Energy-Efficient SNN Implementation Using RRAM-Based Computation In-Memory (CIM)

El Arrassi, Asmae; Gebregiorgis, Anteneh; Haddadi, Anass El ; Hamdioui, Said

DOI
10.1109/VLSI-SoC54400.2022.9939654

Publication date
2022

Document Version
Final published version

Published in
Proceedings of the 2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)

2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC) | 978-1-6654-9005-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/VLSI-SoC54400.2022.9939654

Energy-Efficient SNN Implementation Using RRAM-Based Computation In-Memory (CIM)

Asmae El Arrassi†, Anteneh Gebregiorgis‡, Anass El Haddadi†, Said Hamdioui‡
‡Department of Quantum and Computer Engineering, Delft University of Technology, Delft, The Netherlands
Email: {A.B.Gebregiorgis, S.Hamdioui}@tudelft.nl
†ENSA, Abdelmalek Essaadi University, Al-Hoceima, Morocco
Email: asmaeelarrassi@gmail.com, e.elhaddadi@uae.ac.ma

***Abstract*—Spiking Neural Networks (SNNs) can drastically improve the energy efficiency of neuromorphic computing through network sparsity and event-driven execution. Thus, SNNs have the potential to support practical cognitive tasks on resource constrained platforms, such as edge devices. To realize this, SNN requires energy-efficient hardware which can run applications with a limited energy budget. However, the conventional CMOS implementations cannot achieve this goal due to the various architectural and technological challenges. In this work, we address these issues by developing an energy-efficient and accurate SNN hardware based on Computation In-Memory (CIM) architecture using Resistive Random Access Memory (RRAM) devices. The developed SNN architecture is based on unsupervised Spike Time Dependent Plasticity (STDP) learning algorithm with online learning capability. Simulation results show that the proposed architecture is energy-efficient with a consumption of ≈20 fJ per spike, while maintaining state-of-the-art inference accuracy of 95% when evaluated using the MNIST dataset.**

***Index Terms*—SNN, RRAM, In-Memory Computing, STDP.**

I. Introduction

The breakthrough in Artificial Intelligence (AI) has led to a booming increase in AI-based applications and services [1]. In this regard, Spiking Neural Networks (SNNs) are empowering a variety of tasks, such as Brain Machine Interface (BMI). Although SNNs are able to mimic the human brain and deliver high accuracy for various applications, their implementation on conventional Von-Neumann based architectures (such as CPU, GPU, and TPU) suffer from the three well-known architectural walls, such as the so-called *memory-wall* [2]. As a result, excessive time and energy are spent on moving massive amounts of data between the memory and data paths, which makes such architectures to be extremely energy-inefficient [3], [4]. These challenges are therefore hindering the widespread application of SNN on resource-constrained platforms, such as edge computing. Therefore, there is a clear need for energy-efficient implementation of SNN.

Computation In-Memory (CIM) using conventional CMOS or emerging devices such as Resistive Random Access Memory (RRAM) offers great opportunities to overcome these challenges [5], and implement energy-efficient SNN with a radically new architecture. CIM integrates computation and storage together (brain like) and provides an efficient implementation of Vector-Matrix Multiplication (VMM), which is the kernel operation in SNN [6]. Therefore, CIM provides a huge potential for energy-efficient SNN-based edge computing. CIM-based accelerators have been explored to improve the energy efficiency of different applications and kernels, such as big data and neuromorphic computing [7]–[9]. However, there have been few studies exploring RRAM-based SNN implementation [10]–[14]. Although the works in [10]–[13] exploit the inherent CIM potential to reduce energy consumption, the lack of bio-plausibility makes them energy-inefficient. The work in [14] combines analog neurons and RRAM-based synapses to implement an SNN. However, its simplistic hardware structure can only accommodate a single fully connected layer, which makes it have poor classification accuracy. Moreover, CIM-based SNN implementations with in-situ learning support for runtime fine-tuning and adjustment are needed for energy-efficient and accurate inference. Therefore, there is a clear demand for energy-efficient CIM with on-line learning support for edge applications.

