On-chip learning of a domain-wall-synapse-crossbar-array-based convolutional neural network

Varun Bhavin Dave, Divya Kaushik, Janak Sharda and Debanjan Bhowmik

1 Department of Electrical Engineering, Indian Institute of Technology Delhi, New Delhi, Delhi 110016, India
2 Department of Electrical Engineering, Indian Institute of Technology Bombay, Mumbai, Maharashtra 400076, India

Abstract

Domain-wall-synapse-based crossbar arrays have been shown to be very efficient, in terms of speed and energy consumption, while implementing fully connected neural network algorithms for simple data-classification tasks, both in inference and on-chip-learning modes. But for more complex and realistic data-classification tasks, convolutional neural networks (CNN) need to be trained through such crossbar arrays. In this paper, we carry out device–circuit–system co-design and co-simulation of on-chip learning of a CNN using a domain-wall-synapse-based crossbar array. For this purpose, we use a combination of micromagnetic-physics-based synapse-device modeling, SPICE simulation of a crossbar-array circuit using such synapse devices, and system-level-coding using a high-level language. In our design, each synaptic weight of the convolutional kernel is considered to be of 15 bits; one domain-wall-synapse crossbar array is dedicated to the five least significant bits (LSBs), and two crossbar arrays are dedicated to the other bits. The crossbar arrays accelerate the matrix vector multiplication operation involved in the forward computation of the CNN. The synaptic weights of the LSB crossbar are updated after forward computation on every training sample, while the weights of the other crossbars are updated after forward computation on ten samples, to achieve on-chip learning. We report high classification-accuracy numbers for different machine-learning data sets using our method. We also carry out a study of how the classification accuracy of our designed CNN is affected by device-to-device variations, cycle-to-cycle variations, bit precision of the synaptic weights, and the frequency of weight updates.

1. Introduction

Neural-network algorithms are routinely and widely used now by the artificial intelligence and machine learning community for various tasks like data classification, recognition of objects in images, transcribing speech into text, showing news items and product advertisements on the user’s feed based on the user’s interests, etc [1]. In a traditional computer which follows the von Neumann architecture, the memory and computing units are physically separate, with a large part of the memory unit on a different chip compared to the computing/processing unit [2]. So, while running neural-network algorithms, a large amount of time and energy is spent in shuffling data, like the weight parameters of the neural network, between the computing unit and the off-chip memory unit. This high latency of memory fetches is known as the von Neumann bottleneck in literature [2, 3]. The matrix–vector multiplication (MVM) operation is one operation fundamental in all neural-network algorithms which uses frequent interaction between the memory and computing units and is, hence, subject to the von Neumann bottleneck.

In order to overcome this von Neumann bottleneck, a novel neuromorphic computing paradigm (or in-memory computing paradigm) has been proposed and implemented through crossbar arrays of non-volatile
Figure 1. (a) Schematic of the domain-wall-based synaptic device. $R_{\text{read}}$ is the resistance of the read path (MTJ structure) and $I_{\text{read}}$ is the read current. $R_{\text{write}}$ is the resistance of the write path (heavy-metal layer) and $I_{\text{write}}$ is the write current. (b) Snapshots of magnetic moments of the ferromagnetic free layer in the domain-wall device, as obtained from micromagnetic simulation and depicting the motion of domain wall upon the application of ‘write’ current (of magnitude $I$). All the snapshots are taken after a current pulse of magnitude $I$ and duration 3 ns have been applied on the device from the same initial condition: the domain wall is at the center of the device. In all the snapshots, the color corresponds to the out-of-plane component of magnetization ($m_z$). Dark blue means moments point into the plane, or $m_z = -1$ (vertically downward in (a)). Yellow means moments point out of the plane, or $m_z = 1$ (vertically upward in (a)). The region where the color changes from blue to yellow corresponds to the domain wall. The domain wall moves by distances in integral multiples of 17 nm with magnitudes of current pulses in integral multiples of 100 $\mu$A.

memory devices known as synaptic devices. In these analog crossbar arrays, memory and computing units are essentially intertwined, and hence they can implement the MVM operation much faster and with much more energy efficiency, when compared to traditional computing units [3–11].

Among these non-volatile devices, the ferromagnetic domain-wall device (figure 1(a)), essentially a spintronic device, has been shown to provide a faster and more energy-efficient crossbar solution, for on-chip learning (training of the neural network on hardware itself), compared to other non-volatile memory devices, like resistive random access memory (RRAM) devices and phase change memory (PCM) devices [12–18]. This is because the domain-wall device has a much more linear and symmetric conductance response, or synaptic characteristic, compared to that of the RRAM or PCM device [16]. On-chip learning is known to provide several advantages for edge devices in terms of data security and the like [6, 7]. This makes the domain-wall synapse device an important device in the context of the ongoing research on spintronics-based neuromorphic computing [19–26].

Thus far, simulation studies of on-chip learning on domain-wall-synapse-based crossbar arrays have been mostly restricted to non-spiking fully connected neural networks (FCNNs) and some simple spiking neural network architectures [14–17, 22, 23, 27–34]. A FCNN is good at solving simple data-classification and image-recognition problems and shows high classification accuracy for simpler data sets like Fisher’s Iris set, Wisconsin breast cancer set, and MNIST images [1, 15–17, 35, 36].

But more realistic tasks, for real applications, make use of much more complex images (e.g. the CIFAR-10 data-set [37]), videos, and speech. Convolutional neural networks (CNN) have been employed regularly for such tasks in recent days with much success [1, 3]. These CNNs have been typically implemented on
conventional computing units like CPU and GPU. So for higher-speed and lower-energy implementation, if crossbar arrays are to be used, then on-chip learning of crossbar-array-based CNN needs to be explored.

