SpikeSim: An End-to-End Compute-in-Memory Hardware Evaluation Tool for Benchmarking Spiking Neural Networks

Abhishek Moitra\textsuperscript{a}, Graduate Student Member, IEEE, Abhiroop Bhattacharjee\textsuperscript{a}, Graduate Student Member, IEEE, Runcong Kuang\textsuperscript{b}, Gokul Krishnan\textsuperscript{c}, Member, IEEE, Yu Cao\textsuperscript{a}, Fellow, IEEE, and Priyadarshini Panda\textsuperscript{a}, Member, IEEE

Abstract—Spiking neural networks (SNNs) are an active research domain toward energy-efficient machine intelligence. Compared to conventional artificial neural networks (ANNs), SNNs use temporal spike data and bio-plausible neuronal activation functions such as leaky-integrate fire/ integrate fire (LIF/IF) for data processing. However, SNNs incur significant dot-product operations causing high memory and computation overhead in standard von-Neumann computing platforms. To this end, in-memory computing (IMC) architectures have been proposed to alleviate the “memory-wall bottleneck” prevalent in von-Neumann architectures. Although recent works have proposed IMC-based SNN hardware accelerators, the following key implementation aspects have been overlooked: 1) the adverse effects of crossbar nonideality on SNN performance due to repeated analog dot-product operations over multiple time-steps and 2) hardware overheads of essential SNN-specific components, such as the LIF/IF and data communication modules. To this end, we propose SpikeSim, a tool that can perform realistic performance, energy, latency and area evaluation of IMC-mapped SNNs. SpikeSim consists of a practical monolithic IMC architecture called SpikeFlow for mapping SNNs. Additionally, the nonideality computation engine (NICE) and energy–latency–area (ELA) engine performs hardware-realistic evaluation of SpikeFlow-mapped SNNs. Based on 65nm CMOS implementation and experiments on CIFAR10, CIFAR100 and TinyImagenet datasets, we find that the LIF/IF neuronal module has significant area contribution (\( \approx 11\% \)) of the total hardware area. To this end, we propose SNN topological modifications that leads to 1.24x and 10x reduction in the neuronal module’s area and the overall energy-delay-product value, respectively. Furthermore, in this work, we perform a holistic comparison between IMC implemented ANN and SNNs and conclude that lower number of time-steps are the key to achieve higher throughput and energy-efficiency for SNNs compared to 4-bit ANNs. The code repository for the SpikeSim tool is available at Github link.

Index Terms—Analog crossbars, emerging devices, in-memory computing (IMC), spiking neural networks (SNNs).

I. INTRODUCTION

IN THE last decade, spiking neural networks (SNNs) have gained significant attention in the context of energy-efficient machine intelligence [1]. SNNs encode input data information with discrete binary spikes over multiple time-steps making them highly suitable for asynchronous event-driven input processing applications [2], [3]. Recent works have proposed full-scale general-purpose von-Neumann architectures leveraging the temporal processing property of SNNs [4], [5]. Other works, such as [6] and [7] have proposed novel dataflow to minimize the hardware overhead in von-Neumann implementation of SNNs. However, SNNs like conventional artificial neural networks (ANNs) entail significant dot-product operations leading to high memory and energy overhead when implemented on traditional von-Neumann architectures (due to the “memory wall bottleneck”) [8], [9]. To this end, analog in-memory computing (IMC) architectures [10], [11], [12] have been proposed to perform analog dot-product or multiply-and-accumulate (MAC) operations to achieve high memory bandwidth and compute parallelism, thereby overcoming the “memory wall bottleneck.”

Being an emerging and heavily researched computing paradigm, IMC architectures require hardware evaluation platforms for fast and accurate algorithm benchmarking. To this effect, many state-of-the-art hardware evaluation frameworks [14], [15], [16], [17] have been proposed for realistic evaluation of IMC-mapped ANNs. However, they are unsuitable for hardware-realistic SNN evaluations as they lack key architectural modifications required for temporal spike processing and nonlinear activation functions, such as leaky integrate fire/integrate fire (LIF/IF). In the context of hardware evaluation platforms for SNNs, works, such as [18] and [19] have been proposed for benchmarking SNN training on digital CMOS platforms. Additionally, works, such as [9], propose IMC architectures for SNN inference. However, they lack several practical architectural considerations such as

Manuscript received 20 September 2022; revised 23 February 2023; accepted 17 April 2023. Date of publication 10 May 2023; date of current version 20 October 2023. This work was supported in part by CoCoSy; in part by the JUMP2.0 Center and CBRC a JUMP1.0 Center sponsored by DARPA and SRC; in part by Google Research Scholar Award; in part by the National Science Foundation CAREER Award, TRI (Abu Dhabi); in part by the DARPA AI Exploration (AIE) Program; and in part by the DoE MMICC Center SEACROGS under Award DE-SC0023198. This article was recommended by Associate Editor M. Zapater. (Abhishek Moitra and Abhiroop Bhattacharjee contributed equally to this work.) (Corresponding author: Abhishek Moitra.)

Abhishek Moitra, Abhiroop Bhattacharjee, and Priyadarshini Panda are with the Department of Electrical Engineering, Yale University, New Haven, CT 06520 USA (e-mail: abhishek.moitra@yale.edu).

Runcong Kuang and Yu Cao are with the School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, AZ 85287 USA.

Gokul Krishnan is with Meta Reality Labs, Redmond, WA, USA.

Digital Object Identifier 10.1109/TCAD.2023.3274918
nonidealities incurred during analog MAC computations [20], [21], [22], data communication overhead among others rendering them unsuitable for a holistic hardware evaluation for IMC mapped SNNs. All of these have been qualitatively illustrated and compared in Table I. Finally, it is noteworthy to mention works, such as [23] and [24] that have proposed fully memristive SNNs to perform analog LIF functionality. However, fully memristive SNNs require mapping all the layer weights onto the same crossbar array limiting their applicability to small-scale SNN networks only. Therefore, in the current literature, there is an evident gap between the SNN algorithm design and a holistic evaluation platform for hardware-realistic benchmarking of these algorithms.

To this end, we propose SpikeSim, an end-to-end hardware evaluation tool for benchmarking SNN inference algorithms. SpikeSim consists of a monolithic IMC-based tiled hardware architecture called SpikeFlow that maps a given SNN onto nonideal analog crossbars. In SpikeFlow, we incorporate SNN-specific nonlinear activation functions, such as LIF/IF neuron and leverage the binary spike input data to propose a lightweight module (the DIFF module) for facilitating signed MAC operations without the need for traditional dual-crossbar approach [14], [25]. For hardware-realistic SNN inference performance benchmarking, we develop a nonideality computation engine (NICE). NICE incorporates a nonideality-aware weight encoding to improve the robustness of SNNs when mapped on analog crossbars [26]. NICE incorporates circuit analysis methods to realize nonideal MAC operations and provide hardware realistic SNN inference performance. Furthermore, we design an energy–latency–area (ELA) engine to benchmark hardware realistic energy, latency, and area of the SpikeFlow-mapped SNN.

The key contributions of our work can be summarized as follows.

1) We propose SpikeSim which is an end-to-end hardware benchmarking tool for SNN inference. SpikeSim consists of SpikeFlow—a tiled memristive crossbar architecture. SpikeFlow incorporates LIF/IF functionality, and a novel fully digital DIFF module that eliminates dual-crossbar approach for signed MAC computations [14], [25]. Additionally, it contains NICE and ELA engines for crossbar-realistic hardware evaluations.

2) We develop NICE to perform fast and realistic modeling of resistive and device conductance variation nonidealities for crossbar-aware performance evaluations of SNNs. NICE incorporates a nonideality aware (NI-aware) weight encoding scheme that improves the inference accuracy of pretrained SNNs implemented on analog crossbars.

3) We perform extensive hardware evaluations on benchmark datasets—CIFAR10, CIFAR100 [27], and TinyImageNet [28] and unravel that the neuronal module consumes a significant portion of the total chip area (11%–30%) owing to the requirement to store a large number of membrane potentials in between time-steps.

4) Through extensive experiments we show that simple SNN topological modifications, such as reducing the number of output channels in the first convolutional layer, can ameliorate the area overhead of the neuron module by 1.24× and improve the energy-delay product (EDP) by 10×. Furthermore, we show that the NI-aware weight encoding improves the crossbar-mapped SNN accuracy by more than 70% (for CIFAR10 dataset) compared to vanilla weight encoding onto the SpikeFlow architecture.