In this paper, we propose an energy-efficient SNN based on CIM architecture using RRAM devices. The developed SNN architecture is based on unsupervised Spike Time Dependent Plasticity (STDP) learning algorithm with support for light-weight online learning. The online learning enables the proposed SNN architecture to fine-tune the synaptic weights to learn new features and maintain its accuracy when subjected to environmental changes. Simulation results show that the proposed CIM-based SNN architecture is highly energy-efficient with a consumption of ≈20 fJ per spike, while maintaining state-of-the-art inference accuracy of 95%. The main contributions of the paper are summarized as follows:

- Design and implementation of energy-efficient CIM-based SNN using RRAM crossbar array.
- Lightweight online learning feature to fine-tune the weights and maintain the trained accuracy.
- Validation of the architecture using software and hardware simulation tools.
- Results show that the proposed CIM-based SNN achieves an energy efficiency of ≈20 fJ per spike.

The remainder of the paper is organized as follows: Section II presents the basic concepts. Section III presents the proposed architecture and STDP training. Section IV presents the architectural implementation followed by simulation results in Section V. Finally, Section VI concludes the paper.

This work was supported in part by the EU H2020 grant "DAIS" that has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 101007273.

978-1-6654-9005-4/22/$31.00 ©2022 IEEE

II. Preliminaries

A. SNN fundamentals

1) SNN basics: SNN mimics the human brain, which consists of billions of neurons interconnected by trillions of synapses [15]. A biological neuron integrates incoming spike voltages received from other connected neurons to form the membrane potential. When the membrane potential crosses the firing threshold, the neuron fires an output spike. This way information is transmitted only when the neuron membrane potential reaches the threshold. There are different models such as Hodgkin and Huxley model [16] and the *Leaky Integrate and Fire LIF)* [17], [18] to implement SNNs that mimic biological neurons by encoding information into spikes [19]. The encoding can be achieved by either *rate coding* [20] or *temporal coding* [21].

2) SNN training technique: SNN training can be classified into two categories [22], namely, a converted SNN and a directly trained SNN. In the first category, the network is trained as Artificial Neural Network (ANN) with back-propagation technique [23]–[25]. Then, the trained ANN is parsed into an SNN, which uses the pre-trained weights for inference activities. However, converted SNNs perform poorly during inference making them undesirable [26]. The second class is the directly trained SNN that uses different learning methods such as STDP [27]–[29] to train the SNN architecture. The main advantage of this approach is that it is scalable and it exploits the event-driven nature of SNN for efficient training [30].

B. CIM using RRAM devices

RRAM devices have gained widespread attention due to their non-volatility, high integration density, and ability to overcome memory bandwidth issues by executing operations within the memory [31]. RRAM devices can be structured in a crossbar array to build a CIM unit, where computation and storage are integrated within the same physical location. Figure 1 shows a crossbar structure with N wordlines (input voltages) and M bitlines, in which the wordlines and bitlines are connected through an RRAM bit-cell (1T1R) at their intersection. The crossbar can perform Vector Matrix Multiplication (VMM) operation by applying a voltage vector $V{=}V_i$ (where $i{\in}\{1, N\}$) to the crossbar matrix of conductance values $G{=}G_{ij}$ (where $i{\in}\{1, N\}$, $j{\in}\{1, M\}$). At any instance, each column can perform a multiplication-and-accumulation (MAC) operation, with the output current vector I, in which each element of the output current is obtained as shown in Equation (1).

$$I_j = \sum_{i=1}^{N} V_i \cdot G_{ij} \tag{1}$$

All M MAC operations are performed with O(1) complexity, which is essential to implement energy-efficient SNN.

III. Proposed CIM-based SNN architecture design

For energy-efficient and accurate SNN implementation we develop an architecture with the following key features:

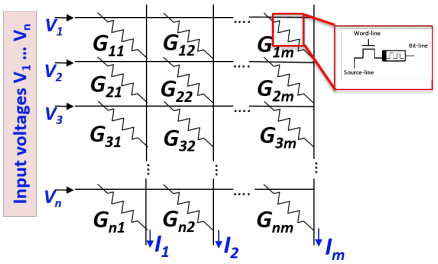


Fig. 1: RRAM crossbar array structure for CIM operation.

- CIM-based two layer architecture: An optimized two layer SNN architecture is developed to improve the energy efficiency, while maintaining high accuracy.
- Adopting LIF neuron model: LIF neuron model is adopted for efficient implementation due to its simplicity and biological plausibility [32].
- Light-weight on-line learning support: On-line learning functionality to fine-tune synapses to maintain accuracy.

Next these key features will be discussed in details.