However, unlike in a FCNN, the operations involved in a CNN are much more involved [1, 3]. A FCNN uses a series of alternate MVM and non-linear activation function operations and can be completely implemented in analog through crossbar arrays and transistor-based analog peripheral circuits, as shown in Bhowmik et al [15] and Kaushik et al [17]. On the contrary, a CNN uses various stages of the convolution operation, which needs to be implemented as repeated MVM operations (more details in later sections), non-linear activation functions, pooling operations, and drop-out operations [1, 36, 38]. This makes it extremely challenging to implement a CNN on purely analog systems.

As a result, the implementation of inference (forward computation of pre-trained network on neuromorphic/in-memory computing hardware) and on-chip learning (training on the neuromorphic/in-memory computing hardware itself) of CNNs has been recently proposed and simulated on analog–digital–hybrid systems in [7, 39, 40]. Here, the MVM operation for convolution is carried out on crossbar arrays of synapse devices because the crossbar executes $O(n^2)$ multiplications and additions for a $n \times n$ matrix just in one computational step [39, 40]. This leads to much higher computational speed compared to conventional computers, where such massive parallelism is absent. The rest of the operations in the CNN is carried out in the digital unit, and ADCs and DACs are employed to convert between the analog and digital signals [7, 39, 40].

The crossbar designs in [39, 40] use memristive RRAM devices as synapses, while the crossbar design in [7] uses a combination of RRAM device and capacitor as a synapse. However, in this paper, we implement on-learning of a CNN on a domain-wall-synapse-based crossbar array instead (figure 4) since the domain-wall synapse is known to be a higher-speed and lower-energy alternative to the RRAM synapse [16]. But we use a similar design approach as in [7, 39, 40]: MVM of the convolution operation happens on the crossbar array, and rest of the operations on the digital unit (figures 4 and 5). We use here a modified version of the thresholding algorithm proposed in [17] to achieve the training.

We next present the layout of the paper, and through that, we show how we carry out device–circuit–system co-design in the paper. We also mention here how the different components of our device–circuit–system co-design are inter-connected. In section 2, we discuss our device-level modeling of the domain-wall synapse device on micromagnetic physics package ‘mumax3’ (figure 1) and show its conductance response to current pulses of different magnitudes, as appropriate for our CNN-training scheme (figure 3). In section 3, we discuss our CNN-training algorithm and how we have implemented our scheme on a crossbar array of domain-wall synapses (figure 4). We present our circuit-level SPICE simulation of the crossbar array; the conductance response obtained from device-level simulation in section 2 is used inside the circuit simulation of the crossbar array as a Verilog A module (figures 4 and 5). In section 4, we present our training results of the CNN for different data sets including the CIFAR-10 data set and analyze our results (figure 7). Beside providing classification accuracy numbers, we also comment on the speed and energy performance of the proposed CNN implementation. In section 6, we carry out a study of how the classification accuracy of our designed CNN is affected by device-to-device variations, cycle-to-cycle variations, bit precision of the synaptic weights, and the frequency of weight updates. We also discuss some recent experimental works connected to the domain-wall-synapse device and compare our simulation results with those experimental findings. Some peripheral circuits in our design are also provided. In section 7, we conclude the paper.

2. Micromagnetic simulation of the domain-wall-synapse device

2.1. Operating principle of the device

The three-terminal domain-wall synapse device used here is based on the heavy metal/ferromagnetic metal hetero-structure, which exhibits perpendicular magnetic anisotropy (PMA), as shown in figure 1(a) [12, 13, 15, 16, 26, 41–44]. When in-plane ‘write’ current flows through the heavy-metal layer (‘write’ path, from terminal T3 to T1 in figure 1(a)), the domain wall present in the ferromagnetic free layer above it moves due to spin orbit torque experienced by the magnetic moments of the free layer at the interface with the heavy metal. This changes the conductance of the device along the ‘read’ path (between terminal T2 and T1 in figure 1(a)) due to tunneling magneto-resistance (TMR) effect in the magnetic tunnel junction structure of the ‘read’ (ferromagnetic free layer/oxide/ferromagnetic fixed layer). More details on the operation of this device can be found in [15, 16].

2.2. Micromagnetic simulation parameters for the device

The domain wall considered here is of Neel chirality because of the presence of Dzyaloshinskii Moriya interaction (DMI) at the interface [41, 42, 45]. For such a Neel wall, it has been shown experimentally that in-plane current pulses can drive the domain wall even in the absence of an external magnetic field
For our synaptic device, we model a nanotrack based on the Pt (heavy metal)/CoFe (ferromagnetic metal)/MgO structure, with dimensions 1000 nm × 200 nm, on micromagnetic simulation package ‘mumax3’ [48] (figure 1(a)). The heavy-metal layer is of 10 nm thickness, and the ferromagnetic-metal layer of 1 nm thickness. Parameters used in our micromagnetic simulation are the same as that used in [45], where the model is benchmarked against experimentally obtained data. Thus, in our simulation, saturation magnetization ($M_s = 7 \times 10^5$ A m$^{-1}$), PMA constant ($K = 8 \times 10^3$ J m$^{-3}$), exchange correlation constant ($a = 1 \times 10^{-11}$ J m$^{-2}$), DMI $= 1.2 \times 10^{-3}$ J m$^{-2}$, and damping factor $= 0.3$.