5) Finally, we compare the performance as well as area and energy distributions of crossbar mapped VGG9 ANN and SNNs trained on CIFAR10 dataset. We find that SNNs exhibit ∼ 1000 × higher neuronal module area compared to ANNs and can achieve iso-performance and higher energy-efficiency and throughput benefits at small value of time-steps (T = 3, 4, 5) compared to 4-bit ANNs.

To the best of our knowledge, SpikeSim is the first hardware-realistic evaluation platform for SNNs mapped on the IMC architecture. Through SpikeSim, we bring out some of the key parameters in the SNN algorithm and IMC architecture design that can potentially lead to IMC-aware SNN research directions in the future.

II. RELATED WORKS

A. Hardware Evaluation Platforms for ANN Inference

Eyeriss [13] has proposed a reconfigurable digital systolic-array architecture for energy-efficient ANN accelerators. The authors show that data transfer from DRAM memory to the computation unit contributes significantly to the energy consumption in von-Neumann ANN accelerators and hence propose a row-stationary dataflow to mitigate the memory overhead. More recent works such as ISAAC [29], used IMC architectures such as analog crossbars to perform fast and energy-efficient computation of ANNs. They performed extensive hardware evaluation with different crossbar sizes, analog-to-digital converter (ADC) precision among others. PUMA [12] proposes a memristive crossbar-based ANN accelerator that uses graph partitioning and custom instruction set architecture to schedule MAC operations in a multicrossbar architecture. The work by Chen et al. [14] Neurosim, proposes an end-to-end hardware evaluation platform for evaluating monolithic analog crossbar-based ANN accelerators. Recent work SIAM by Krishnan et al. [17] proposed an end-to-end hardware evaluation platform for chiplet-based analog crossbar-based ANN accelerators. While the above works provide state-of-the-art evaluation platforms for ANN
accelerators, they are insufficient for accurate SNN evaluation as they lack critical architectural modifications required for temporal spike data processing and LIF/IF activation functionalities.

B. Hardware Evaluation Platforms for SNN Inference

In a recent work SpinalFlow [6], Narayanan et al. showed that naive hardware implementation of SNNs on Spiking Eyeriss-like architecture lowers the energy efficiency claimed by SNNs. To this end, the work proposed architectural changes and used a tick-batched dataflow to achieve higher energy efficiency and lower hardware overheads. Another work RESPARC [9] proposed analog crossbar-based hardware accelerators for energy-efficient implementation of SNNs. The energy efficiency of their implementation is achieved due to the event-driven communication and computation of spikes. However, the work overlooks the underlying hardware overheads for event-driven communication and the effect of analog crossbar nonidealities on SNN performance.

Given the current literature gap in IMC-based hardware evaluation platforms for SNNs, we propose SpikeSim, an end-to-end platform for hardware realistic benchmarking of SNNs implemented on IMC architectures. SpikeSim contains SpikeFlow crossbar architecture that incorporates SNN-specific spike data processing and LIF/IF Neuron functionality. SpikeSim also incorporates the NICE and ELA engine for hardware-realistic performance, energy, latency, and area evaluation of IMC-mapped SNNs.

III. BACKGROUND

A. Spiking Neural Networks

SNNs [1], [30] have gained attention due to their potential energy-efficiency compared to standard ANNs. The main feature of SNNs is the type of neural activation function for temporal signal processing, which is different from a ReLU activation for ANNs. A leak-integrate-and-fire (LIF) neuron is a temporal signal processing, which is different from a ReLU activation for ANNs. A leak-integrate-and-fire (LIF) neuron is typically used as an activation function for SNNs. The LIF neuron has a membrane potential which accumulates the weighted summation of asynchronous spike inputs , which can be formulated as follows:

\[
U^t_i = \lambda U^{t-1}_i + \sum_j w_{ij} S_j^t. \tag{1}
\]

Here, \( t \) stands for time-step, and \( w_{ij} \) is for weight connections between neuron \( i \) and neuron \( j \). Also, \( \lambda \) is a leak factor. The LIF neuron \( i \) accumulates membrane potential and generates a spike output \( o^t_i \) whenever membrane potential exceeds the threshold \( \theta \)

\[
o^t_i = \begin{cases} 
1, & \text{if } u^t_i > \theta \\
0, & \text{otherwise.} 
\end{cases} \tag{2}
\]

The membrane potential is reset to zero after firing. This integrate-and-fire behavior of an LIF neuron generates a nondifferentiable function, which is difficult to be used with standard backpropagation.

To address the nondifferentiability, various training algorithms for SNNs have been studied in the past decade.

\[
\begin{align*}
\end{align*}
\]

Fig. 1. IMC crossbar array with input voltages \( V_i \), IMC devices bearing synaptic conductances \( G_{ij} \), and output currents \( I_j \).

ANN-SNN conversion methods [31], [32], [33], [34], [35] convert pretrained ANNs to SNNs using weight (or threshold) scaling in order to approximate ReLU activation with LIF/IF activation. They can leverage well-established ANN training methods, resulting in high accuracy on complex datasets. On the other hand, surrogate gradient learning addresses the nondifferentiability problem of an LIF/IF neuron by approximating the backward gradient function [36]. Surrogate gradient learning can directly learn from the spikes, in a smaller number of time-steps.

Based on the surrogate learning, several input data encoding schemes have been compared. A recent work [37] compares two state-of-the-art input data encoding techniques- Direct Encoding and Rate Encoding. Rate encoding converts a input data to stochastically distributed temporal spikes using poisson coding technique [38]. In contrast, direct encoding leverages features directly extracted from the inputs over multiple time-steps. It has been shown that direct encoding schemes can achieve higher performance at lower number of time-steps.

B. Analog Crossbar Arrays and Their Nonidealities

Analog crossbars consist of 2-D arrays of IMC devices, digital-to-analog converters (DACs) and ADCs and write circuits for programming the IMC devices. The activations of a neural network are fed in as analog voltages \( V_i \) to each row of the crossbar and weights are programmed as synaptic device conductances \( (G_{ij}) \) at the cross-points as shown in Fig. 1. For an ideal \( N \times M \) crossbar during inference, the voltages interact with the device conductances and produce a current (governed by Ohm’s Law).

Consequently, by Kirchoff’s current law, the net output current sensed at each column \( j \) is the sum of currents through each device, i.e., \( I_j(\text{ideal}) = \sum_{i=1}^{N} G_{ij} * V_i \).

We term the matrix \( G_{\text{ideal}} \) as the collection of all \( G_{ij} \)’s for a crossbar. However, in reality, the analog nature of the computation leads to various hardware noise or nonidealities, such as interconnect parasitic resistances and synaptic device-level variations [16], [20], [26], [39], [40]. This results in a \( G_{\text{non-ideal}} \) matrix, with each element \( G'_{ij} \) incorporating the impact of the nonidealities. Consequently, the net output current sensed at each column \( j \) in a nonideal scenario becomes \( I_j(\text{non-ideal}) = \sum_{i=1}^{N} G'_{ij} * V_i \), which deviates from its ideal value. This manifests as huge accuracy losses for neural networks mapped onto crossbars. Larger crossbars entail greater nonidealities, resulting in higher accuracy losses [16], [20], [26], [41].
Fig. 2. SpikeSim entails. (a) SNN mapping on the SpikeFlow hardware architecture, (b) hardware realistic performance evaluation using NICE, and (c) hardware evaluation using ELA engine.