A. Two layer architecture design

The proposed two layer SNN architecture shown in Figure 2 is chosen as it minimizes the number of neurons which makes it energy-efficient architecture. The first (input) layer takes the input data vector and generates spike trains by using Poisson encoding schemes [33]. It should be noted that a pre-processing step is needed to convert the stimulus (two-dimensional input image) pixels into a one-dimensional (1D) vector of pixels. As shown in the figure, the 1D pixel vector is fed to the input layer, where each pixel in the 1D vector is mapped to the corresponding neuron which uses Poisson encoding to generate the corresponding spike train, being equivalent to the input pixel intensity. Finally, the spike trains are fed to the output layer through a fully connected network.

The neurons at the output layer are based on the Leaky-Integrate-and-Fire (LIF) model [18] and perform excitation and inhibition tasks by increasing or decreasing the firing probability. This is achieved by reinforcing the connected synapses based on the STDP learning rule [34]. In the excitation phase, the neurons accumulate the weighted input spikes. The intensity of the weighted input spikes determine the excitation level of the output neurons, resulting in an output spike only from the neuron that are sensitive to a particular feature. Once an output neuron generates a spike, it sends an inhibition signal to all other neurons in the output layer to update their membrane potential based on the intensity of their synapses. The inhibition synapse connectivity defines the inhibition intensity and membrane potential shift amplitude of the inhibited neurons. Increasing the inhibition synapse connectivity to the non-firing neurons weakens their membrane potential which allows the firing neuron (winner) to dominate the spike response. This approach aims at mimicing the biological phenomenon by enabling competition between the neurons, which makes the proposed SNN biologically plausible architecture.

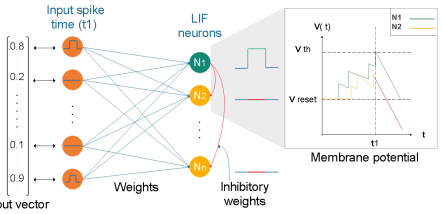


Fig. 2: Two-layer SNN architecture with an input layer to generate spike trains, and a fully-connected output layer.

B. Adopted neuron model

The LIF neuron model presented in [18] is adopted in this work as it provides better trade-off between accurately mimicking of biological neuron and implementation simplicity [32], [35]. The LIF model captures the facts that a neuron has a leaky membrane and performs spatial and temporal integration of synaptic inputs to generate a spike when the voltage reaches a certain threshold. After spiking, the neuron goes to a refractory period [32]. The LIF model can be expressed as shown in Equation (2) [18]:

$$\tau_{mem}\frac{\partial v(t)}{\partial t} = -v(t) + v_{rest} + I(t) \tag{2}$$

where τ_{mem} is the membrane time constant of the neuron, $v(t)$ is the neuron membrane potential, v_{rest} is the resting membrane potential, and $I(t)$ is the total current of a neuron at time t. Figure 3 shows a LIF neuron and its dynamics. As shown in the figure, the LIF neuron receives spike inputs from multiple presynaptic neurons and accumulates the weighted spikes to increases its membrane potential. When the membrane potential reaches the threshold voltage v_{th}, the neuron generates an output spike and its membrane potential is decayed to a reset voltage v_{reset}. Then, the neuron enters a refractory period in which it does not accumulate any incoming spikes. Once the refractory period has lapsed, the neuron will be active again to receive input spikes. When an active neuron does not receive any input spike, its membrane potential will decay (leak) gradually.

C. STDP learning method

STDP learning rule is adopted due to its bio-plausibility, and training speed [36]. STDP is a phenomenon in which the precise timing of spikes affects the sign and magnitude of changes in synaptic strength [27]. STDP is dependent on the spike sequences of presynaptic and postsynaptic neurons. When a presynaptic neuron precedes the postsynaptic neuron, the synapse connecting these neurons is potentiated (increased) otherwise, the synapse is depressed (decreased) as shown in Figure 4. The intensity of the potentiation and depression of the synapses is dependent on the timing of the postsynaptic neuron and the presynaptic neuron firing $\Delta t = t_{post} - t_{pre}$. Therefore, the postsynaptic spike weight update is expressed as shown in Equation (3):

$$\Delta w_{post} = \eta_{post} x_{pre}(w_{max} - w)^{\mu} \tag{3}$$

where η_{post} is the learning rate of the postsynaptic neurons, w_{max} is the maximum weight, μ defines the weight update dependency on the old weight, and x_{pre} monitors the presynaptic