We consider the heavy metal here to be Pt, so we take spin hall angle to be 0.07 in accordance with experimentally reported values [49–51]. The thickness of the heavy metal layer (10 nm) is higher than the spin diffusion length ($\approx 2-4$ nm) [52, 53]. Hence, the in-plane spin current density introduced into the free layer by the heavy metal layer ($J_s$) = in-plane charge current density ($J_c$) × spin hall angle. Dynamics of the magnetic moments of the ferromagnetic layer, in which the domain wall moves, has been simulated through micromagnetics here, under the influence of this spin current density ($J_s$) [54, 55]. Pt is chosen as the heavy metal here because of its low conductivity; this reduces the energy consumption per ‘write’ current pulse for conductance change/weight update and hence reduces overall energy consumption for on-chip learning on a crossbar array [17]. The antiferromagnetic layer present at either edge over the ferromagnetic layer pins the magnetic moments of the ferromagnetic layer over there and prevents the domain wall from getting destroyed at the edge [12, 13, 56].

2.3. Micromagnetic simulation results
In our micromagnetic simulation of the domain-wall device on ‘mumax3’, we start from the domain wall initially being at the center of the device (figure 1(b)). The domain wall located at the extreme left edge of the device corresponds to minimum conductance ($G_{\min}$) of the MTJ, since the moments of the fixed and the free layer are oriented anti-parallel to each other in this configuration. The domain wall located at the extreme right edge corresponds to the maximum conductance of the device ($G_{\max}$), owing to the parallel orientation of the moments of the fixed and the free layer in this configuration. A positive-polarity current pulse causes the domain wall to move toward the right edge of the device, leading to the conductance/weight increase. A negative-polarity current pulse causes the domain wall to move toward the left edge, leading to the conductance/synaptic weight decrease.

We will see in the next section that the algorithm we use to train our CNN involves increasing or decreasing each synaptic weight by integral multiples of a fixed positive value. So in this section, we show through micromagnetics that such kind of conductance modulation is indeed possible in our domain-wall device.

From our micromagnetic simulation of the device, we observe that when a current pulse of magnitude 100 $\mu$A and duration 3 ns is applied, the corresponding spin current density ($J_s$) lets the magnetic moments evolve over time such that the domain wall moves by approximately 17 nm (figure 1(b)). For current pulses of 3 ns duration and a magnitude which is a multiple of 100 $\mu$A ($n \times 100$ $\mu$A), the domain wall moves by a distance which is again a multiple of 17 nm ($n \times 17$ nm). For example, when a current pulse of magnitude 200 $\mu$A is applied for 3 ns, then the domain wall moves by approximately 34 nm.

This happens because the velocity of the domain wall is linearly proportional to the charge current density, and in turn the ‘write’ current magnitude (if cross-sectional area through which the in-plane current flows does not change, as is the case here), for a wide range of current density/‘write’ current magnitude. This has been reported experimentally in [41, 42, 47] and verified through micromagnetic simulations and one-dimensional domain-wall theory in [46]. We observe this in our micromagnetic simulation as well; for a given pulse duration (3 ns), we apply current pulses of different magnitudes and record the distances moved by the domain wall. Ratio of the domain-wall position to the pulse duration yields the domain-wall velocity. We plot the domain-wall velocity as a function of current density in figure 2 of the revised manuscript and show that the domain-wall velocity is indeed a linear function of the ‘write’ current magnitude. The analytical expression corresponding to this linear dependence has been obtained through one-dimensional domain-wall theory and reported in [46].

The tilting of the domain wall with the application of current, as observed in figure 1(b), has also been explained through the one-dimensional domain-wall theory in [46]. For the MTJ structure considered here, based on R–A product of $4.04 \times 10^{-12}$ ohm m$^{-2}$ and TMR ratio of 120% [57], $G_{\min} \approx 2.9 \times 10^{-3}$ mho and $G_{\max} \approx 6.1 \times 10^{-3}$ mho. Since the lowest distance moved by the domain wall here is 17 nm (figure 1(b)) and the motion always happens in integral multiples of 17 nm, number of conductance states in the device $= 60$. So the device is considered to store five bits of synaptic weight. Lowest change in conductance = $\frac{G_{\max} - G_{\min}}{50} \approx 5.33 \times 10^{-3}$ mho. Thus, conductance change in the ‘read’ path of the device (MTJ) structure in figure 1(a) occurs in the multiples of $5.33 \times 10^{-5}$ mho.

We embed this information in a Verilog A module for the device [58]. When we simulate a circuit component corresponding to this module on Cadence Virtuoso circuit simulator, we observe that conductance
In our micromagnetic simulation, ‘write’ current pulses of different magnitudes but the same duration (3 ns) are applied, and the distances moved by the domain wall are obtained. Ratio of the domain-wall position to the pulse duration yields the domain-wall velocity, which is plotted as a function of the current magnitude.

Simulation of the circuit component for the synapse on SPICE, using the Verilog A code based on the device simulation of figure 1. Conductance of the ‘read’ path of the device is observed in response to a ‘write’ current waveform (figure 1). Conductance changes as multiples of $5.33 \times 10^{-5}$ mho corresponding to current pulses with magnitudes in multiples of 100 $\mu$A, due to motion of the domain wall by distances in multiples of 17 nm (figure 1(b)). Positive current pulses increase the conductance, while negative current pulses decrease the conductance. All aforementioned simulation parameters (both for ‘mumax3’ and Verilog A) and their respective values have been presented again in table 1.
Table 1. Different simulation parameters and their respective values, used for our work.

| Simulation parameters                     | Values used  |
|------------------------------------------|--------------|
| Saturation magnetization ($M_s$)         | $7 \times 10^5$ A m$^{-1}$ |
| PMA constant ($K$)                       | $8 \times 10^5$ J m$^{-3}$ |
| PMA constant for triangular notch regions| $9 \times 10^5$ J m$^{-3}$ |
| DMI                                      | $1.2 \times 10^{-3}$ J m$^{-2}$ |
| Damping factor                           | 0.3          |
| Exchange correlation constant ($A$)       | $1 \times 10^{-11}$ J m$^{-1}$ |
| Resistance-area product                   | $\approx 10^{-11}$ Ω m$^2$ |
| TMR ratio                                 | 120%         |
| Minimum conductance ($G_{min}$)           | $2.9 \times 10^{-3}$ mho |
| Maximum conductance ($G_{max}$)           | $6.1 \times 10^{-3}$ mho |

