An In-Memory Analog Computing Co-Processor for Energy-Efficient CNN Inference on Mobile Devices

Mohammed Elbtity[1], Abhishek Singh[2], Brendan Reidy[1], Xiaochen Guo[2], Ramtin Zand[1]
[1]Department of Computer Science and Engineering, University of South Carolina, Columbia, SC 29208, USA
[2]Department of Electrical and Computer Engineering, Lehigh University, Bethlehem, PA 18015, USA

***Abstract*—In this paper, we develop an in-memory analog computing (IMAC) architecture realizing both synaptic behavior and activation functions within non-volatile memory arrays. Spin-orbit torque magnetoresistive random-access memory (SOT-MRAM) devices are leveraged to realize sigmoidal neurons as well as binarized synapses. First, it is shown the proposed IMAC architecture can be utilized to realize a multilayer perceptron (MLP) classifier achieving orders of magnitude performance improvement compared to previous mixed-signal and digital implementations. Next, a heterogeneous mixed-signal and mixed-precision CPU-IMAC architecture is proposed for convolutional neural networks (CNNs) inference on mobile processors, in which IMAC is designed as a co-processor to realize fully-connected (FC) layers whereas convolution layers are executed in CPU. Architecture-level analytical models are developed to evaluate the performance and energy consumption of the CPU-IMAC architecture. Simulation results exhibit 6.5% and 10% energy savings for CPU-IMAC based realizations of LeNet and VGG CNN models, for MNIST and CIFAR-10 pattern recognition tasks, respectively.**

***Index Terms*—in-memory computing, magnetic random access memory (MRAM), convolutional neural networks (CNNs), mixed-precision and mixed-signal inference.**

I. Introduction

Deep learning algorithms are playing an important role in pursuing safer self-driving cars, smarter robots, smartphone applications, etc., which are typically running on mobile computing devices. One of the major limitations of implementing deep learning models on these mobile computing devices is their limited computing power and severe energy constraints. Machine learning (ML) applications such as image and speech recognition are known to be more data-centric tasks, in which most of the energy and time is consumed in data movement rather than computation [1]. As alternatives to von Neumann architectures, in-memory computing (IMC) and near-memory computing (NMC) architectures aim to address these issues through performing processing within or near storage devices, respectively. Various approaches have been proposed in recent years to achieve this goal, from 3D integration technology [2] to emerging nonvolatile resistive memory devices, which can store information in their conductance states [1], [3]. Some of the resistive memory technologies that have been used to realize IMC systems are resistive random-access memory (ReRAM) [4], [5], phase-change memory (PCM) [6], and magnetoresistive random-access memory (MRAM) [7].

A wide range of the previous memristive-based IMC and NMC schemes operate in the digital domain [7], meaning that they leverage resistive memory crossbars to implement Boolean logic operations such as XNOR/XOR within memory subarrays, which can be utilized to implement multiplication operation in binarized neural networks [8]. While digital IMC approaches provide important energy and area benefits, they are not fully leveraging the true potential of resistive memory devices that can be realized in the analog domain. Mixed-signal analog/digital IMC architectures, such as the recently-proposed AiMC [9], leverage the resistive memory crossbars to compute multiply and accumulation (MAC) operation in O(1) time complexity using various physical mechanisms such as Ohm's law and Kirchhoff's law in electrical circuits. Here, we use MRAM technology to develop analog neurons as well as synapses to form an in-memory analog computing (IMAC) architecture that can compute both MACs and activation functions within an MRAM array. This enables maintaining the computation in the analog domain while processing and transferring data from one layer to another layer in fully connected (FC) classifiers. Despite their performance and energy benefits, the low precision computation associated with analog IMC architectures is prohibitive for many practical mobile computing applications which require large scale deep learning models [3]. Thus, alternative solutions are sought to integrate the energy-efficient but low-precision IMAC architecture with high-precision mobile CPUs. In this work, we will conduct algorithm- and architecture-level innovations to design and simulate a heterogeneous mixed-precision and mixed-signal CPU-IMAC mobile processor achieving low-energy and high-performance inference for deep convolutional neural networks (CNNs) without compromising their accuracy.