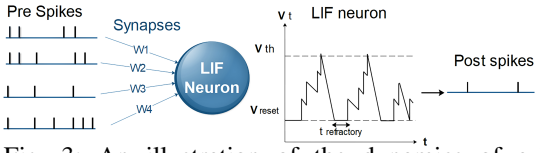


Fig. 3: An illustration of the dynamics of a LIF neuron membrane potential.

neuron activity history. x_{pre} gradually increases/decreases its value depending on the reception of input spike. Similarly, the presynaptic spike weight update is expressed in Equation (4):

$$\Delta w_{pre} = -\eta_{pre} x_{post} w^{\mu} \tag{4}$$

where η_{pre} is the learning rate of presynaptic neurons, and x_{post} traces postsynaptic neuron activity [27]. The synapse weight update is expressed in Equation (5):

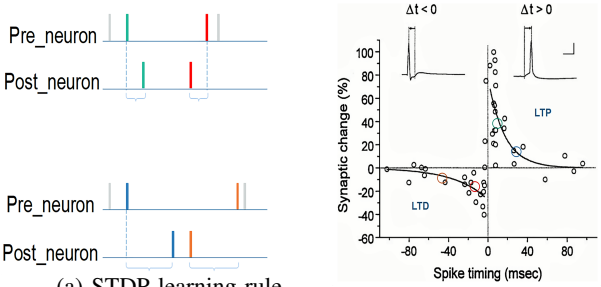
$$\Delta w = \Delta w_{pre} + \Delta w_{post} + w \tag{5}$$

Where Δw_{pre} and Δw_{post} are the presynaptic and postsynaptic spike weight update, and w is the synapse weight value.

Lateral inhibition is implemented using fixed recurrent synapses that connect each neuron to all the other neurons. During training, few neurons can dominate the spike firing activity while the rest of the neurons are less active. This firing imbalance is increased by lateral inhibition making the less active neurons unable to reach the firing threshold. In order to address the domination of the spike generation by few neurons, a homeostasis (dynamic threshold) technique is utilized to balance the firing probability of the neurons according to changing conditions. This can be implemented by increasing the threshold voltage ($V_{threshold}$) whenever a neuron receives an input spike, and gradually decreasing the $V_{threshold}$ when it is not generating output spikes [27]. Therefore, the active neurons need relatively higher membrane potential to reach the dynamic threshold, while the less active neurons are encouraged to fire by having smaller thresholds. The dynamic threshold of a neuron is expressed as follows:

$$v_{threshold}(T_{step}) = V_{threshold} + \theta(T_{step}) \tag{6}$$

Where $V_{threshold}$ is the constant threshold voltage and $\theta(T_{step})$ is a variable that shifts the dynamic threshold depending on the activity of the neuron. Moreover, the trained weights



(a) STDP learning rule (b) Biological inspiration

Fig. 4: STDP learning rule: (a) basics (b) the biological data on which STDP was based [34].

Algorithm 1 Unsupervised training algorithm

Input: Network$_{arch}$, N 28×28 pixel training samples
Output: Trained network

procedure TRAINING PHASE
  Initialize network weights with uniform distribution
  for Sample$_i$ = 1 to N **do**
    ST = Poisson (Sample$_i$); /* ST= 784 spike trains*/
    $X_{(pre/post)}$ = 0; /*$X_{(pre/post)}$ is pre/post synaptic trace*/
    Δw = 0;
    for T_{step} = 1 to 350 **do**
      $\forall_{neurons}$ Compute membrane potential $V(T_{step})$;
      $V(T_{step})$= $\sum_{i=1}^{784}(w_{(i)}ST_i(T_{step}) + b_i)$;
      $v_{threshold}(T_{step})$ = $V_{threshold}$ + $\theta(T_{step})$; /* (See Equation (6)) */
      if $V(T_{step}) \geq v_{threshold}(T_{step})$ **then**
        Generate output spike;
        Decay $V(T_{step})$ to a reset voltage;
        $\theta(T_{step})$ Increase;
      else
        $\theta(T_{step})$ decay;
      if a Spike is generated **then**
        ST(T_{step}) = 1;
        $X_{(pre/post)}$ = 1;
      else
        ST(T_{step}) = 0;
        $X_{(pre/post)}$ -= ϵ; / ϵ= decay rate */
      Calculate $\Delta w_{pre/post}$ based on Eq.(3 and 4);
      Calculate Δw based on Equation(5);
    Update weight w by Δw;
    Normalize neuron weights for a balanced learning.

are normalized to further balance the neuron responses, prevent over-fitting and reduce weight saturation or vanishing phenomena.