Figure 4. (a) Voltages corresponding to a portion ($3 \times 3 \times 3$) of an image the CIFAR10 data set are applied as input to the crossbar. (b) 64 different convolutional filters with distinct weights are used in the case of the CIFAR10 data set. These filters correspond to the first convolutional layer. (c) The MSB crossbar of domain-wall synapses for this set of filters stores the MSB part of the weights, which are updated column-wise after every 50 iterations of the training algorithm. (d) The LSB crossbar for this set of filters, which stores the LSB part of the weights, and is updated in every iteration of the training algorithm. Circuits $X$ and $Q$ associated with every domain-wall synapse help in weight-update calculation: $X$ is the multiplier, and $Q$ thresholds the input. Common part of the weight update fed back by the output nodes (red dashed lines) is not thresholded in our algorithm.

3. Circuit-and-system-level simulation of the CNN with domain-wall synapses

3.1. Forward computation

Forward computation needs to be implemented on the proposed neuromorphic hardware both in the inference mode (the CNN is first trained on a conventional computer and then only the forward computation of the trained CNN on new/test data is implemented on the neuromorphic hardware) and in the on-chip learning mode (the CNN is both trained and tested both on the neuromorphic hardware). Among the different operations in forward computation, the convolution operation needs to be implemented on the domain-wall-synapse-based crossbar array, as mentioned in section 1. Figure 4 shows our implementation scheme for the same.

Following the method shown in [7], we load different convolutional filters onto different crossbar arrays and apply different segments of the input images on them. Different parts of the outputs are computed, some
of them are summed, and then they are reconstructed to obtain the convolution. Since each synaptic weight in the convolutional kernel is of 15 bits in our algorithm (this helps us achieve the desired level of accuracy, as we discuss in the next section), we dedicate the five least significant bits (LSBs) to a LSB crossbar and the other ten bits to two MSB crossbars (figures 4(c) and (d)). We have three separate crossbars because each domain-wall synapse device can only store five bits of weight, as explained in the previous section.

As shown in figure 4(a), when the input images are in RGB mode (e.g., the images in CIFAR-10 data set [37]), one portion of each input image (of dimensions $3 \times 3 \times 3$) is extracted at a time (figure 4(a)). Then voltages corresponding to these $27$ $(3 \times 3 \times 3)$ pixel intensities are applied on all the three aforementioned crossbars simultaneously. For CIFAR-10 data set, 64 different convolutional filters, each of $3 \times 3 \times 3$ dimensions, are used. Each convolutional filter represents a distinct feature. $w_i^1$, $v_i^1$ and $u_i^1$ denote the synaptic weights of the convolutional kernels, where $i$ represents the convolutional kernel number (ranges from 1 to 64), $j$ represents the position in the kernel matrix (ranges from 1 to 9), and $u$, $v$, or $w$ correspond to R, G, or B (figure 4(b)).

For the first convolutional kernel, inputs $x_1$, $y_1$ and $z_1$ (of the input image) are multiplied with $w_i^1$, $v_i^1$, and $u_i^1$ respectively, $x_2$, $y_2$ and $z_2$ are multiplied with $w_i^2$, $v_i^2$, and $u_i^2$ respectively, and so on. The synaptic weights corresponding to the first kernel are shown in red font in figure 4: $u_i^1$, $v_i^1$, $w_i^1$, $u_i^2$, $v_i^2$, $w_i^2$, $u_i^3$, $v_i^3$, $w_i^3$. These multiplications are carried out in our designed crossbar array along just one vertical bar (the leftmost bar). Conductance of each domain-wall synapse is proportional to each synaptic weight; so currents flowing through the synapses are proportional to the product of each input value and each weight. When these currents add up at the leftmost vertical bar following Kirchhoff’s current Law, we essentially implement:

$$\text{out}^1 = (x_1u_1^1) + (y_1v_1^1) + (z_1w_1^1) + (x_2u_2^1) + (y_2v_2^1) + (z_2w_2^1) + \cdots (x_9u_9^1) + (y_9v_9^1) + (z_9w_9^1).$$ \tag{1}

All these multiplications taking place simultaneously provide the speed advantage for the crossbar array compared to conventional computing units.

Similarly, the second vertical column of the crossbar array, from the left, corresponds to the second convolutional kernel (its weights shown in violet font in figure 4) and executes the following multiplication:

$$\text{out}^2 = (x_1u_1^2) + (y_1v_1^2) + (z_1w_1^2) + (x_2u_2^2) + (y_2v_2^2) + (z_2w_2^2) + \cdots (x_9u_9^2) + (y_9v_9^2) + (z_9w_9^2).$$ \tag{2}

The other 62 vertical bars of the crossbar array take care of the other 62 convolutional kernels. It is to be noted that these aforementioned multiplication operations happen simultaneously on the three crossbar arrays mentioned earlier: five MSBs of weights are handled in the first MSB crossbar (figure 4(c)), the next five bits of weights are handled in the second MSB crossbar, and the five LSBs of weights are handled in the LSB crossbar (figure 4(d)).