II. Background

A. Fundamentals of SOT-MRAMs

We use spin-orbit torque (SOT) MRAM devices [10] as the building block for our proposed IMAC architecture. The SOT-MRAM cell includes a magnetic tunnel junction (MTJ) with two ferromagnetic (FM) layers separated by a thin oxide layer. MTJ has two different resistance levels that are determined based on the angle (θ) between the magnetization orientation of the FM layers. The resistance of the MTJ in parallel (P) and antiparallel (AP) magnetization configurations can be obtained using the following equations [11]:

TABLE I
PARAMETERS OF THE SHE-MRAM DEVICE [11].

Parameter	Description	Value
MTJ_{Area}	$l_{MTJ}.w_{MTJ}.\pi/4$	$50nm \times 30nm \times \pi/4$
HM_V	$l_{HM}.w_{HM}.t_{HM}$	$100nm \times 50nm \times 3nm$
RA	RA	$10\ \Omega.\mu m^2$
TMR_0	TMR	200
V_0	Fitting Parameter	0.65

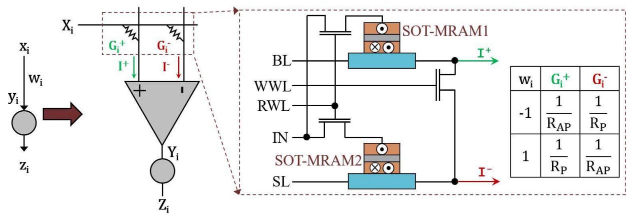

Fig. 1. The SOT-MRAM based binary synapse.

$$
\begin{aligned}
R(\theta) &= \frac{2R_{MTJ}(1+TMR)}{2+TMR(1+\cos\theta)} \\
&= \begin{cases} R_P = R_{MTJ}, & \theta = 0 \\ R_{AP} = R_{MTJ}(1+TMR), & \theta = \pi \end{cases}
\end{aligned} \tag{1}
$$

$$
TMR = \frac{TMR_0/100}{1+(\frac{V_b}{V_0})^2} \tag{2}
$$

where $R_{MTJ} = \frac{RA}{Area}$, in which RA is the resistance-area product value. TMR is the tunneling magnetoresistance that is a function of bias voltage (V_b). V_0 is a fitting parameter, and TMR_0 is a material-dependent constant. In MTJ, the magnetization direction of electrons in one of the FM layers is fixed (pinned layer), while those of the other FM layer (free layer) can be switched. In [10], it is shown that passing a charge current through a heavy metal (HM) generates a spin-polarized current using the spin Hall Effect (SHE), which can switch the magnetization direction of the free layer. The ratio of the generated spin current to the applied charge current is normally greater than one leading to an energy-efficient switching operation [12]. Herein, we use (1) and (2) to develop an SOT-MRAM device model using the parameters listed in Table I [11]. The SOT-MRAM model is used along with the 14nm HP-FinFET PTM library to implement the neuron and synapse circuits as described in the following.

B. SOT-MRAM Based Synapse

Resistive devices have been broadly studied to be used as weighted connections between neurons in neural networks. Fig. 1 shows a neuron with $Y_i = X_i \times W_i$ as its input, in which X_i is the input signal and W_i is a binarized weight. The corresponding circuit implementation is also shown in the figure, which includes two SOT-MRAM cells and a differential amplifier as the synapse. The output of the differential amplifier (Y_i) is proportional to $(I^+ - I^-)$, where $I^+ = X_iG_i^+$ and $I^- = X_iG_i^-$. Thus, $Y_i \propto X_i(G_i^+ - G_i^-)$, in which G_i^+ and G_i^- are the conductance of SOT-MRAM1 and SOT-MRAM2, respectively. The conductance of SOT-MRAMs can be adjusted to realize negative and positive weights in a binary synapse. For instance, for $\mathbf{W}_i = -1$, SOT-MRAM1 and SOT-MRAM2 should be in AP and P states, respectively, $R_{AP} > R_P$, which means $G_{AP} < G_P$ since $G = 1/R$, therefore $G_i^+ < G_i^-$ and $Y_i < 0$.

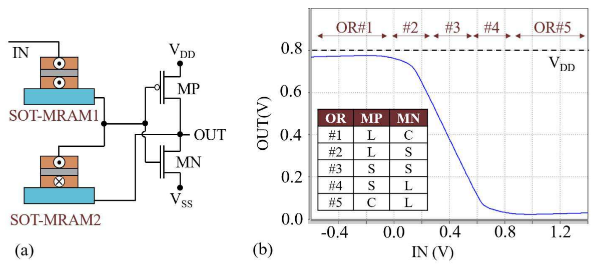

Fig. 2. (a) The SOT-MRAM based neuron, (b) The VTC curves showing various operating regions (ORs) of PMOS (MP) and NMOS (MN) transistors, in which *L*, *S*, and *C* represent linear, saturation, and cutoff regions, respectively.