The details of the adopted unsupervised STDP training algorithm with its homeostasis and weight normalization techniques is presented in Algorithm 1.

IV. CIM ARCHITECTURAL IMPLEMENTATION

The CIM implementation of the proposed SNN architecture (Figure 2) using RRAM crossbar array is shown in Figure 5. The crossbar shown in Figure 5 uses a 1T1R bit-cell structure to store the synaptic weights. The spikes from the presynaptic neurons are connected to the synapses array in the crossbar via the bitline input voltages. The input voltages generate an output current which represents the weighted sum of the input spikes in each column that corresponds to one LIF neuron. The column output currents stimulate the LIF neurons to modify their membrane voltage (excitation phase). Once the membrane voltage of any postsynaptic neuron reaches the threshold, an output spike and an inhibition signal are generated. The inhibition signal is fed to the other neurons

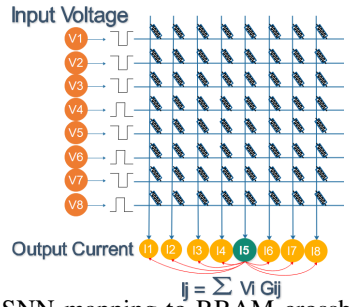


Fig. 5: SNN mapping to RRAM crossbar array.

to suppress their potential (RESET operation), while the output spike is used to determine the output of the network. Thus, CIM enables the implementation of massively parallel Vector Matrix Multiplication (VMM) operation in a compact and efficient manner. Therefore, the RRAM crossbar-based CIM implementation reduces the complexity of the kernel operation (VMM) from $O(n^2)$ to $O(1)$. Moreover, RRAM has comparatively lower read and write latency (<10ns) than other emerging technologies such as PCM [37]. Thus, RRAM offers good trade-off between latency and energy-efficiency for wide range of applications.

Different RRAM reliability issues such as conductance drift, endurance and retention failure can affect the stored synaptic weights and lead to inference accuracy degradation of the CIM-based SNN architecture [38]. To cope with these issues and enable the network to learn new features, a light-weight online learning scheme is adopted to fine-tune the synaptic weights in the RRAM crossbar. This helps to maintain the baseline trained accuracy during the operational time of the SNN. Moreover, online learning increases the long-term network stability. The online learning is implemented using a simplified version of the STDP learning rule [39] by adding signals at the output of the neuron that will fine-tune the RRAM synaptic weights. The proposed online learning has minor impact on energy efficiency and device endurance due to the small number of write operations on the RRAM devices.

V. EXPERIMENTAL RESULTS AND DISCUSSION

A. Simulation setup

The simulation setup used in this work is presented in Table I. The proposed architecture is implemented and trained in Python using bindsnet [40], pytorch and NVIDIA CUDA libraries. The MNIST dataset [41] is used to train and evaluate the accuracy of the SNN architecture. The MNIST image pixels are converted into spike trains using Poisson encoding scheme, where each spike train is 350ms in length (duration). The network is trained offline and then implemented in hardware with VHDL and spice simulation of RRAM crossbar. The parameters of the RRAM devices used to simulate the CIM architecture are also presented in Table I.

B. Accuracy evaluation

The training and inference accuracies of the proposed SNN architecture are evaluated with MNIST dataset. This subsec-

TABLE I: SNN simulation setup and RRAM parameters.

SNN parameter	Value
Threshold voltage constant	-52 mV [42]
Resting potential	-65 mV [42]
Membrane reset potential	-60 mV [42]
Refractory period	5 ms
Membrane time constant	100 ms
Spike trace decay time constant	20 ms
Postsynaptic learning rate	10^{-4}
Presynaptic learning rate	10^{-2}
Weight dependency constant	0.4
Normalization weight constant	78.4
Spike train duration	350 ms

RRAM parameters	Specification
RRAM Device	HfO_2/TiO_x [43]
R_{off} (HRS)	$100k\Omega$
R_{on} (LRS)	$3k\Omega$
CMOS Technology	90nm TSMC

tion discusses the training accuracy evaluation first followed by a discussion on the inference accuracy evaluation.