At the end of these simultaneous operations on the three crossbar arrays, convolution of the first segment from the image (figure 4(a)) with the weight kernel is accomplished. Then the voltages corresponding to the next segment of the image are applied on the crossbar, and so on. Thus unlike in the implementation of forward computation in a FCNN on a crossbar array, where the entire image is loaded into the crossbar at one go, the image is loaded in segments into the crossbar array for a CNN. Voltages corresponding to only one segment are applied at a time on the horizontal bars of the crossbar array in figures 4(c) and (d), then voltages corresponding to the next segment of the image, and so on. Further operations on the output signals obtained from the crossbar, corresponding to the forward computation in the CNN, are carried out on the digital unit of the system, after conversion from analog to digital using an ADC of 15 bit precision.

In our work, we simulate the aforementioned analog, convolution operations on three crossbar arrays (two for MSB, one for LSB) which we design on Cadence Virtuoso SPICE circuit simulator. Verilog A modules for the domain-wall synapses, based on the micromagnetic simulation of the devices on `mumax3` (figures 1 and 3), are inserted in our circuit design on SPICE, as mentioned in section 1. After every iteration, the outputs generated by the circuit simulator are then passed on to a code we write on a high-level language (Python) which takes care of the digital operations in the forward computation as well as the weight-update calculation. The conductance values of the Verilog A synaptic modules in the SPICE crossbar design are updated in accordance with the weight-update calculation, and the same process is repeated for other samples to carry out training/learning of the CNN. Thus, we carry out device (`mumax`)—circuit (SPICE)—system (Python) co-design and co-simulation in this work to simulate on-chip learning of a CNN (figure 5). We discuss our training algorithm in details next.
3.2. Training

In the training algorithm that was used to train the FCNN in [17], thresholding function \( f \) has been applied both on the inputs and common part of weight update, calculated at the output node of each layer, where:

\[
    f(x) = \begin{cases} 
    \theta, & \forall x > q_{\text{thresh}} \\
    0, & -q_{\text{thresh}} \leq x \leq q_{\text{thresh}} \\
    -\theta, & \forall x < -q_{\text{thresh}} 
    \end{cases}
\]

(\( \theta \) and \( q_{\text{thresh}} \) are hyper-parameters of the network). (Refer to [17] for more details).

But thresholding both the input and common part of weight update in an extremely deep network like the CNNs used here leads to a significant drop in the classification accuracy. In fact, for CIFAR-10 data set, we observe that the CNN does not even get trained at all for such heavy thresholding. So in the modified algorithm used here, meant to train CNNs on crossbar arrays, we have thresholded the input with the aforementioned function \( f \), but not the common part of weight update. We discuss our weight-update rule for training next.

As mentioned earlier, any particular weight in the convolutional kernel is split into three different weight strings, of five bits each, and stored on three different synapse devices, corresponding to the three crossbars. Let \( W_M \) be the decimal number corresponding to the leftmost five bits (five MSBs), \( W_N \) be the decimal number corresponding to the middle five bits, and \( W_L \) be the decimal number corresponding to the rightmost five bits (five LSBs). \( W \) is the entire weight string of 15 bits in decimal.

In our modified training algorithm, only the LSB part of the weight (\( W_L \)) is updated after every iteration. From the stochastic gradient descent (SGD) algorithm [1], this weight update is supposed to be:

\[
    W_L \leftarrow W_L - \alpha \left( \frac{\partial L}{\partial W} \right),
\]

where \( L \) is the loss function computed for the network after one cycle of forward computation (corresponding to one sample in the training set), \( \alpha \) is the learning rate, and the symbol \( \leftarrow \) means value transfer/update.

But since the weight is quantized (up to 15 bits), we cannot perform an update of arbitrary precision. Moreover, this update needs to be thresholded as well, since the LSB-synapse device can only store weight values between \(-W_{L}^{\text{max}}\) and \(W_{L}^{\text{max}}\). So, after approximating the update \( \frac{\partial L}{\partial W} \) to the nearest quantization level, we can write the update as:

\[
    W_L \leftarrow \text{thresh}(W_L - kW_{L}^{\text{min}}),
\]

where \( k \) is the rounded-off integer of \( \frac{\partial L}{\partial W} \), and \( W_{L}^{\text{min}} \) is the minimum non-zero value that can be stored in the LSB-synapse device (decimal equivalent of one bit).

The middle part of the weight is updated only once every ten iterations (one iteration corresponds to one sample in the training set):

\[
    W_N \leftarrow W_N + \delta W_N.
\]

If \( W_L \) has reached its maximum value, \( W_L \) is updated to \(-W_{L}^{\text{max}}\) and \( \delta W_N \) is set to \( W_{N}^{\text{min}}\). Similarly, if \( W_L \) has reached its lowest value, \( \delta W_N \) is set to \(-W_{N}^{\text{min}}\), and \( W_L \) is updated to \( W_{L}^{\text{max}}\). Unless either of these two scenarios occurs, \( \delta W_N \) takes the value of zero. \( W_{N}^{\text{min}} \) is the minimum positive value that \( W_N \) can take. If \( W_N \) reaches its maximum or minimum value, then \( W_N \) is updated the same way; otherwise, \( W_M \) is not updated.

Since the weights of the LSB crossbar need to be updated after every iteration, a dedicated weight-update circuitry needs to be present with every domain-wall synapse just like in the synapse cell design of the FCNN crossbar array in [17]. Just like in [17], the input needs to be multiplied with the common part of weight update generated at the output node of that stage. However, more transistors will be needed here than that needed in the synapse cell of [17] (only two were needed there) because in [17], both the input and the common
Figure 6. All operations inside the forward computation that we carry out on the Python code, including the ones that need to be computed on the digital unit of the final neuromorphic hardware, are shown here. More details on dropout, zero padding, and non-linear activation functions (SeLu and Softmax) can be found in [1, 38, 60, 62].