TABLE II
COMPARISON OF VARIOUS ANALOG SIGMOIDAL NEURONS.

	[13]	[14]	Proposed Herein
Normalized Power Consumption	7.4×	0.98×	1×
Normalized Area Consumption	10×	12.3×	1×
Normalized Power-Area Product	74×	12×	1×

III. PROPOSED SOT-MRAM BASED NEURON

Here, we propose an analog sigmoidal neuron, which includes two SOT-MRAM devices and a CMOS-based inverter, as shown in Fig. 2 (a). The magnetization configurations of SOT-MRAM1 and SOT-MRAM2 devices should be in P and AP states, respectively. The SOT-MRAMs in the neuron's circuit create a voltage divider, which reduces the slope of the linear operating region in the inverter's voltage transfer characteristic (VTC) curve. The reduction in the slope of the linear region in the CMOS inverter creates a smooth high-to-low output voltage transition, which enables the realization of a $sigmoid$ activation function. Fig. 2 (b) shows the SPICE circuit simulation results of the proposed SOT-MRAM based neuron using $V_{DD} = 0.8V$ and $V_{SS} = 0V$. The results verify that the neuron can approximate a $sigmoid(-x)$ activation function that is biased around $b = \frac{1}{2}(V_{DD} - V_{SS})$ voltage. The non-zero bias voltage can be canceled at both circuit- and algorithm-level.

Table II provides a comparison between our SOT-MRAM based sigmoidal neuron and previous power- and area-efficient analog neurons [13], [14]. The SPICE circuit simulation results obtained show an average power consumption of $64\mu W$ for the SOT-MRAM based sigmoid neuron. Moreover, the layout design of the proposed neuron circuit shows an area occupation of $13\lambda \times 30\lambda$, in which λ is a technology-dependent parameter. Herein, we used the 14nm FinFET technology, which leads to the approximate area occupation of $0.02\mu m^2$. To provide a fair comparison in terms of area and power dissipation, we utilized the general scaling method [15] to normalize the power dissipation and area of the designs listed in Table II. Comparison results indicate that the proposed SOT-MRAM-based neuron achieves significant area reduction while realizing comparable power consumption compared to the previous analog neuron

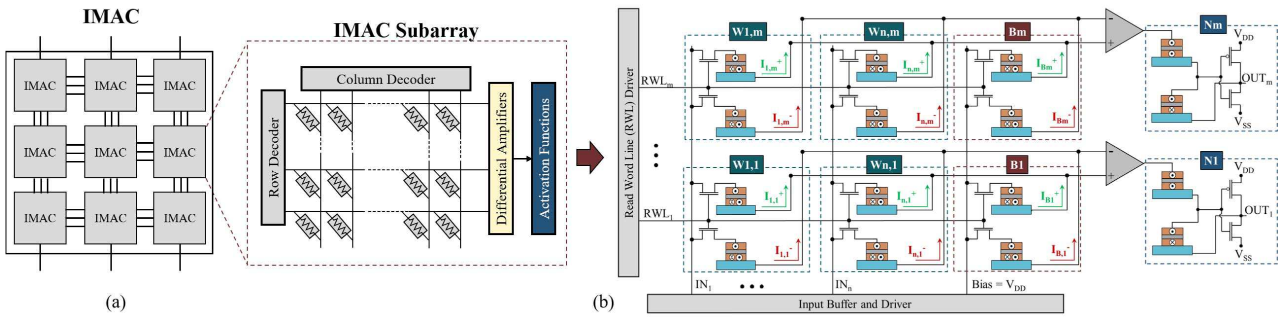

Fig. 3. (a) IMAC architecture, (b) The read path of an $n \times m$ IMAC subarray.