1) Training accuracy evaluation: To evaluate the training progress of the network, we evaluated the accuracy of the network by gradually increasing the number of training samples by 20 epochs. Figure 6 shows the evolution of the training accuracy as a function of training samples. As shown in the figure, the training accuracy reaches its peak accuracy, 95%, when trained with 44,000 samples. Afterward, the network stabilizes with minor over-fitting.

2) Inference accuracy evaluation: To evaluate the inference accuracy of the architecture, the input image pixels are first converted into spike trains as shown in Figure 7. From the figure, it can be observed that the spike trains are dependent on the pixel intensity and they are divided into three main levels. High-intensity image pixels are converted into high rate spike trains (yellow). Medium-intensity pixels, e.g., pixels around the edges of the digit, are converted into medium rate spike trains (green, blue). Finally, low-intensity pixels are converted into low rate spike trains (purple). The figure also shows the reconstructed image from the inference output spikes of the network.

These spike trains are used to evaluate the inference accuracy of different STDP models such as additive [44], multiplicative [45], power law [46], Gutig [47] and Rossum [48]. Figure 8 shows the inference accuracy of these STDP model variants. From the figure, it can be observed that the Gutig model achieves higher inference accuracy (95%) than the other models. The Gutig model also demonstrated high stability

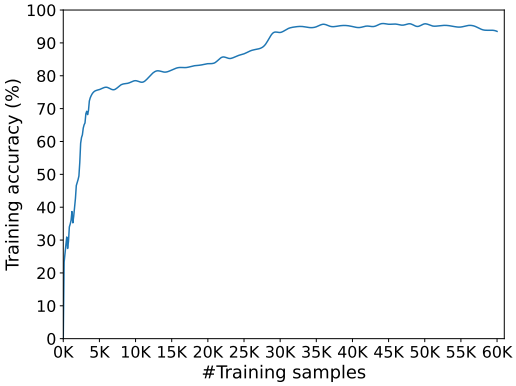


Fig. 6: Training accuracy vs training samples for 20 epochs.

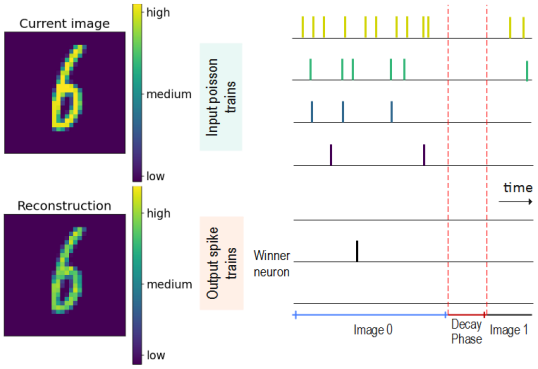


Fig. 7: The schematic of the spike trains activity.

and fast convergence during the training phase. Therefore, the Gutig model is adopted as the standard STDP model in this work. Moreover, we evaluated the difference between supervised and unsupervised learning by training our architecture using the supervised STDP learning algorithm presented in [28]. Table II presents the differences in inference accuracy and the training time of the adopted unsupervised STDP learning and the supervised learning rule [28]. The table shows that supervised learning achieved slightly better accuracy than unsupervised learning at the cost of a higher training time.

TABLE II: Supervised vs unsupervised learning.

Learning rule	Inference accuracy	Training Time (s/sample)	Testing Time (s/sample)
Unsupervised	95%	14.62	0,6
Supervised	96.28%	27.72	0.75

C. Energy efficiency evaluation

To evaluate the energy consumption of the proposed CIM-based SNN architecture, we programmed the trained weights to the RRAM crossbar, and the spike trains are provided as input for the VMM operation on the crossbar. The CIM implementation utilizes 20, 512 $\times$ 512 RRAM crossbars. Results show that the proposed CIM implementation performs VMM in an energy-efficient manner with an energy consumption of $\approx$20fJ energy per spike.

D. State of the art comparison

There are different works in the literature implementing SNN architectures with unsupervised training using the STDP learning rule [49], [50]. The work in [49] proposed a locally connected network that shows high spike activity for training

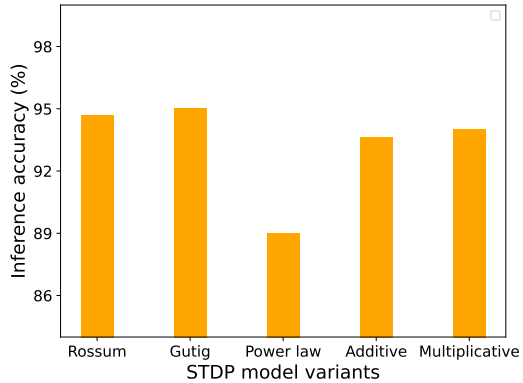


Fig. 8: Inference accuracy of different STDP model variants.