Table 2. Final classification accuracy numbers of the CNN, following our thresholding algorithm, for different data sets have been tabulated here, along with total energy consumed and energy consumed per sample for weight updates during the training process across all the crossbar arrays in the CNN. The five LSBs of the weights have been updated after every sample/cycle/iteration, while the other bits have been updated every ten samples/cycles/iterations.

| Data set         | Number of images | Train set | Test set | Train accuracy (%) | Test accuracy (%) | Energy (mJ)  | Energy per sample (J) |
|------------------|------------------|-----------|----------|--------------------|-------------------|-------------|-----------------------|
| MNIST            | 50 000           | 10 000    | 100      | 100                | 99.28             | 0.108       | 2.106 × 10⁻⁹          |
| Fashion MNIST    | 60 000           | 10 000    | 100      | 100                | 91.66             | 0.936       | 1.87 × 10⁻⁸           |
| CIFAR10          | 50 000           | 10 000    | 100      | 100                | 74.5              | 3.767       | 7.5 × 10⁻⁸            |

part of weight update are thresholded by the aforementioned function $f$ and hence can only take three values $(\theta, 0, -\theta)$. On the contrary, here, the input is thresholded but the common part of weight update is not thresholded.

We do not use a dedicated update circuit with each synapse of the two MSB crossbars because the weights there are updated only once in ten iterations. In this case, the weight update of each synapse can be calculated in the digital computing unit. Then the synapses can be accessed sequentially, and their weights can be updated by passing appropriate current pulses and moving the domain walls.

4. Training results

4.1. Performance: classification accuracy

Following the device–circuit–system co-simulation method discussed in the previous section and shown in figure 5, we carry out on-chip learning of the CNN on three popular machine-learning data sets: MNIST [35], Fashion MNIST [59] and CIFAR-10 [37]. All operations inside the forward computation that we carry out on the Python code, including the ones that need to be computed on the digital unit of the final neuromorphic hardware, are shown in figure 6.

Classification accuracy as a function of training-epoch number is plotted in figure 7. The final accuracy numbers along with information on the number of images used in the training and test set in each case have been provided in table 2. The performance metrics obtained here are at par with conventional, state-of-the-art methods for the MNIST and Fashion MNIST dataset [36, 60, 61].

For CIFAR-10 data set, our classification accuracy on the test set (test accuracy) is comparable to that reported for a conventional implementation of a CNN on a GPU in [38] and slightly lower than a similar conventional implementation in [60]. But our accuracy number is comparable to that reported for neuromorphic implementation on an analog–digital hybrid system, which uses a combination of a RRAM device and a capacitor as a synapse [7].

Possible ways of improving the test accuracy of our implementation further and making it comparable to that used for training a CNN on a conventional computer are as follows:
Figure 7. Classification accuracy (%) of the CNN in figure 6 shown as a function of training epochs for different data sets: (a) MNIST, (b) Fashion MNIST, (c) CIFAR10. The five LSBs of the weights have been updated after every sample/cycle/iteration, while the other bits have been updated every ten samples/cycles/iterations.

(a) We use a simple optimization technique, SGD, here to minimize the loss function (equation (3)). Using a more complex optimizer like the Adam optimizer [63], which has a momentum term in the weight update, may yield a better result.

(b) We use a batch size of 1 here for the LSBs of the synaptic weights, i.e., we update the LSBs of the weights after forward computation on every sample of the training set. A large batch size may yield better results, but a large batch size also means weight updates for several iterations need to be added up and the sum needs to be stored in the digital unit. This implies more digital memory consumption.

(c) In a conventional CPU or GPU implementation, the weights can have a precision of 32–64 bits. But here, since each domain-wall device can store five bits only and we use three crossbars, we are limited to weight resolution of just 15 bits. This reduces the accuracy number for our algorithm. Increasing bit resolution will increase the classification accuracy. Similarly, lowering bit resolution below 15 will reduce the accuracy significantly, as we show in the next section.
While the test accuracy is ≈ 75%, the classification accuracy on the training set (training accuracy) is 100%. This may be a consequence of over-fitting. Introducing regularization techniques like ‘dropout’ in our proposed hardware can reduce over-fitting [38].

4.2. Performance: training time per sample
The training time can be calculated in terms of the time taken for the most fundamental operation in the crossbar: the vector–matrix operation. In fact, the very reason to use crossbar arrays is that it makes the vector–matrix multiply operation much faster than digital implementations. If we assume the time taken for one such operation to be \( t_0 \), then the time for one complete forward computation can be calculated based on the architecture of the CNN. For any particular layer of the CNN, if the output shape is given by \( n_H \times n_W \times n_C \), then the time taken to compute this output is \( t_0 n_H n_W \) for our design, since all the computations for a particular output pixel (all channels taken together) are computed in parallel, as depicted in figure 4. Moreover, the time taken to calculate the output from a fully connected layer is simply \( t_0 \), since only one vector–matrix multiplication takes place. Thus, the time taken for one forward computation can be estimated by adding the product \( n_H \times n_W \) for the output of different layers. For the case of our architecture, using this method, the delay is calculated to be nearly \( 2.7 \times 10^3 t_0 \).

Since the computations which occur in the backward pass (gradient calculation part) are of the same nature as the forward part (the outer-product calculation needs to be done on the crossbar array), the time taken for the backward pass will also be close to this figure. Thus, the total time for one sample (forward and backward pass) can be estimated to be nearly \( 5.4 \times 10^3 t_0 + \) delay during weight update. Calculating \( t_0 \) and the delay during weight update, however, is challenging, and we discuss possible ways to do that in the next section.