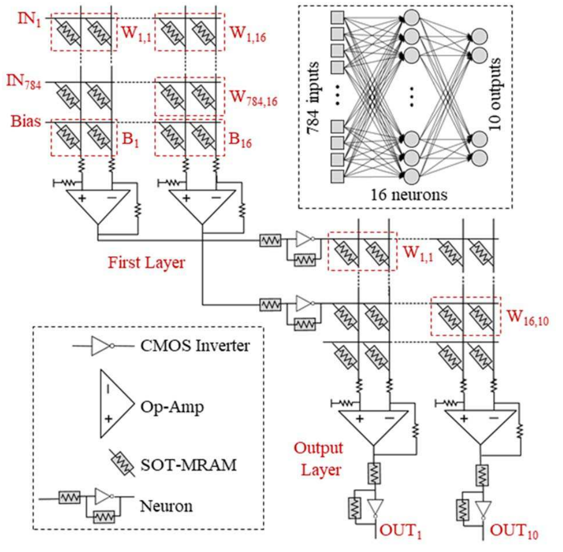

Fig. 4. The $784 \times 16 \times 10$ SOT-MRAM based MLP circuit.

implementations. This leads to a $74\times$ and $12\times$ reduction in power-area product compared to the designs introduced in [13] and [14], respectively. Moreover, our proposed implementation is compatible with SOT-MRAM synapses, and consequently, this enables developing MRAM-based memory arrays that can realize both synaptic behaviors and activation functions within their architecture without requiring to transfer the data to the processor to compute the activation functions.

IV. IMAC Architecture

The proposed SOT-MRAM-based neurons and synapses are utilized to form an in-memory analog computing (IMAC) architecture, as shown in Fig. 3. IMAC architecture includes a network of tightly coupled IMAC subarrays, which consist of weights, differential amplifiers, and neuron circuits, as shown in Fig. 3 (b). We have only shown the read path of the array for simplicity since the focus of this work is on the inference phase of the neural networks. The synaptic connections are designed in the form of a crossbar architecture, in which the number of columns and rows can be defined based on the number of input and output nodes in a single FC layer, respectively. During the configuration phase, the resistance of the SOT-MRAM-based synapses will be tuned using the bit-lines (BLs) and source-

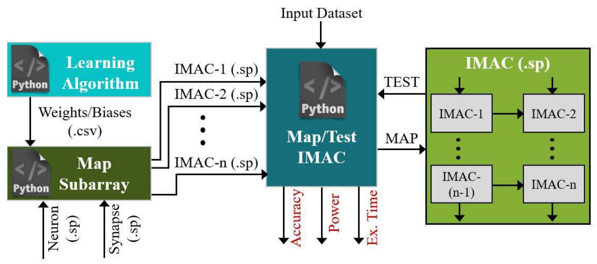

Fig. 5. Python-based simulation framework developed for SPICE circuit realization of IMAC unit.

lines (SLs) which are shared among different rows. The write word line (WWL) control signals will only activate one row in each clock cycle, thus the entire array can be updated using j clock cycles, where j is equal to the number of neurons in the output layer. In the inference phase, BL is connected to the input signals, SL is in a high-impedance (Hi-Z) state, and read word line (RWL) and WWL control signals are connected to VDD and GND, respectively. This will generate I^+ and I^- currents shown in Fig. 3 (b). The amplitude of produced currents depends on the input signals and the resistances of SOT-MRAM synapses already tuned in the configuration phase. Each row includes a shared differential amplifier, which generates an output voltage proportional to $\sum_i (I^+_{i,n} - I^-_{i,n})$ for the nth row, where i is the total number of nodes in the input layer. Finally, the outputs of the differential amplifiers are connected to the SOT-MRAM-based sigmoidal neurons to compute the activation functions.

In the IMAC architecture, each subarray computes both MAC operations and neurons' activation functions of a single FC layer and passes the result to its downstream neighbor IMAC subarrays that can compute the next FC layer. Thus, the IMAC architecture can be readily used to implement a multi-layer perceptron (MLP). Fig. 4 depicts the circuit realization of a $784 \times 16 \times 10$ SOT-MRAM based MLP classifier. In this regard, we developed a Python-based simulation framework to realize the SPICE circuit implementation of the IMAC-based MLP classifier, as shown in Fig. 5. The simulation framework includes a *Map Subarray* component that receives the trained weights and biases from an offline learning algorithm and builds individual subcircuits of IMAC subarrays for each FC layer in the MLP model. Then, the *Map/Test IMAC* component maps the generated IMAC subcircuits into the IMAC archi-

TABLE III
THE NOTATIONS AND DESCRIPTIONS OF THE PROPOSED LEARNING MECHANISM FOR THE IMAC-BASED MLP.

