Hall of Fame in October 2018. He is an elected Fellow of the IET in UK and Engineers Australia. He received the TNQ Innovation Award in 2016, and Pearcey Entrepreneurship Award in 2017, and Engineers Australia Cairns Engineer of the Year in 2017. He has been awarded several prestigious fellowship titles, including a Queensland International Fellowship, an Endeavor Research Fellowship, a Smart Futures Fellow, and a JSPS Invitational Fellow. He is the Vice Chair of the IEEE Northern Australia Section.

Mostafa Rahimi Azghadi received the Ph.D. degree in electrical and electronic engineering from The University of Adelaide, Adelaide, SA, Australia, in 2014. From 2012 to 2014, he was a Visiting Ph.D. Student with the Neuromorphic Cognitive System Group, Institute of Neuroinformatics, University and Swiss Federal Institute of Technology (ETH), Zürich, Switzerland. He is currently a Senior Lecturer with the College of Science and Engineering, James Cook University, Townsville, QLD, Australia, where he is researching neuromorphic engineering and brain-inspired architectures and developing custom hardware and software solutions for a variety of engineering applications ranging from medical imaging to precision agriculture. Dr. Azghadi was a recipient of several national and international awards and scholarships, such as the 2020 JCU Award for Excellence in Innovation and Change, the Queensland Young Tall Poppy Science Award in 2017, and the South Australia Science Excellence Awards in 2015. He was a recipient of the Doctoral Research Medal and the Adelaide University Alumni Medal in 2014.