TABLE III: Accuracy and energy efficiency comparison.

Network architectures	Learning rule	Spike activity (spikes/iteration)	Number of neurons	Test Accuracy	Energy/Spike
Saunders [49]	unsupervised	×7	9000	95.02%	N/A
Meng [50]		×19	8400	95.6%	N/A
Valentian [14]	Supervised	N/A	N/A	88%	180 pJ
Lee [51]		×19	919	69%	270 fJ
Our work	Unsupervised STDP	×1	6400	95%	**20 fJ**

and inference. However, the augmented spike activity results in high energy consumption due to the distributed nature of the recurrent inhibitory connections. Similarly, the work in [50] utilizes a highly-parallel SNN consisting of multiple independent sub-networks. However, it has high spike activity than the activity in [49]. Table III compares the spike activity, accuracy and energy efficiency of our work with different State-of-The-Art SNN architectures [14], [49]–[51]. Although the works in [49], [50] achieve comparable (slightly better) accuracy than our work, they have much higher spike activity and resource utilization (more than 30%) than our proposed solution, as shown in Table III. The works in [14] and [51] proposed an SNN implementation on RRAM crossbar. However, both solutions are energy-inefficient when compared to our work.

VI. Conclusion

In this work, we proposed an energy-efficient SNN hardware based on Computation-in-Memory (CIM) architecture using Resistive Random Access Memory (RRAM) devices. The developed SNN architecture is based on unsupervised Spike Time Dependent Plasticity (STDP) learning algorithm with online learning capability. Simulation results illustrate that the proposed CIM-based SNN architecture is highly energy-efficient with a consumption of ≈20 fJ per spike while maintaining state-of-the-art inference accuracy of 95%.

References

[1] Y. Sun *et al.*, "Deepid3: Face recognition with very deep neural networks," *arXiv*, 2015.

[2] D. A. Patterson, "Future of computer architecture," in *BARS*, 2006.

[3] H. A. Du Nguyen *et al.*, "Memristive devices for computing: Beyond cmos and beyond von neumann," in *VLSI-SoC*, 2017.

[4] M. Hu *et al.*, "Memristor-based analog computation and neural network classification with a dot product engine," *Advanced Materials*, 2018.

[5] H. A. D. Nguyen *et al.*, "A classification of memory-centric computing," *JETC*, 2020.

[6] M. Hu, H. Li *et al.*, "Hardware realization of bsb recall function using memristor crossbar arrays," in *DAC*, 2012.

[7] X. Qiao *et al.*, "Atomlayer: a universal reram-based cnn accelerator with atomic layer computation," in *DAC*, 2018.

[8] S. Gupta *et al.*, "Nnpim: A processing in-memory architecture for neural network acceleration," *Transactions on Computers*, 2019.

[9] F. Chen *et al.*, "Regan: A pipelined reram-based accelerator for generative adversarial networks," in *ASP-DAC*, 2018.

[10] Y. Wang *et al.*, "Energy efficient rram spiking neural network for real time classification," in *GLS-VLSI*, 2015.

[11] D. Ielmini, "Brain-inspired computing with resistive switching memory (rram): Devices, synapses and neural networks," *Microelectronic*, 2018.

[12] Y. Guo *et al.*, "Unsupervised learning on resistive memory array based spiking neural networks," *Neuroscience Frontiers*, 2019.

[13] T. Tang *et al.*, "Spiking neural network with rram: Can we use it for real-world application?" in *DATE*, 2015.

[14] A. Valentian *et al.*, "Fully integrated spiking neural network with analog neurons and rram synapses," in *IEDM*, 2019.

[15] A. Taherkhani *et al.*, "A review of learning in biologically plausible spiking neural networks," *Neural Networks*, 2020.

[16] A. L. Hodgkin *et al.*, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *Journal of physiology*, 1952.

[17] A. Delorme *et al.*, "Spikenet: A simulator for modeling large networks of integrate and fire neurons," *Neuro computing*, 1999.