	Teacher Network	Student Network
Weights	$\mathbf{w}_i \in R$	$\mathbf{W}_i \in \{-1, +1\}$
Biases	$\mathbf{b}_i \in R$	$\mathbf{B}_i \in \{-1, +1\}$
Transfer Function	$y_i = \mathbf{w}_i x + \mathbf{b}_i$	$y_i = \mathbf{W}_i x + \mathbf{B}_i$
Activation Function	$o_i = sigmoid(-y_i)$	$o_i = sigmoid(-y_i)$

TABLE IV
PERFORMANCE COMPARISON AMONG VARIOUS IMPLEMENTATIONS OF THE BINARIZED $784 \times 16 \times 10$ MLP CLASSIFIER.

Architecture	Domain		Performance $(\frac{1}{s})$
	MAC	Activation Function	
CPU[1]	Digital	Digital	$\sim 10^4$
NMC [7]	Digital	Digital	$\sim 10^5$
AiMC [9]	Analog	Digital	$\sim 10^6$
IMAC	**Analog**	**Analog**	$\sim 10^8$

[1] Implemented on Intel® Core™ i9-10900X.

tecture, and runs SPICE circuit simulation to obtain accuracy, and measure power consumption and execution time.

Furthermore, we developed a hardware-aware teacher-student learning approach for IMAC with full-precision teacher and binarized student networks. Table III provides the notations and descriptions for the networks, in which x is the input of the network and y_i and o_i are the input and output of the ith neuron, respectively. To incorporate the features of the SOT-MRAM based synapses and neurons within our training mechanism, we made two modifications to the approaches previously used for training binarized neural networks (BNNs) [8]. First, we used binarized biases in the student networks instead of real-valued biases. Second, since our SOT-MRAM neuron realizes sigmoidal activation function ($sigmoid(-x)$) without any computation overheads, we could avoid binarizing the activation functions and reduce the possible information loss in the teacher or student networks [8]. After each weight update in the teacher network, we clip the real-valued weights and biases within the $[-1, 1]$ interval and then use the below deterministic binarization approach to binarize them:

$$W_{ij} = \begin{cases} +1, & w_{ij} \geq 0 \\ -1, & w_{ij} < 0 \end{cases} \quad \text{and} \quad B_{ij} = \begin{cases} +1, & b_{ij} \geq 0 \\ -1, & b_{ij} < 0 \end{cases} \tag{3}$$

Simulation results show a classification accuracy of 85.56% for $784 \times 16 \times 10$ IMAC-based MLP classifier, which is comparable to the 86.54% accuracy realized by BNNs such as XNOR-Net [8] and [16]. Moreover, Table IV provides a performance comparison between IMAC and other CPU, NMC, and IMC implementations of a $784 \times 16 \times 10$ MLP architecture. As listed in the table, IMAC-based MLP can complete the classification task approximately four, three, and two orders of magnitude faster than CPU, digital NMC [7], and mixed-signal AiMC [9] architectures, respectively. In particular, the IMAC's execution time to perform the recognition task is less than 40 clock cycles of the Intel® Core™ i9-10900X CPU with 3.7 GHz frequency, while it takes more than 1,000,000 cycles for the CPU to complete the similar task.

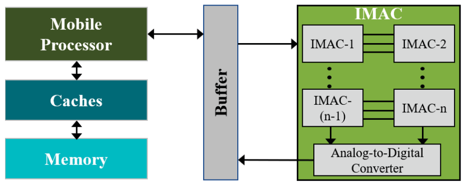

Fig. 6. Heterogeneous CPU-IMAC data flow architecture.

TABLE V
X86 ISA EXTENSION.

Instruction	Function
store_imac r1, addr;	Signed binarization and store data to buffer
load_imac r1, addr;	Load data from buffer

V. HETEROGENEOUS CPU-IMAC ARCHITECTURE

Despite the aforementioned performance advantages of the IMAC arrays for MLP classifiers, their low-precision computation can be prohibitive for many mobile computing applications that require large-scale deep learning models. Thus, this section proposes a simple but effective method to integrate IMAC with general-purpose mobile processors to realize a mixed-signal and mixed-precision CNN inference achieving performance and energy improvements.

We propose a heterogeneous architecture that uses the CPU to realize full-precision convolution layers, while the low-precision FC layers are implemented on IMAC. The CPU-IMAC architecture uses IMAC as an on-chip co-processor that shares the cache hierarchy with CPU as shown in Fig. 6. This is because the intermediate data transfer between CPU and IMAC can be faster as compared to placing IMAC off-chip. To remove the need for digital-to-analog converters (DACs) between conventional digital CPU and analog IMAC, a $sign$ unit is used in the last convolution layer to convert the output of the convolution layer to `-1,0,1` values which can be realized by V_{SS}, GND, V_{DD} voltages without requiring a DAC. To enable fast data transfer between CPU and IMAC, a hardware buffer and a 'ready' register are added. The buffer can be used to store both inputs and outputs of the IMAC.