| TABLE II | TABLE DESCRIBING VARIOUS SNN-LEVEL, CIRCUIT-LEVEL, AND DEVICE-LEVEL PARAMETERS PERTAINING TO SPIKE SIM |
|-------------------------|----------------------------------------------------------------------------------------------------------|
| **SNN Parameters**      |                                                                                                          |
| Network Topology        | SNN network structure information                                                                    |
| Sparsity                | SNN layer-specific Spike Sparsity                                                                      |
| k                       | Weight Quantization                                                                                  |
| \(k_{mem}\)             | Membrane Potential Quantization                                                                      |
| \(T\)                   | Number of Time-Steps                                                                                  |
| Activation Type         | LIF or IF                                                                                              |
| **Circuit Parameters**  |                                                                                                          |
| NoC Topology            | Mesh or Tree type                                                                                     |
| \(SU\)                  | DIFF module SpeedUp                                                                                   |
| Scheduling Factor       | Layer Scheduling Factor                                                                               |
| Clock Frequency         | Frequency of operation                                                                                 |
| \(X\)                   | IMC Crossbar Array Size                                                                               |
| \(N_C\)                 | Crossbar count in each PB                                                                             |
| \(N_{PE}\)              | PB count in each Tile                                                                                  |
| \(MUX Size\)            | Number of columns multiplexed                                                                       |
| \(B_{GB}, B_{TB}, B_{PB}\) | Global, Tile and PE Buffer Size                                                                     |
| \(B_{TIB}, B_{PIB}\)    | Tile and PE Input Buffer Size                                                                         |
| NoC Width               | NoC Channel Width                                                                                     |
| \(V_{DD}\)              | Supply Voltage                                                                                         |
| \(V_{read}\)            | Read Voltage                                                                                          |
| \(h\)                   | Precision of the Crossbar ADC                                                                        |
| \(r\)                   | Column Parasitic Resistance                                                                         |
| **Device Parameters**   |                                                                                                          |
| Technology              | CMOS technology                                                                                        |
| IMC device              | SRAM or RRAM device                                                                                   |
| Bits/Cell               | Precision of 1 IMC device                                                                             |
| \(R_{on} \) and \(R_{off}\) | On and Off IMC device resistances                                                                    |
| \(\sigma\)              | Synaptic Conductance Variation                                                                      |

IV. SPIKESIM

SpikeSim platform as shown in Fig. 2 requires various SNN, circuit and device parameter inputs (details provided in Table II) for the hardware evaluation. It consists of three different stages.

1) **SpikeFlow Mapping**: A pretrained SNN is partitioned and mapped on a realistic analog crossbar architecture called SpikeFlow (see Section IV-A for details).

2) **Nonideality Computation Engine**: Incorporates circuit analysis and ADC quantization to evaluate hardware-realistic inference performance of SpikeFlow mapped SNNs (see Section IV-B).

3) **ELA Engine**: Computes the energy, latency, and area of the SpikeFlow-mapped SNN (see Section IV-C).

A. SpikeFlow Architecture

**Architecture Overview**: SpikeFlow (shown in Fig. 3) follows a hierarchical analog crossbar-based monolithic chip architecture [14]. The top hierarchy consists of Tiles and digital peripheral modules, such as the global buffer (GB), pooling module (PO), global accumulator (GA), and LIF/IF neuronal module for storage, pooling, accumulation, and LIF/IF neuronal activation functionality, respectively. The Tiles and peripheral modules are connected by a network on chip (NoC) interconnect [17]. Each Tile consists of a Tile Input Buffer, a fixed number of processing elements (PEs) and peripherals—tile accumulator (TA), and tile buffer (TB). Each PE consists of a PE Input Buffer, a fixed number of analog crossbars (C) and peripherals—PE Accumulator (PA) and PE Buffer (PB). Inside the Tile and PE, all modules are connected using an H-Tree—point to point interconnect [14]. The PB, TB and GB store the MAC outputs from the crossbar, PE and Tile, respectively. Similarly, PA, TA and GA accumulate the partial sum outputs from the crossbars, PE and Tile, respectively.

Each crossbar, consists of an \(X \times X\) IMC device array, Input Peripherals, Multiplexers, ADCs, Shift-and-Add circuit, and DIFF modules. Note that the DIFF and LIF/IF Neuronal Module are specific to SpikeFlow designed to leverage SNN-specific functionalities. The Multiplexers facilitate sharing of crossbar columns with flash ADCs, Shift-and-Add circuit and the DIFF modules. The number of columns shared by an ADC, Shift-and-Add and DIFF module depends on the \(MUX size\) parameter shown in Table II. Shift-and-Add circuit are incorporated to support bit-splitting of weights. Typically, crossbars in ANNs [14] require separate Shift-and-Add circuit to support input bit-serialization and bit-splitting for weights. Due to the binary spike input, SpikeFlow only requires shift-and-add circuit to support weight splitting and not input serialization. Additionally, binary spike data allows replacing the traditional dual-crossbar approach [14] for performing signed MAC computation with fully digital DIFF modules which reduce SpikeFlow’s crossbar area significantly compared to ANN crossbars. Note, the crossbar area in case of SpikeFlow includes the area of the IMC device area, Input Peripherals, Multiplexers, ADCs, Shift-and-Add circuit and DIFF modules.

The Input Peripherals contain switch matrices [14] for selecting the bit-lines (BL0-BLX), select lines (SL0-SLX), and word lines (WL0-WLX). The BLs are used to provide input \(V_{read}\) voltage to the IMC device terminal. WLs facilitate crossbar row selection and SLs carry the output current in each column. In the case of RRAM IMC-crossbars, the input peripherals additionally contain level-shifters that provide higher write voltages for RRAM device programming [14].
The weight mapping strategy used in Neurosim [14] for partitioning and mapping the pretrained software SNN weights. Note, that the spike is determined by the address appended with the spike. The destination address of the output is relayed via NoC to the Tile Input Buffer of the tile.

Fig. 4. Architecture of the LIF/IF Neuronal Module.

cache) or 0 (if t = 0 or membrane potential is reset because of neuron firing). Depending on the LIF signal being high or low, the output of the adder (integration functionality) or the subtractor (leaky-integration functionality) is passed on to the comparator. A neuron spikes at time t (produces a binary “1”) if the membrane potential U'i is greater than the threshold (Ti which is specific to layer i). The neuronal spike output is relayed via NoC to the Tile Input Buffer of the tile mapping the preceding layer. The destination address of the spike is determined by the address appended with the spike. Note, that the V_mem Cache is an SRAM memory required to store the membrane potentials over multiple time-steps. As we will see in Section V-D, V_mem has a significant contribution to the overall hardware overhead in SpikeFlow.

Mapping SNNs Onto SpikeFlow: SpikeSim employs the weight mapping strategy used in Neurosim [14] for partitioning and mapping the pretrained software SNN weights. Fig. 5(a), shows how a weight kernel of size $N \times M \times d \times d$ is partitioned on $X \times X$ crossbars. Here, $N$ and $M$ are the output and input channel dimensions, respectively and $d$ is the kernel dimension. The weight kernels are partitioned along the input channel ($N$) dimension and mapped along a crossbar column. While, the corresponding weights along the output channel ($M$) dimension are mapped on different crossbars. Hence, for a $d = 3$ kernel, 9 crossbars (C1-C9) are required. A similar partitioning is applied to the spike input map $S$ over the $M \times d \times d$ section. This mapping strategy maximizes input data reuse and minimizes the buffer access. The software weights are converted to IMC device conductance values using a linear mapping scheme [14], [16] (see Section IV-B and Fig. 9 for more details). To understand how the SNN is mapped across crossbars, PEs, and Tiles in the SpikeFlow architecture, let us consider a scenario in which an SNN consists of three convolutional layers in succession with input ($M$)/output ($N$) channels as follows: $64/64$, $64/128$, and $128/512$, kernel size $d=3$, and circuit parameters $X = 64$, $N_{PE} = 8$, and $N_C = 9$. In this case, layer-I requires $(64 \div 64) \times (64 \div 64) \times 3^2 = 9$ crossbars which require 1 PE to be mapped (or $PE_1 = 1$). In SpikeFlow, we adopt a design choice that an SNN layer can be mapped over multiple Tiles but multiple layers cannot be mapped in one Tile. Therefore, during mapping of layer I, the PEs are replicated over $N_{PE}$ PEs resulting in parallel mapping. The parallelization $Par_i$ for layer i is computed by $N_{PE} \div PE_i$. In the case of layer I, the parallel mapping $Par_1 = N_{PE} \div PE_1 = 8$. Similarly, the weights of layers II and III require on $PE_2 = 2$ and $PE_3 = 16$. Consequently, $Par_2 = 4$ and $Par_3 = 1$. In total, a total of four Tiles are required for mapping the three-layered SNN.

The DIFF Module: In general, a dual-crossbar approach [25] is employed to implement signed-MAC operations for ANNs. However, this approach is highly hardware-intensive, requiring additional energy expended on analog computations in the crossbars as well as the

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
energy-hungry ADCs. Furthermore, with the dual-crossbar approach, the total hardware area overhead of the PEs get doubled. An interesting by-product of the nonideality-aware weight encoding (Section IV-B) is that the dual-crossbar approach can be replaced with a lightweight, fully digital DIFF module shown in Fig. 5(b).