[18] W. Gerstner *et al.*, *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[19] W. Gerstner, "A framework for spiking neuron models: The spike response model," in *Handbook of Biological Physics*, 2001.

[20] W. Gerstner *et al.*, *Spiking neuron models: Single neurons, populations, plasticity*, 2002.

[21] S. M. Bohte, "The evidence for neural information processing with precise spike-times: A survey," *Natural Computing*, 2004.

[22] J. H. Lee *et al.*, "Training deep spiking neural networks using backpropagation," *Frontiers in neuroscience*, 2016.

[23] R. Midya *et al.*, "Artificial neural network (ann) to spiking neural network (snn) converters based on diffusive memristors," *AEM*, 2019.

[24] B. Rueckauer *et al.*, "Conversion of analog to spiking neural networks using sparse temporal coding," in *ISCAS*, 2018.

[25] P. U. Diehl *et al.*, "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," in *ICRC*, 2016.

[26] E. Ledinauskas *et al.*, "Training deep spiking neural networks," *arXiv*, 2020.

[27] P. U. Diehl *et al.*, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in computational neuroscience*, 2015.

[28] Y. Hao *et al.*, "A biologically plausible supervised learning method for spiking neural networks using the symmetric stdp rule," *Neural Networks*, 2020.

[29] H. Fang *et al.*, "Brain inspired sequences production by spiking neural networks with reward-modulated stdp," *Frontiers in Computational Neuroscience*, 2021.

[30] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, 2014.

[31] S. Yu *et al.*, "Rram for compute-in-memory: From inference to training," *TCAS-I*, 2021.

[32] M. Zare *et al.*, "An area and energy efficient lif neuron model with spike frequency adaptation mechanism," *Neurocomputing*, 2021.

[33] D. Banerjee *et al.*, "Efficient optimized spike encoding of multivariate time-series," in *Neuro-Inspired Computational Elements*, 2022.

[34] M. M. Asl, "Propagation delays determine the effects of synaptic plasticity on the structure and dynamics of neuronal networks," 2018.

[35] L. Wang *et al.*, "Memristive lif spiking neuron model and its application in morse code," *Frontiers in Neuroscience*, 2022.

[36] C. Lee *et al.*, "Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning," *Neuroscience Frontiers*, 2018.

[37] S. Yu *et al.*, "Emerging memory technologies: Recent trends and prospects," *Solid-State Circuits Magazine*, 2016.

[38] M. Zhao *et al.*, "Crossbar-level retention characterization in analog rram array-based computation-in-memory system," *TED*, 2021.

[39] Z. Zhou *et al.*, "The characteristics of binary spike-time-dependent plasticity in hfo 2-based rram and applications for pattern recognition," *Nanoscale Research Letters*, 2017.

[40] H. Hazan *et al.*, "Bindsnet: A machine learning-oriented spiking neural networks library in python," *Frontiers in neuroinformatics*, 2018.

[41] Y. LeCun, "The mnist database of handwritten digits," *http://yann.lecun.com/exdb/mnist/*, 1998.

[42] J. Platkiewicz *et al.*, "A threshold equation for action potential initiation," *computational biology*, 2010.

[43] A. Hardtdegen *et al.*, "Improved switching stability and the effect of an internal series resistor in hfo 2/tio x bilayer reram cells," *TED*, 2018.

[44] S. Song *et al.*, "Competitive hebbian learning through spike-timing-dependent synaptic plasticity," *Nature neuroscience*, 2000.

[45] G. G. Turrigiano *et al.*, "Activity-dependent scaling of quantal amplitude in neocortical neurons," *Nature*, 1998.

[46] A. Morrison *et al.*, "Spike-timing-dependent plasticity in balanced random networks," *Neural computation*, 2007.

[47] R. Gütig *et al.*, "Learning input correlations through nonlinear temporally asymmetric hebbian plasticity," *Journal of Neuroscience*, 2003.

[48] V. Rossum *et al.*, "Stable hebbian learning from spike timing-dependent plasticity," *Journal of neuroscience*, 2000.

[49] D. J. Saunders *et al.*, "Locally connected spiking neural networks for unsupervised feature learning," *Neural Networks*, 2019.

[50] M. Meng *et al.*, "High-parallelism inception-like spiking neural networks for unsupervised feature learning," *Neurocomputing*, 2021.

[51] D. Lee *et al.*, "Various threshold switching devices for integrate and fire neuron applications," *Advanced Electronic Materials*, 2019.