This design extends the existing **x86 Instruction Set Architecture (ISA)** with two new instructions, which are **`store_imac`** and **`load_imac`** as their format and description listed in Table V. The buffer address is not part of the memory address space. Before IMAC starts its computation, each of the input data is converted through the $sign$ unit and stored in the buffer. A designated address (*e.g.,* 0x0) is reserved for the 'ready' register. Before transferring data to the buffer, a **`store_imac`** instruction is executed to set the 'ready' register to `0`. After all of the input data are stored in the buffer, the 'ready' register is set to `1`. When the IMAC computation is done, the analog output of the IMAC is converted to digital via an array of 3-bit analog-to-digital converters (ADCs). The buffer is used to store the IMAC output, and the 'ready' register is set to `-1`, indicating that the buffer is not used for holding input data anymore.

When IMAC computes, the CPU waits for the results. Typically, there are two ways to resume the CPU computation after offloading computation to a co-processor: pulling and interrupt. Pulling requires the CPU to periodically read the completion status of the IMAC, which adds instruction overheads and wastes energy. Interrupt allows the co-processor to notify the CPU when the computation on the co-processor is done so that the CPU can run other tasks in the meanwhile. However, handling interrupt requires additional latency. The proposed IMAC computation is deterministic and has relatively low latency (*i.e.,* tens of CPU cycles time). Therefore, the proposed design uses a timer instead of the pulling or interrupt mechanism to resume CPU computation. For different neural network topologies, the expected computation time can be determined and loaded to a timer register before IMAC starts computation. After input transfer to the buffer is done, the timer register starts to count down. The CPU can start to read the IMAC results after the timer counts down to zero.

A. Hardware-Aware Learning Algorithm

To fully leverage the energy and performance benefits of the heterogeneous CPU-IMAC architecture without compromising accuracy, we developed a hardware-aware learning algorithm realizing the computation limitations and features of our mixed-precision and mixed-signal CPU-IMAC architecture. The learning algorithm includes two training steps: in *step-1*, the vanilla full-precision CNN model is trained using backpropagation without any changes in the learning mechanism or CNN model. In *step-2*, we divide the CNN models into two parts, convolution layers, and FC layers, and retrain the isolated FC layers while incorporating the hardware characteristics of IMAC subarrays since that is the portion of the CNN model that will be implemented on the IMAC unit. To achieve this goal, first, we input the entire training dataset to the CNN model trained in step-1 and read the output of the last convolution layer after flattening to obtain a new train dataset for training the FC layers. A $sign$ function is applied to the output of the convolution layer to imitate the inference hardware and generate `-1, 0, 1` values for the input of the FC layers. Accordingly, we modify the FC layers based on the features of IMAC by using binarized synapses and $sigmoid(-x)$ activation functions, which can be realized by SOT-MRAM-based synapses and neurons. Finally, the teacher-student learning mechanism described in the previous section will be utilized along with the convoluted training dataset to train the IMAC-based FC classifier. It is worth noting that most of the existing CNN models use Rectified Linear Units (ReLUs) to realize a non-saturating nonlinearity due to their implementation simplicity and performance benefits compared to digital implementations of $tanh$ and $sigmoid$ activation functions. However, while we still use ReLU in the convolution layers implemented on CPU, in IMAC architecture, our proposed analog neurons realize intrinsic high-performance sigmoidal activation functions that provide accuracy benefits with minimal performance overheads.

B. Simulation Results and Discussion

In this work, we implemented two CNN models on the proposed CPU-IMAC architecture, *i.e.* LeNet [17] and VGG [18] for MNIST [17] and CIFAR-10 [19] pattern recognition applications, respectively. To obtain the inference accuracy of the CPU-IMAC based CNN implementations, first, we used TensorFlow [20] platform to implement the convolution layers, then the output of the last convolution layers is transferred to the Python-based simulation framework that we developed for the SPICE circuit implementation of IMAC, shown in Fig. 5. The simulation results show recognition accuracy values of 97.39% and 92.87% for MNIST and CIFAR-10 datasets using mixed-precision and mixed-signal CPU-IMAC implementation of LeNet and VGG models, respectively, which is comparable to the 98.29% and 93.14% accuracies realized by full-precision digital implementations of these models on CPU.