The DIFF module services the dot-product outputs from the Shift-and-Add circuit inside a Crossbar and performs accurate signed-MAC operations. The DIFF module contains SU number of 2-bit shift registers (SRs) as shown in Fig. 5(a). SU (the DIFF SpeedUp), dictates the number of parallel shift-registers, AND gates and the width of the adder tree for parallelly computing multiple rows of a column. Fig. 5(b) shows an example when the DIFF module has SU = 4 and the spike input V1 to crossbar C1 [shown in Fig. 5(a)] is mapped at the first bit location (i,j) in SR1–SR4. Here, c and j correspond to the cth column and jth row of crossbar C1. Similarly, the sign bits of the weights in the cth column and jth row is stored in the second location b_{cj}. bc_{cj} stores “1” for a negative-valued weight and a binary “0” for a positive-valued weight. Like ADC, and Shift-and-Add modules, the DIFF module is shared by multiple columns of the crossbar. For each crossbar column, the DIFF module requires X \div SU cycles. For example, in the case of a crossbar with size X=64, a DIFF with SU = 4 will complete the operations in X \div SU = 16 cycles for one column.

In each cycle, the spike input i_{cj} and the corresponding b_{cj} undergo AND operation followed by addition in the Adder Tree. The Adder Tree outputs the count of negative weights receiving a spike input in each cycle N_{neg}. The N_{neg} is accumulated over all the cycles resulting in N_{tot}. N_{tot} represents the total count of negative weights in a column receiving spike inputs (N_{tot}). N_{tot} scaled by 2^p (see Section IV-B for the details on p) is subtracted from the Shift-and-Add circuit output to obtain the signed-MAC output for the column. The scaling is performed by a p-bit left shift operation using the Barrel Shifter.

Fig. 6 shows that in our SpikeFlow architecture, the DIFF module results in 32x and 18x more energy-efficient MAC operations as opposed to the standard dual-crossbar approach for a single 64 \times 64 and 128 \times 128 crossbar-arrays, respectively. Here, we consider the total energy expended by the crossbar-array, input peripherals, multiplexer, ADC, and the shift-adders.

**Computation Cycle:** In SpikeFlow, crossbars are the fundamental MAC operation modules. First, the SL switch matrix converts the digital spike inputs to voltages (V_{read} for spike inputs and 0V for no spike case). Simultaneously, the WL switch matrix activates WL0-WLX, enabling parallel read from all the IMC devices in the crossbar. The input voltages and programmed IMC device conductance values undergo multiplication and accumulation by virtue of Ohm’s law and Kirchoff’s current law, respectively, leading to crossbar column currents along the bit-lines (BL0-BLX). The column currents are converted to digital partial sum values using flash ADCs.

The ADC outputs from each crossbar are fed to the Shift-and-Add circuit and successively to the DIFF module for obtaining the signed MAC output value. The DIFF module output is accumulated in the PA and stored in the PB. The outputs from PB are relayed to TA. Post accumulation, the outputs are relayed to TB. Every entry in the TB is appended with the respective destination tile’s address that maps the preceding layer. Next, the address appended values are pushed into the NoC router buffer which relays the TB outputs to the GA. The outputs from GA are fed directly to the LIF/IF Neuronal module for the nonlinear activation. Post activation, the spike data is transmitted over the NoC to the destination Tile.

**B. NICE: Nonideality Computation Engine**

Our NICE is designed in Python to compute the accuracy of SNNs mapped onto the SpikeFlow architecture. This computation engine captures the impact of hardware-level resistive crossbar nonidealities to generate MAC outputs during inference.

**Need for the NICE Simulator:** Prior works for ANNs, such as Neurosim [14] and CrossSim [15] that carry out crossbar-based inference of ANNs, neglect the inclusion of resistive parasitic nonidealities. A recent framework called RxNN [16], although takes into account resistive nonidealities, requires long simulation time to generate nonideal ANN weights from the ideal ones and performs input-independent modeling of nonidealities. To this effect, GenieX [20] captures input-dependent nonidealities for ANN inference, but follows an empirical approach by training an auxiliary fully connected network to model the crossbar nonidealities. This makes transferability of GenieX across different crossbar sizes difficult as the model requires retraining for every new crossbar size. To this end, NICE adopts a generalized circuit analysis-based method that incorporates input data dependency and makes it transferable across different SNN architectures and crossbar array sizes.
Algorithm 1 NICE’s Evaluation Flow

**Input:** Pre-trained SNN with \(N_l\) layers, weights \(W\), and \(T\) time-steps, Crossbar Size \(X\), ADC Precision \((h)\), \(R_{on}\), \(R_{off}\), Bits/Cell, Variation \(\sigma\), Wire resistance \(r\)

**Output** Hardware-Realistic Accuracy:

1: /* Before SNN Inference */
2: for \(i = 1\) to \(N_l\) do
3:    if “Conv” or “Linear” layer then
4:        \(p = \text{ceil}(\log_2(\min(|W_{ideal,i}|)))\) // Initialization
5:        \(W_{Enc} = W_{ideal,i}\) \(\triangleright\) Device Variations
6:        for \(w_j \in W_{Enc}\) do
7:            if \(w_j < 0\) then
8:                \(w_j = w_j + 2^p\) \(\triangleright\) NI-aware Encoding
9:            else
10:                \(w_j = w_j\)
11:        end if
12:    end for
13:    \(W_{Enc, Noisy} = W_{Enc} + \mathcal{N}(0, \sigma)\)
14: end for
15: Partition \(W_{Enc, Noisy}\) into crossbars of size \(X\).
16: Create coefficient matrix \(A_i\).
17: end if
18: end for
19:
20: /* During SNN Inference */
21: for \(t = 1\) to \(T\) do
22:    for \(i = 1\) to \(N_l\) do
23:        if “Conv” or “Linear” layer then
24:            Create \(B_{t,i}\) using the layer \(i\)’s inputs at \(t\).
25:            \(I \leftarrow \text{solution of system } A_iI = B_{t,i}\)
26:            \(MAC_u \leftarrow I\) converted to \(h\)-bit digital value.
27:        end if
28:        /* Inside DIFF Module */
29:        Compute \(N_{tot}\) for each crossbar column
30:        \(MACs = MAC_u - N_{tot} \times 2^p\)
31:    end for
32: end for
33: end for

**Nonideality-Aware Weight Encoding**

As shown in Algorithm 1, all pretrained weights of the convolution and linear layers undergo NI-aware encoding. For each layer \(i\), we first initialize \(W_{Enc}\) with software SNN weights \(W_{ideal,i}\). Next, \(2^p\) factor is added to \(W_{ideal,i}\) for NI-aware encoding (\(p\) is defined in line 5 of Algorithm 1). Note, the \(2^p\) factor will be later subtracted from the MAC outputs using the DIFF module. The NI-aware encoding encodes the negative weights to unsigned values, thereby increasing the number of 0s. This results in higher resistance values of the IMC devices mitigating the nonideal effects of interconnect parasitics in crossbars [40], [42]. The value of \(p\) is specific to layer \(i\)’s weight distribution. Fig. 7(left) illustrates an example of a 4-bit \((k = 4)\) quantized layer having weights \(W = \{-2, -1, 1, 2\}\) in its distribution. For a 4-bit quantization, \(p = 4\) represents the naive 2’s complement representation of the weights. As \(p\) reduces, the number of 0’s in the representation of \(-1\) and \(-2\) increases. For the given weight distribution, the maximum number of 0s are obtained for \(p = \text{ceil}(\log_2(\min(|W|))) = \text{ceil}(\log_2(1 - 2))\), i.e., at \(p = 1\).

Now, we describe the computation flow of NICE as follows has been illustrated in Fig. 8.

**Creating Coefficient Matrix (\(A_i\)):** After the NI-aware encoding, device conductance variation with distribution \(\mathcal{N}(0, \sigma)\) is added to the encoded weights \(W_{Enc}\) to create \(W_{Enc, Noisy}\). Next, \(W_{Enc, Noisy}\) is partitioned into analog crossbars of size \(X\) (see Section IV-A for details). Next, each crossbar column is converted to resistance ladders of size \(X\). Depending on the weight value, the device conductance values (or resistances \(R_{1} - R_{X}\)) are linearly mapped as shown in Fig. 9. To model the IR-drop nonideality, we incorporate column resistances \(r\) at each device node. For each resistance ladder a coefficient matrix \(A_{xy}\) (\(xth\) column of the \(yth\) crossbar) is generated. \(A_{i}\) is the stack of all the coefficient matrices \(A_{xy}\) for layer \(i\). Additionally, \(A_{i}\) is statically computed before the SNN inference begins.