4.3. Performance: training energy per sample
Next, the total energy for carrying out the weight updates during the training process, across all the crossbar arrays in the designed CNN, is calculated. This is a significant contributor to the total energy and total power consumed for on-chip learning. Figure 3 shows that to change the weight by the smallest amount in our synapse device, say one quantum (for which conductance changes by \( 5.33 \times 10^{-5} \) mho), a ‘write’ current pulse of magnitude 100 \( \mu \)A is required. For a change of \( n \) quanta, a current pulse of magnitude \( n I \) needs to be applied (as mentioned before), since the conductance change is linearly proportional to the input current. Thus, the energy dissipated in this weight update is given by (6):

\[
E = (nI)^2 R_{\text{write}} \Delta t,
\]

where \( R_{\text{write}} \) is the resistance of the write path of the particular synapse, and \( \Delta t \) is the length of the write pulse.

For our domain-wall device (shown in figure 1, \( R_{\text{write}} = 50 \Omega \)) for the given length and breadth of the device (1000 nm \( \times \) 200 nm), under the assumption that majority of the current flows through the heavy-metal (Pt) layer. This is a valid assumption because the resistivity of the ferromagnetic layer (CoFe say) is about 170 \( \mu \Omega \) cm [49], which is much higher than that of Pt (\( \approx 10 \mu \Omega \) cm [64]). Also the thickness of the ferromagnetic layer is 1 nm only, but the thickness of the heavy-metal layer is 10 nm. So the ferromagnetic-layer path is much more resistive than the heavy-metal-layer path, and majority of the current flows through the latter. Duration of the ‘write’ current pulse (\( \Delta t \)) = 3 ns, as mentioned before (figure 3).

In figure 8(a), the energy consumed per training sample for all weight updates, across all the crossbar arrays in the designed CNN, is plotted as a function of training epochs for different data sets (equation (6) has been used for the calculation). As training is achieved, weight update goes down and energy consumption per epoch goes down. For the most challenging data set, CIFAR10, the highest amount of energy is consumed, as expected.

A similar trend is observed in figure 8(b) where the cumulative energy (energy for the current epoch plus all previous epochs) is plotted as a function of epochs. The final cumulative energy, which is the total write energy consumed for the training process, is also mentioned in figure 8(b) for all the data sets. This energy includes training over all the samples in the training set. Dividing the energy by the number of training samples yields energy per sample consumed for weight updates across the crossbar arrays in the CNN, which we tabulate in table 2. Once the energy is divided by the training time, the corresponding power consumption can be calculated.

It is to be noted that energy consumed in forward computation has not been considered in this calculation (figure 8). Typically, the ‘read’ current is several orders of magnitude smaller than the ‘write’ current because the ‘read’ current is not supposed to move the domain wall from its position unlike the ‘write’ current. So the energy consumed per input–vector–weight–matrix multiplication on the crossbar array in the forward computation can be considered negligible compared to that for weight update. But unlike in a FCNN, there’s not just
Figure 8. (a) Energy consumed per training sample for all weight updates, across all the crossbar arrays in the designed CNN, is plotted as a function of training epochs for different data sets. (b) Cumulative energy consumed for such weight updates as a function of number of epochs. Here the energy for all samples in the data set has been added up. Total energy consumed for the entire training is also mentioned for each data set.

one input-vector-weight-matrix multiplication on the crossbar array corresponding to a weight-update operation in the case of a CNN. In a CNN, the same crossbar array is used several times for input-vector-weight-matrix multiplication between the weight kernel and several segments of the image (as explained in section 3) before weight update of the synapses in the crossbar array takes place once. Once the energy consumed in all these input-vector-weight-matrix multiplications during the forward computation is added up, the net energy in forward computation can be comparable to the energy consumed for weight update which we have calculated and plotted in figure 8. Estimating the energy consumed for forward computation on the crossbar array is a part of our future work.

5. Discussion

In this section, we study the dependence of the classification accuracy of our designed CNN on process/device-level variations, cycle-to-cycle variations due to write noise, bit precision, and the number of samples/cycles after which the other bits of weight are updated (the five LSBs are always updated every sample/cycle). Since CIFAR-10 is the most challenging of the three data sets used for this work (as mentioned earlier), we carry out this study for CIFAR-10 data set only. For the other data sets, our accuracy numbers are already above 90% and will not be affected much by the aforementioned variations.
5.1. Process/device-to-device variations

It has been shown in [65, 66] that in spin-transfer-torque magnetic random access memory devices (STTMRAM), process variations like variations in material parameters (saturation magnetization, PMA, damping factor) as well as variations in device dimensions lead to more write errors. Similar phenomenon will occur in domain-wall devices, which have the same material stack and similar properties as the STTMRAM devices.

Thus, process variation can lead to the following kind of device variation: we apply ‘write’ current pulses on the domain-wall synapses, but for a given pulse, the actual motion of the domain wall and hence the actual change in the weight/conductance is different from what’s expected by a certain factor. This factor is constant for all weight changes of a particular synaptic device but varies from device to device. At the same time, it does not vary from cycle to cycle unlike in the case of ‘write’ noise which we discuss next.

Following this principle, we include random errors in synaptic-weight changes in our algorithm for CNN training. We carry out Monte Carlo simulations for the same, considering different extents of device-level variations from their expected values: 4%, 8%, 12%, 16%, and 20%. We use the CIFAR-10 data set for this purpose and plot our results in figure 9.

In all cases, we observe that the classification accuracy is not affected (figure 9). This may be due to the negative feedback mechanism in the learning process. Learning is achieved by minimizing the global error of the network, obtained by comparing the actual output with the desired output. As long as the error decreases, the network trains itself independent of the exact values of the weights. Also, in physical implementation of
the CNN through crossbars, the same crossbar will be used for forward computation (vector–matrix multiplication) and also for feedback (the outer product calculation in the back-propagation algorithm during weight update). Thus the error that can creep in due to variations in the synaptic weight values during the forward computation can get nullified when the same weight values are used for the outer product calculation during the weight update [17, 40].