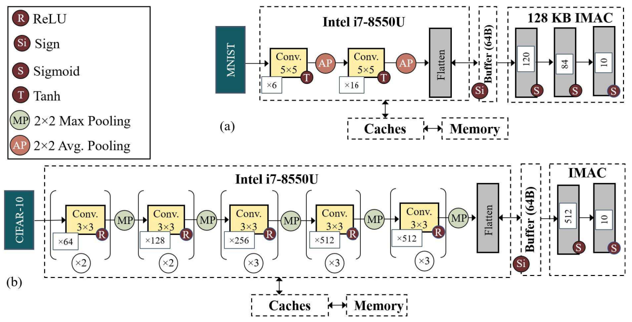

Fig. 7. (a) LeNet-5 [17] for MNIST, (b) VGG [18] for CIFAR-10.

For performance analyses, we use Champsim [21], a trace-based simulator that models an out-of-order core with a detailed memory system. The core parameters are adapted from mobile processor Intel i7-8550U [22]. The main memory (LPDDR3) timings are adopted from Micron EDF8132A1MC [23]. IMAC architecture includes 128KB of SOT-MRAM cells constituting four IMAC subarrays of 512b×512b. The size of the buffer is 64 bytes, which is enough to transfer the data produced in the last convolution layer of LeNet-5 and VGG models to IMAC and the result of IMAC computation back to CPU. The simulation results exhibit 11.2% and 1.3% speedup for the inference operation of LeNet and VGG models, respectively, which is proportional to the ratio of FC layers to convolution layers computation. The LeNet model used herein has 2 convolution layers and 3 FC layers, while the VGG model used for the CIFAR-10 dataset includes 13 convolution layers, and only 2 FC layers, as shown in Fig. 7.

To realize the energy benefits of the proposed CPU-IMAC architecture, we developed an analytical model based on McPAT [24], CACTI [25], and Micron DDR3 SDRAM System-Power Calculator [26]. CACTI is used to get per access energy for different levels of cache. McPAT is used to get the energy consumed by the core. We modify the Micron DDR3 SDRAM System-Power Calculator to model memory power consumption with current numbers from Micron EDF8132A1MC [23]. Fig. 8 provides a comparison between CPU-IMAC architecture and the baseline mobile processor in terms of energy consumption. The results demonstrate a 10% and 6.5% energy reduction for CPU-IMAC-based implementations of LeNet-5 and VGG models, respectively. It is worth noting that the

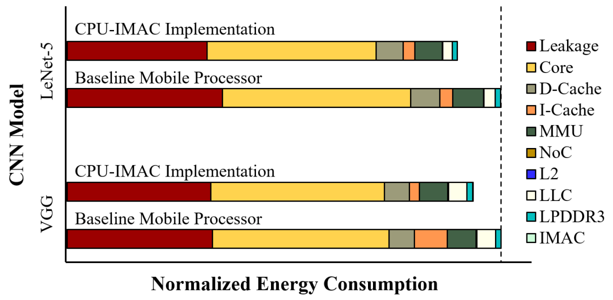

Fig. 8. Energy consumption comparison between baseline mobile processor and CPU-IMAC architecture for LeNet-5 and VGG CNN models.

TABLE VI
SPEEDUP, ENERGY IMPROVEMENT AND ACCURACY DIFFERENCE OF CPU-IMAC COMPARED TO THE BASELINE MOBILE PROCESSOR.

CNN Model	Speedup	Energy Improvement	Accuracy Diff.
LeNet-5 [17]	11.2%	10%	-0.9%
VGG [18]	1.3%	6.5%	-0.27%

total energy consumption of IMAC is equal to 97 nJ and 512 nJ for the LeNet and VGG implementations, respectively, which are negligible compared to the energy consumption of CPU as shown in Fig. 8. Finally, Table VI summarizes the speedup, energy improvement, and accuracy difference of CPU-IMAC architecture compared to the baseline mobile processor, showing that the proposed architecture can achieve important performance and energy improvements while realizing comparable accuracy.