**Generating the Constant Matrix \(B_{t,i}\) During SNN Inference:**

During SNN inference, the input spike map at time \(t\) and layer
i, $S_{r,i}$ is partitioned according to the crossbar size and sent to the rows of the crossbar $V_q$ (for the $q$th crossbar). The $V_q$ matrix contains the input voltage values ($V_{\text{read}} = v$ and 0 represents spike and no-spike, respectively). The $V_q$ matrix is used to create the matrix $B_{r,i}$. The output column currents ($I_{x}$) of all the crossbars are obtained by solving the system of linear equation $A_{x}I = B_{r,i}$. Here, $I$ represents the variable matrix of the linear equation system. The output column currents $I$ are converted to $h - \text{bit}$ digital values, where $h$ is the ADC precision.

Obtaining Signed MAC Outputs (MACs): Due to NI-aware encoding, the resulting MAC values obtained after ADC quantization are unsigned (MACu). To convert the unsigned outputs to signed values MACs, the DIFF module computes $N_{\text{tot}}$ for each crossbar column. As mentioned in Section IV-A, $N_{\text{tot}}$ is the number of negative weights in each crossbar column receiving a spike input. $N_{\text{tot}}$, scaled by $2^p$ ($p$ determined during NI-aware encoding) is then subtracted from MACu resulting in MACs. Multiplying the factor $2^p$ compensates for the $2^p$ factor added to the negative weights during NI-aware weight encoding. Note, that the given method of NI-aware weight encoding feasibly works due to the binary nature of spike inputs. Due to this, the value of $N_{\text{tot}}$ is merely the number of negative weights that received spike inputs. In the case when there are multibit inputs, the calculation of $S$ is not straightforward.

Fig. 10(a) shows that the crossbar currents obtained using a Cadence Virtuoso SPICE simulator and our NICE engine have very close resemblance with very small percent accuracy for crossbar sizes above $32 \times 32$. Here, the value of interconnect parasitic resistance is taken as $5\Omega$. Thus, NICE computes circuit-accurate currents that incorporate the impact of crossbar nonidealities during inference. Fig. 10(b) and (c) further corroborates the efficacy of NICE in capturing the crossbar nonideality. As expected, the MSE between the hardware and software convolution operations increase as the bit-line resistances $r$ and crossbar sizes increase. This has also been observed in prior literature [20].

C. ELA Engine

Our (ELA) Engine is designed in Python to compute the hardware-realistic energy, latency, and area for a SpikeFlow-mapped SNN model.

Inference Latency Evaluation: The latency of a particular layer $i$ (convolution or linear layer) has two components as shown in (3). First, the Tile Latency $\ell_i$ required for performing all the computations inside the Tile and second, the NoC Latency $\ell_i$ for performing all the communications in the particular layer

$$\ell_i = \text{Tile Latency}_i + \text{NoC Latency}_i.$$  \hfill (3)

Tile Latency $\ell_i$ can be computed using (4). Here, Cycles$_i$ denotes the number of clock cycles required for all the operations in the layer $i$ and Clock Period denotes the period of one clock cycle

$$\text{Tile Latency}_i = \text{Cycles}_i \times \text{Clock Period}.$$  \hfill (4)

Cycles$_i$ can be computed using (5), where $N_{\text{ops},i}$ denotes the number of operations (convolutions or linear operations) per output channel in layer $i$ and Cycles/Op is the number of clock cycles required for one convolution operation. In crossbar architectures, multiple PEs in a Tile execute in a parallel fashion to perform one or more operations. Therefore, Cycles/Op depends on the PE latency ($\alpha$) and the parallel mapping in layer $i$ (Par$_i$) as shown in (6)

$$\text{Cycles}_i = \text{Cycles/Op} \times N_{\text{ops},i}$$  \hfill (5)

$$\text{Cycles/Op} = \frac{\alpha}{\text{Par}_i}.$$  \hfill (6)

The PE Latency ($\alpha$) is defined as the number of clock cycles required for reception of spike-inputs by PEs, MAC computations inside the PEs by all NC crossbars, digital subtractions using the DIFF module, accumulation in PA and finally storage of the MAC outputs in PB. $\alpha$ has a fixed value depending on the number of crossbars inside PE (NC) and the SpeedUp of the DIFF module.

Pipelined Mode of Inference: The SpikeFlow-mapped SNN is executed in a pipelined dataflow. We define a layerscheduling factor which determines the percentage of total operations in layer $i$ that must be completed before layer $i + 1$ computations begin. Fig. 11 shows the pipelined execution of the 3-layered SNN discussed in Section IV-A. For illustration, let layers 1, 2, and 3 require 10, 8, and 6 convolution operations per output channel ($N_{\text{conv},1} = 10$, $N_{\text{conv},2} = 8$ and $N_{\text{conv},3} = 6$), respectively. Additionally, we will assume that the PE latency $\alpha = 8$ and the SNN requires only one time-step for this illustration. Note, that the SNN’s time-step is different from the Simulation Time-Stamps shown in Fig. 11. Based on the values of Par$_1$, Par$_2$, and Par$_3$, layers 1, 2, and 3 require ($\alpha/8$), ($\alpha/4$), and ($\alpha/1$) cycles per convolution, respectively. Due to layer scheduling of 25%, layer 2 begins after $0.25 \times N_{\text{conv},1} = 4$ convolutions of layer 1 and layer 3 begins after 2 convolutions of layer 2. The trace generator in the ELA engine tracks the number of active layers over all the simulation time stamps.
Fig. 11. Figure illustrating an example of SpikeFlow’s pipelined mode of inference.

The trace generator further records the Steady State during the SNN execution. Steady State is defined as the state at which the maximum number of SNN layers are active. The ELA engine estimates the optimal $V_{\text{mem}}$ cache size required for the LIF/IF neuronal module using the steady state information. As we will see in Section V-D, the steady state directly impacts the hardware area consumed by the LIF/IF Neuronal module.

Next, we add the NoC communication latency to the tile latency using the approach proposed in [17] to obtain the final hardware latency. The NoC latency is computed separately using a cycle-accurate NoC simulator [17], [46].

The number of data packets is computed using (8). Here, $A_i$ denotes the number of activations in layer $i$, $k_{\text{mem}}$ is the quantization of membrane potential value $U$ and NoC width denotes the NoC channel width.

$$\text{NoC latency}_i = N_{p,i} \times \text{Packet Latency} \quad (7)$$

$$N_{p,i} = \frac{A_i \times k_{\text{mem}}}{\text{NoC width}}. \quad (8)$$

Inference Energy and Area Evaluations: The hardware energy per inference and area estimations by the ELA Engine is straightforward. For a given set of SNN, circuit and device parameters, after the SNN is mapped on the SpikeFlow architecture, the number of Tiles, PEs, crossbars are determined. Further, the trace generator estimates the optimal $V_{\text{mem}}$ size of the neuronal module. The total hardware area is the sum of the area occupied by the individual components. Likewise, as the inference progresses on the SpikeFlow architecture, the total dynamic hardware energy expended is the sum of the dynamic energy consumed by the individual active components at a given point in time.

V. EXPERIMENTS AND RESULTS

A. Experimental Setup

Datasets: In this work, we benchmark pretrained SNN model using our Python-based SpikeSim tool. The inference is carried out using benchmark datasets, namely, CIFAR10, CIFAR100, and TinyImagenet. CIFAR10 and CIFAR100 datasets consist of RGB images (50000 training and 10000 testing) of size $32 \times 32$ belonging to 10 and 100 classes, respectively. The TinyImagenet dataset is a more complex dataset with RGB images (10000 training and 10000 testing) of size $64 \times 64$ belonging to 200 classes.

SpikeSim Parameters: The SNN Parameters for different datasets have been provided in Table III. All the SNNs are trained using back-propagation through time (BPTT) algorithm using the Adam optimizer with an initial learning rate of $1e^{-3}$. For the BPTT training, the inputs are direct rate encoded [37]. We use BPTT as it achieves higher performance with fewer time-steps compared to standard ANN-SNN conversion methods [31], [32], [33], [34], [35]. Although we use BPTT-based direct encoded models, SpikeSim can evaluate different pretrained SNN models irrespective of the training algorithm used. Unless stated otherwise, the circuit and device-specific parameters used in the experiments with SpikeSim have been listed in Table IV.

Hardware Evaluation Using NICE: For RRAM-based performance evaluation using NICE, we consider device programming noise and IR-drop noise. For SRAM-based evaluation, we only consider the IR-drop noise. Note, NICE can also analyze the impact of device read noise. However, from our analyses, we find that the IR-drop noise is much stronger compared to the device-level noises [44], [45].

Simulation Tools Used to Create the ELA Engine: The total energy, latency, and area of the SpikeFlow-mapped SNN is computed by the ELA engine using the energy, latency, and area values of individual hardware modules as explained in
Section IV-C. The energy, latency, and area of individual modules are extracted from simulation tools, such as Synopsys Design Compiler, Cadence Virtuoso, and BookSim [46]. For the digital modules (such as Accumulators, Buffers, PO, Shift-and-Add, LIF/IF neuronal and DIFF module, etc.), we use 65nm CMOS standard cell-based synthesis on Synopsys Design Compiler. The design of analog modules (e.g., IMC device, ADC etc.) are obtained from [14], [44], [45] and evaluated on Cadence Virtuoso platform using in-house circuit model designs. The analysis of communication modules are performed using the cycle-accurate BookSim simulator [46]. Given the bus width (32-bits), and the number of NoC routers (each tile is associated with one router), BookSim evaluates the overall energy, latency, and area based on primitives, such as the wire width, length, pitch, and the data transmission rate per cycle.

B. Hardware Realistic Performance Evaluation Using NICE

Fig. 12, shows that the NI-aware weight encoding significantly improves the performance of crossbar-mapped SNNs compared to vanilla encoding. Vanilla encoding on SpikeSim leads to SNN accuracy degradation to random values of $\sim 11\%$, $\sim 2\%$, and $\sim 1\%$ for CIFAR10, CIFAR100, and TinyImagenet datasets, respectively. This is attributed to error accumulation across multiple time-steps owing to resistive crossbar nonidealities [26]. With our novel NI-aware weight encoding scheme, we find that the inference accuracy of VGG9/CIFAR10, VGG16/CIFAR100 and VGG16/TinyImagenet SNNs improve more than 70%, 50% and 40%, respectively, compared to the vanilla encoding due to the reduction in crossbar nonideal effects.

C. Area and Energy Distributions

Figs. 13 and 14 show the componentwise area and energy distribution for SNNs with VGG9/CIFAR10, VGG16/CIFAR100 and VGG16/TinyImagenet network topology. All results correspond to $64 \times 64$ crossbar mapping with RRAM IMC device having 1-bit/Cell. Here, Crossbar includes the IMC crossbar array, multiplexer, ADC, shift-adder and input peripherals. The digital peripherals consist of PA, PB, PE Input Buffer, TA, TB, Tile Input Buffer, GB, GA, and PO modules shown in Fig. 3. It can be observed that the area and energy values increase with larger network sizes as deeper networks require more tiles and more computations. Furthermore, for the same SNN topology (VGG16), the neuronal module area and the total energy increase by $> 6 \times$ and $> 4 \times$ with increase in the input feature dimension from 32 (CIFAR100) to 64 (TinyImagenet). This is because of a higher number of convolution operations per SNN layer.

D. Neuronal Area Overhead and Mitigation

From Fig. 13, we can observe that the total neuronal area accounts for a significant percentage share in the overall chip area. This share is as large as 24% (1.2 mm$^2$) for the VGG9 SNN, which is almost equal the total chip area consumed by the PE and the DIFF units. For a VGG16 SNN inferred using a more complex dataset like TinyImagenet with input dimensions = 64, the absolute area encompassed by the neuronal unit becomes as large as 9.32 mm$^2$. This observation is because the LIF/IF neuron module contains a large $V_m$ SRAM cache to store the intermediate membrane potentials over multiple time steps (see Fig. 4). Due to the pipelined dataflow, multiple layers can be active at the steady state and thus the LIF/IF $V_m$ cache size must be large enough to store the membrane potentials of the active layer neurons. To this end, we propose mitigation strategies to ameliorate the LIF/IF neuron module’s area overhead.

In Fig. 15, we analyze the tradeoffs among three different dataflow strategies for the VGG9/CIFAR10 SNN mapped on a $64 \times 64$ RRAM crossbar. Case-I refers to SpikeFlow’s standard pipelined dataflow with scheduling factor (explained in Section IV-C) of 25% for each layer. The number of active layers in the steady state are 3. In Case-II, we analyze the effect of the tick-batched dataflow introduced in [6]. In tick-batching, layers are processed sequentially and all the time-steps for a specific layer are computed together before proceeding to the next layer (scheduling factor = 100%). This leads to a $2 \times$ reduction in the total neuronal area. However, tick-batching is highly impractical for crossbar-mapped SNNs as the inference latency increases $4 \times$. Furthermore, the average value of $GOPS/\mu m^2$ in the steady state is reduced by $5 \times$ due to significant hardware area-underutilization. In Case-III, we adopt a layer-specific scheduling dataflow wherein, we heuristically assign different scheduling factors for each convolutional layer to reduce the number of active layers in the steady state. Here, the typical scheduling factors for different layers range between 25% and 75%. Due to this, the number of active layers is reduced to 2. We find that the neuronal area reduces by $1.5 \times$ with a marginal increase in inference latency by $1.17 \times$ and reduction in the average $GOPS/\mu m^2$ by $1.38 \times$. Although the layer-specific scheduling method reduces the neuronal area overhead it inevitably entails tradeoffs between the neuronal area overhead, inference latency and hardware underutilization. However, our objective is to minimize both neuronal area as well as inference latency. To this end, we propose SNN topological modifications guided by hardware evaluations using SpikeSim to achieve higher area and compute efficiency.

Fig. 16(a) presents how the total hardware EDP per inference varies across different convolutional layers of the VGG9/CIFAR10 SNN mapped on $64 \times 64$ RRAM crossbar. We observe that the first convolution layer incurs the highest EDP in comparison to the other layers. This is because the first layer incurs higher convolution operations due to
Fig. 13. Pie-chart showing componentwise area distribution for (a) VGG9/CIFAR10 ($T = 5$), (b) VGG16/CIFAR100 ($T = 10$), and (c) VGG16/TinyImagenet ($T = 10$) SNNs. All SNNs are evaluated on the SpikeFlow architecture with $64 \times 64$ crossbars.

Fig. 14. Pie-chart showing the componentwise inference energy expended by (a) VGG9/CIFAR10 ($T = 5$), (b) VGG16/CIFAR100 ($T = 10$), and (c) VGG16/TinyImagenet ($T = 10$) SNNs when evaluated on a SpikeFlow architecture with $64 \times 64$ crossbars.

TABLE V

| SNN topology    | Time-steps ($T$) | SW accuracy (%) | Accuracy using NCU (%) | Energy ($\mu$J) | Area (mm$^2$) | Latency (ns) |
|-----------------|------------------|-----------------|------------------------|-----------------|--------------|--------------|
| VGG9/CIFAR10    | 5                | 93.17           | 93.17                  | 90.93           | 10.1         | 3.08         |
| VGG16/CIFAR100  | 10               | 95.36           | 95.36                  | 95.36           | 16.88        | 34.36        |

Fig. 15. Bar-chart showing the tradeoff between neuronal area, total inference latency and average GOPS/μm$^2$ in the steady state for a VGG9/CIFAR10 SNN inferred on a SpikeFlow for different dataflows. The y-axis denotes Latency (ns), Neuronal Area (mm$^2$), and average GOPS/μm$^2$ values.

Fig. 16(c) presents the variation in neuronal area, neuronal energy and overall inference latency with respect to the number of output channels in the first layer. Overall with 8 output channels, the neuronal area overhead is reduced by $1.24 \times$ along with a $1.15 \times$ reduction in the overall inference latency. Consequently, the contribution of the neuronal area to the overall chip area is reduced to 20% from 24%. Furthermore, due to channel scaling, we observe a 26% reduction in the energy consumption of the neuronal module and other components as seen in Fig. 17.