VI. CONCLUSION

We proposed a heterogeneous mixed-precision and mixed-signal CPU-IMAC architecture to realize energy and performance improvements for CNN inference in mobile devices. The analog IMAC units were proposed to be integrated with digital mobile processors to implement FC and convolution layers of CNN models, respectively. We investigated the circuit-, architecture-, and algorithm-level requirements for efficient realization of the CPU-IMAC architecture and verified its potential performance and energy benefits via circuit and architecture level simulations of two CNN models, i.e. LeNet and VGG. It has been shown that the IMAC unit can realize orders of magnitude performance improvement for FC classifiers. However, when integrated with mobile processors to implement CNN models, the CPU-IMAC architecture performance and energy improvements follow Amdahl's law and is proportional to the ratio of FC layers to convolution layers. Despite these limitations, we could obtain an energy reduction of 6.5% and 10% for VGG and LeNet models, which is considerable for mobile computing applications. The proposal of CPU-IMAC architecture provides several possibilities for future work to realize significantly more performance and energy improvements, including but not limited to: (1) design space exploration to develop CNN models optimized to take advantage of the benefits of CPU-IMAC architecture, especially via tuning the ratio of convolution layers to FC layers within a CNN model; (2) Extending the utilization of IMAC to convolution layers through convolution unrolling techniques.

REFERENCES

[1] D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," *Nature Electronics*, vol. 1, no. 6, pp. 333–343, 2018.
[2] J. Ahn *et al.*, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ser. ISCA '15, 2015, p. 105–117.
[3] M. Le Gallo *et al.*, "Mixed-precision in-memory computing," *Nature Electronics*, vol. 1, no. 4, pp. 246–253, 2018.
[4] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16, 2016, p. 27–39.
[5] S. Yin *et al.*, "High-throughput in-memory computing for binary deep neural networks with monolithically integrated rram and 90-nm cmos," *IEEE Transactions on Electron Devices*, vol. 67, no. 10, 2020.
[6] K. Spoon *et al.*, "Accelerating deep neural networks with analog memory devices," in *2020 IEEE International Memory Workshop (IMW)*, 2020.
[7] S. Angizi *et al.*, "Mrima: An mram-based in-memory accelerator," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 5, pp. 1123–1136, 2020.
[8] M. Rastegari *et al.*, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Computer Vision – ECCV 2016*, B. Leibe *et al.*, Eds., 2016, pp. 525–542.
[9] J. Doevenspeck *et al.*, "Sot-mram based analog in-memory computing for dnn inference," in *IEEE Symposium on VLSI Technology*, 2020.
[10] L. Liu *et al.*, "Spin-torque switching with the giant spin hall effect of tantalum," *Science*, vol. 336, no. 6081, pp. 555–558, 2012.
[11] Y. Zhang *et al.*, "Compact modeling of perpendicular-anisotropy cofeb/mgo magnetic tunnel junctions," *IEEE Transactions on Electron Devices*, vol. 59, no. 3, pp. 819–826, March 2012.
[12] R. Zand, A. Roohi, and R. F. DeMara, "Energy-efficient and process-variation-resilient write circuit schemes for spin hall effect mram device," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 9, pp. 2394–2401, Sep. 2017.
[13] G. Khodabandehloo, M. Mirhassani, and M. Ahmadi, "Analog implementation of a novel resistive-type sigmoidal neuron," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 4, pp. 750–754, 2012.
[14] J. Shamsi *et al.*, "Hyperbolic tangent passive resistive-type neuron," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015.
[15] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm," *Integration*, vol. 58, pp. 74–81, 6 2017.
[16] M. Courbariaux *et al.*, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," *arXiv: Learning*, 2016.
[17] Y. Lecun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[18] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," in *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, 2015, pp. 730–734.
[19] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.
[20] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th Symposium on Operating Systems Design and Implementation (OSDI 16)*, Nov. 2016, pp. 265–283.
[21] ChampSim, "Champsim simulator," 2020. [Online]. Available: https://github.com/ChampSim/ChampSim
[22] Intel, "Intel core i7-8550u," 2020. [Online]. Available: https://laptopmedia.com/highlights/intel-core-i7-8550u-coffee-lake-specs-performance-and-detailed-benchmarks/
[23] Micron, "Micron edf8132a1mc," 2020. [Online]. Available: https://datasheetspdf.com/datasheet/EDF8132A1MC.html
[24] S. Li *et al.*, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2009, pp. 469–480.
[25] R. Balasubramonian *et al.*, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 2, pp. 14:1–14:25, Jun. 2017.
[26] Micron, "Micron system power calculators," 2020. [Online]. Available: https://media-www.micron.com/-/media/client/global/documents/products/power-calculator