Fig. 18 shows a holistic EDP evaluation for VGG9/CIFAR10 [Fig. 18(a)], VGG16/CIFAR100 [Fig. 18(b)] and VGG16/TinyImagenet [Fig. 18(c)] SNNs implemented on $64 \times 64$, $128 \times 128$ and $256 \times 256$ analog crossbar arrays. Here, VGGX-N-M denotes a VGGX SNN model with N output channels in the conv1 layer and M $\times$ M input feature dimensions. The results further corroborate that channel scaling in the first layer improves the energy efficiency of crossbar-mapped SNNs. Under similar SNN topology, mapping on a $256 \times 256$-sized crossbar reduces the EDP by $4 \times$ compared to a $64 \times 64$ crossbar mapping. This is because, larger crossbars incur lower latency due to higher parallel MAC operations. Similarly, for the same crossbar size, input dimension scaling (16 from 32 in the case of CIFAR10/CIFAR100 and 32 from 64 in the case of...
are based on 65-nm CMOS technology on a 64 × 64 SRAM crossbar array. Both the SNN and ANN are quantized to 4-bit precision. In addition, no channel or input dimension scaling is applied to the SNNs. Note, since the ANN and SNN is implemented on two different platforms all results are normalized with respect to Neurosim ANN implementation to achieve a fair comparison.

As seen in Fig. 19(a), the neuronal module’s area for an SNN (the LIF/IF Neuronal Module) is around ∼100× compared to ANN’s ReLU Neuronal Module. This is because of the large SRAM cache to store membrane potentials ($V_{\text{mem}}$) across different time-steps. Fig. 19(b) shows that the crossbar area required by ANN is 1.8× higher compared to SNNs. This is primarily because in SpikeSim, we replace the dual-crossbar approach used in ANNs for signed MAC operations with the DIFF module. Further, due to the binary spike data, SNN crossbars do not require Shift-and-Add circuits for input bit-serialization, and therefore consume 10% lesser area compared to ANN crossbars. Finally, Neurosim uses H-Tree for all data communications both local and global to the tiles. In contrast, SpikeSim employs NoC and H-Tree interconnects for intertile and intratile communications, respectively. This reduces the data communication area in SpikeSim by > 3× compared to ANNs as seen in Fig. 19(c).

From Fig. 20, we find that at sufficiently lower time-steps ($T = 3, 4$), SNNs can achieve iso-performance while being 1.3× energy-efficient and 1.1× faster compared to a 4-bit quantized ANN. This is because ANN crossbar architectures adopt input bit-serialization to encode multibit activation requiring multiple crossbar read cycles and hence higher computation and communication overheads. In contrast, SNNs require one read cycle over multiple time-steps to perform the MAC operations. If the number of time-steps are low enough, SNNs achieve higher compute efficiency.

Therefore, lower number of time-steps will be advantageous to SNNs in terms of energy and latency compared to ANNs. The current state-of-the-art SNN training methods require larger number of time-steps in order to achieve higher performance especially for complex datasets, such as CIFAR100 and TinyImagenet. It is therefore critical for future research works to focus on achieving higher performance at lower time-step overhead for SNNs to be practically energy efficient.

VI. SUMMARY OF BENCHMARKING RESULTS USING SPIKE SIM

In Table V, we provide holistic inference and hardware performance results using the SpikeSim tool for different SNN models trained on the CIFAR10 and CIFAR100 datasets. The results are shown for mappings on 64 × 64 crossbars. First, the SNN inference accuracy upon crossbar-mapping reduces from its software inference accuracy by ∼9% and ∼13% for 4-bit SRAM and 1-bit RRAM IMC units, respectively. It must be recalled that due to the inherent vulnerability of SNNs toward crossbar nonidealities [26], the inference accuracy of SNNs on SpikeSim (without NI-aware weight encoding) drops down to ∼10% from the corresponding software accuracy. Due to NICE’s novel weight encoding method, the classification performance of SNNs is restored to a suitable range as shown in Table V. Additionally, SNNs with topological modifications and input dimension scaling (VGGX-8-16) achieve 1.15× lower energy, 1.03−1.1× lower area, and 3.72−6.27× lower latency compared to the standard SNN models.

VII. COMPARISON BETWEEN ANN AND SNN

In Fig. 19, we perform a componentwise area comparison between a crossbar-mapped VGG9/CIFAR10 ANN and SNN. The ANN is mapped on the Neurosim [14] platform while the SNN is mapped on the SpikeSim platform. Both the mappings
Fig. 18. Bar-chart showing EDP values in logarithmic scale for SNNs with standard- (a) VGG9/CIFAR10 ($T = 5$), (b) VGG16/CIFAR100 ($T = 10$), (c) VGG16/TinyImagenet ($T = 10$) and models with channel and input dimension scaling across different crossbars.

Fig. 19. Comparison for (a) neuronal module area, (b) crossbar area, and (c) data communication area in Neurosim-based ANN and SpikeSim-based SNN implementations.

Fig. 20. Comparing the (a) latency, (b) energy, and (c) software accuracy for an ANN across different time-steps. The latency and energy correspond to 4-bit SNN and are normalized with respect to a 4-bit quantized ANN baseline.

VIII. CONCLUSION

This work presents SpikeSim, an end-to-end evaluation tool for hardware realistic inference performance (accuracy), energy, latency, and area of crossbar-mapped SNNs. One of the important findings of this work involves high area overhead of the neuronal module in SNNs due to temporal storage of membrane potentials for LIF/IF neuron functionality. To this end, we employ SNN topological modifications, such as channel scaling, input scaling, and layer-specific scheduling strategies to overcome the neuronal module’s area. Based on our study, we recommend the following design strategies for future SNN-IMC-architecture co-design research. 1) Performing hardware-aware SNN architecture design to achieve low neuronal overhead and high crossbar utilization. 2) Designing IMC-architecture-specific dataflows to reduce number of active layers and thereby reduce the neuronal area overhead. 3) Exploiting the spatiotemporal properties in SNNs to achieve higher performance at lower time-steps in order to achieve superior energy efficiency and throughput compared to ANNs.

Finally, it is noteworthy to mention that while SpikeSim’s scope analyzes different nuances of implementing SNNs on realistic IMC hardware, it also motivates future research toward analysing macro-level dataflow paradigms, such as different communication topologies (e.g., star, torus [46]) and understanding latency/energy under highly area constrained scenarios with resource contention.

REFERENCES
[1] K. Roy et al., “Towards spike-based machine intelligence with neuromorphic computing,” Nature, vol. 575, pp. 607–617, Nov. 2019.

[2] E. Strobtias et al., “An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data,” Front. Neurosci., vol. 11, p. 350, Jun. 2017.

[3] Y. Kim and P. Panda, “Optimizing deeper spiking neural networks for dynamic vision sensing,” Neural Netw., vol. 144, pp. 686–698, Dec. 2021.

[4] M. Davies et al., “Loihi: A neuromorphic manycore processor with on-chip learning,” IEEE Micro, vol. 38, no. 1, pp. 82–99, Jan./Feb. 2018.

[5] F. Akopyan et al., “TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 34, no. 10, pp. 1537–1557, Oct. 2015.

[6] S. Narayanan et al., “SpinalFlow: An architecture and dataflow tailored for spiking neural networks,” in Proc. ISCA, 2020, pp. 349–362.

[7] J.-J. Lee et al., “Parallel time batching: Syntactic-array acceleration of sparse spiking neural computation,” in Proc. HPCA, 2022, pp. 317–330.

[8] K.-H. Kim et al., “A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications,” Nano Lett., vol. 12, no. 1, pp. 389–395, 2012.

[9] A. Ankiet al., “RESPARC: A reconﬁgurable and energy-efﬁcient architecture with memristive crossbars for deep spiking neural networks,” in Proc. DAC, 2017, pp. 1–6.

[10] L. Ni et al., “An energy-efﬁcient and high-throughput bitwise CNN on sneak-path-free digital ReRAM crossbar,” in Proc. ISLPED, 2017, pp. 1–6.

[11] I. Chakraborty et al., “Pathways to efﬁcient neuromorphic computing with non-volatile memory technologies,” Appl. Phys. Rev., vol. 7, Jun. 2020, Art. no. 21508.

[12] A. Ankiet al., “PUMA: A programmable ultra-efﬁcient memristor-based accelerator for machine learning inference,” in Proc. ASPLOS, 2019, pp. 715–731.

[13] Y.-H. Chen et al., “Eyeriss: An energy-efﬁcient reconﬁgurable accelerator for deep convolutional neural networks,” IEEE J. Solid-State Circuits, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[14] P.-Y. Chen et al., “NeuroSim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 37, no. 12, pp. 3067–3080, Dec. 2018.

[15] P. T. Nguyen et al., “CrossSim: Exploiting mutual relationships to detect similar OSS projects,” in Proc. 44th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA), 2018, pp. 388–395.

[16] S. Jain et al., “RxNN: A framework for evaluating deep neural networks on resistive crossbars,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 40, no. 2, pp. 326–338, Feb. 2021.

[17] G. Krishnan et al., “SIAM: Chiplet-based scalable in-memory accelerator with mesh for deep neural networks,” ACM Trans. Embedded Comput. Syst., vol. 20, no. 55, p. 68, 2021.

[18] L. Liang et al., “H2Learn: High-efﬁciency learning accelerator for high-accuracy spiking neural networks,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 41, no. 11, pp. 4782–4796, Nov. 2022.

[19] R. Yin et al., “SATA: Sparsity-aware training accelerator for spiking neural networks,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., early access, Oct. 10, 2022, doi: 10.1109/TCAD.2022.3213211.

[20] I. Chakraborty et al., “GENIEx: A generalized approach to emulating non-idealities in memristive Xbars using neural networks,” in Proc. DAC, 2020, pp. 1–6.

[21] G. Krishnan et al., “Robust RRAM-based in-memory computing in light of model stability,” in Proc. IRPS, 2021, pp. 1–5.

[22] G. Krishnan et al., “Exploring model stability of deep neural networks for reliable RRAM-based in-memory acceleration,” IEEE Trans. Comput., vol. 71, no. 11, pp. 2740–2752, Nov. 2022.

[23] Z. Wang et al., “Fully memristive neural networks for pattern classiﬁcation with unsupervised learning,” Nat. Electron., vol. 1, pp. 137–145, Feb. 2018.

[24] X. Zhang et al., “Fully memristive SNNs with temporal coding for fast and low-power edge computing,” in Proc. IEDM, San Francisco, CA, USA, 2020, pp. 29.6.1–29.6.4.
[25] S. N. Truong and K.-S. Min, “New memristor-based crossbar array architecture with 50%-area reduction and 48%-power saving for matrix-vector multiplication of analog neuromorphic computing,” J. Semicond. Technol. Sci., vol. 14, no. 3, pp. 356–363, 2014.

[26] A. Bhattacharjee et al., “Examining the robustness of spiking neural networks on non-ideal memristive crossbars,” in Proc. ISLPED, 2022, pp. 1–6.

[27] A. Krizhevsky et al., “Learning multiple layers of features from tiny images,” Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009-09, 2009.

[28] Y. Le and X. Yang, “Tiny ImageNet visual recognition challenge,” Course Note CS 231N, Stanford Univ., Stanford, CA, USA, 2015.

[29] A. Shafiee et al., “ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars,” ACM SIGARCH Comput. Archit. News, vol. 44, no. 3, pp. 14–26, 2016.

[30] P. U. Diehl and M. Cook, “Unsupervised learning of digital recognition using spike-timing-dependent plasticity,” Front. Comput. Neurosci., vol. 9, p. 95, Aug. 2015.

[31] A. Sengupta et al., “Going deeper in spiking neural networks: VGG and residual architectures,” Front. Neurosci., vol. 13, p. 95, Mar. 2019.

[32] P. U. Diehl et al., “Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing,” in Proc. IJCNN, 2015, pp. 1–8.

[33] B. Han and K. Roy, “Deep spiking neural network: Energy efficiency through time based coding,” in Proc. 16th Eur. Conf. Comput. Vis., Glasgow, U.K., Aug. 2020, pp. 388–404.

[34] Y. Li et al., “A free lunch from ANN: Towards efficient, accurate spiking neural networks calibration,” 2021, arXiv:2106.06984.

[35] B. Rueckauer et al., “Conversion of continuous-valued deep networks to efficient event-driven networks for image classification,” Front. Neurosci., vol. 11, p. 682, Dec. 2017.

[36] Y. Wu et al., “Spatio-temporal backpropagation for training high-performance spiking neural networks,” Front. Neurosci., vol. 12, p. 331, May 2018.

[37] Y. Kim et al., “Rate coding or direct coding: Which one is better for accurate, robust, and energy-efficient spiking neural networks?” in Proc. ICASSP, 2022, pp. 71–75.

[38] Y. Kim and P. Panda, “Revisiting batch normalization for training low-latency deep spiking neural networks from scratch,” Front. Neurosci., vol. 15, Dec. 2021, Art. no. 773954.

[39] B. Liu et al., “Vortex: Variation-aware training for memristor X-bar,” in Proc. DAC, 2015, pp. 1–6.

[40] A. Bhattacharjee et al., “NEAT: Non-linearity aware training for accurate, energy-efficient and robust implementation of neural networks on 1T-1R crossbars,” IEEE Trans. Comput. Aided Design Integr. Circuits Syst., vol. 41, no. 8, pp. 2625–2637, Aug. 2022.

[41] A. Bhattacharjee et al., “Efficiency-driven hardware optimization for adversarially robust neural networks,” in Proc. DATE, 2021, pp. 884–889.

[42] A. Bhattacharjee and P. Panda, “SwitchX: Gmin-Gmax switching for energy-efficient and robust implementation of binary neural networks on ReRAM Xbars.” ACM Trans. Design Autom. Electron. Syst., to be published.

[43] S. K. Roy et al., “Fundamental limits on the computational accuracy of resistive crossbar-based in-memory architectures,” in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), Austin, TX, USA, 2022, pp. 384–388.

[44] A. Jaiswal et al., “ST SRAM cell as a multibit dot-product engine for 8T SRAM cell as a multibit dot-product engine for beyond von Neumann computing,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 27, no. 11, pp. 2556–2567, Nov. 2019.

[45] B. Hajri et al., “RRAM device models: A comparative analysis with experimental validation,” IEEE Access, vol. 7, pp. 168963–168980, 2019.

[46] N. Jiang et al., “A detailed and flexible cycle-accurate network-on-chip simulator,” in Proc. ISPASS, 2013, pp. 86–96.

[47] Priyadarshini Panda (Member, IEEE) received the B.E. degree in electronics and communication engineering and the M.Sc. degree in physics from B.I.T. S. Pilani, India, in 2013, and the Ph.D. degree from Purdue University, West Lafayette, IN, USA, in 2019. She joined Yale University, New Haven, CT, USA, as an Assistant Professor with the Electrical Engineering Department in August 2019. From 2013 to 2014, she worked with Intel, Bangalore, India, on RTL design for graphics power management. She has also worked with Intel Labs, Hillsboro, OR, USA, in 2017, and Nvidia, Bengaluru, in 2013, as a Research Intern. During her internship with Intel Labs, she developed large-scale spiking neural network algorithms for benchmarking the Loihi chip. Her research interests lie in neuromorphic computing, spiking neural networks, energy-efficient accelerators, and compute in-memory processing.

Abhishek Moitra (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with the Intelligent Computing Lab, Yale University, New Haven, CT, USA. His research work has been published in reputed journals, such as the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I: REGULAR PAPERS and IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, and conferences such as DAC. His research interests involve hardware-algorithm co-design and co-exploration for designing robust and energy-efficient hardware architectures for deep learning tasks.

Dr. Cao was a recipient of the 2020 Intel Outstanding Researcher Award, the 2009 ACM SIGDA Outstanding New Faculty Award, the 2006 NSF CAREER Award, the 2006 and 2007 IBM Faculty Award, and five Best Paper Awards. He is a Distinguished Lecturer of the IEEE Circuits and Systems Society.

Abhiroop Bhattacharjee (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with the Intelligent Computing Lab, Yale University, New Haven, CT, USA. His research interests lie in the area of algorithm-hardware co-design of on-chip memory architectures for deep learning and neuromorphic applications. His research has been published in reputed journals and conferences, such as the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, ACM and ISLPED.

Dr. Panda is the recipient of the 2019 Amazon Research Award, the 2022 Google Research Scholar Award, the 2022 DARPA Riser Award, the 2023 ISLPED Best Paper Award, and the 2023 NSF CAREER Award. She was the recipient of the Outstanding Student Award in physics for academic excellence.