Thus our study shows that device-to-device conductance/weight variation up to 20% does not have much of an impact on the classification accuracy. Whether in the actual experimental implementation of the proposed crossbar array of domain-wall devices, the conductance variation will be limited to 20% or not cannot be possibly concluded at the moment. Since the domain-wall device is a relatively new device and less commercial device compared to the STTMRAM device, such study has not been reported yet for the domain-wall device to the best of our knowledge. So this will be the subject of our future study.

5.2. Cycle-to-cycle variations
The Monte Carlo simulation we carry out to account for cycle-to-cycle variations is similar to that we carry out for device-to-device variations. But the key difference is that in the case of device-to-device variations, the error in weight update of one particular synapse device is constant throughout the training process, though it may be different from that of another synapse in the crossbar array. In the case of cycle-to-cycle variations, the error in weight update of any synapse device changes randomly from cycle to cycle, to take into account noise in the ‘writing’ process of the domain-wall device (‘write’ noise).

Based on these simulations, we plot how cycle-to-cycle variations affect the classification accuracy for CIFAR-10 data set in figure 10. We observe that much like in the device-to-device variation case, cycle-to-cycle variation does not affect the classification accuracy much (figure 10). One possible reason for this is again the negative-feedback nature of the error minimization/learning process. Another possible reason is that cycle-to-cycle variation/‘write’ noise does not affect the MSBs of the synaptic weight much (since the MSBs are not updated every cycle but are updated after a fixed number of cycles).

5.3. Bit precision of synaptic weights
We next study how the classification accuracy varies with bit precision of the synaptic weights in the CNN. Since CIFAR-10 is the most complex data set we handle in this paper, we show our results for the CIFAR-10 data set only like we did for the cases of device-to-device and cycle-to-cycle variations. Classification-accuracy-vs-epoch plots for different values of bit precision of weights are shown in figure 11.

We observe that to obtain a high classification accuracy, we need a precision of around 15 bits. This happens because a CNN uses an extremely complex network architecture with several layers of weight-matrix and input vector multiplication, along with several other added functions: non-linear activation functions, max pool, etc. The classification task we want to accomplish is also quite difficult: images are taken from the CIFAR-10 data set. If the bit precision of the synaptic weights is low, information is lost at every stage of the CNN and the accuracy suffers.

However, for simpler network architectures like FCNN and simpler classification tasks, having only 4–8 bits per synapse is sufficient to obtain high classification accuracy. This has been shown in [17, 67].

5.4. Frequency of weight update
In the results presented thus far, we update the middle parts/middle bits of the synaptic weights after every ten cycles/iterations (five LSBs are updated every cycle). Figure 12 shows how the classification accuracy changes for the CIFAR-10 data set for different weight-update frequencies (different number of cycles/iterations after which the middle part of the weight is updated). We observe that if the number of iterations (after which the middle part of the weight is updated) is increased, accuracy goes down which supports intuition.

Thus, choosing to carry out weight update after fewer iterations is a good idea from the accuracy perspective, but it also increases the number of write pulses applied on the synaptic devices corresponding to the middle part of the weight and so increases the power and energy consumption for on-chip learning. Also, as mentioned before, the crossbar corresponding to the five LSBs has a dedicated weight-update circuitry, and so all the weights can be updated simultaneously. On the contrary, in the crossbars corresponding to the other parts of the synaptic weights, the weights can be updated only sequentially using the peripheral digital unit. So reducing the number of cycles for update of the middle parts of weights, or increasing the frequency of weight update, may lead to significant delay in training.

5.5. Experimental calibration
Having presented classification accuracy results for our domain-wall-synapse-based crossbar arrays through device–circuit–system co-design, it is imperative that we compare the behavior of our simulated domain-wall device with recent experimental results. The physics of in-plane-current-driven domain-wall motion in
heavy-metal/ferromagnetic-metal hetero-structure-based devices has been thoroughly studied before through various experiments, as we have mentioned in section 2 [41, 42, 45, 47]. As also mentioned before, the values of our micromagnetic simulation parameters (table 1) are the same as that used in another micromagnetic study carried out by Emori et al [45], which is bench-marked against experiments in similar devices.

The domain-wall velocity we obtain from our micromagnetic simulation is plotted as a function of in-plane ‘write’ current in figure 2. The linear variation of domain-wall velocity with current matches qualitatively with that reported in experiments [41, 42, 47]. Since the width of the device is 200 nm and the thickness of the heavy-metal layer is 10 nm, current density through the heavy metal layer can be obtained from the data on the current in figure 2. After that conversion, it can be observed that for the same value of current density, the domain-wall velocity obtained in our simulation is close to (in the same order of magnitude) as that reported experimentally by Lo Conte et al [47]. For example, for current density of $4 \times 10^{11}$ A m$^{-2}$, Lo Conte et al experimentally report the velocity of the domain wall to be 15 m s$^{-1}$. In our micromagnetic simulation, for 800 $\mu$A of in-plane ‘write’ current which translates to $4 \times 10^{11}$ A m$^{-2}$, the domain-wall velocity we obtain is 40 m s$^{-1}$, which is in the same order of magnitude as the aforementioned experimentally observed value.

However, while there are several experimental reports on such current-driven domain-wall motion, integrating a magnetic tunnel junction with the domain-wall device and studying its conductance response through TMR effect of the tunnel junction have become a topic of experimental research very recently, with the emerging possibility of using such a device as a synapse in a crossbar array for neuromorphic computing.

Figure 10. Classification accuracy (a) train accuracy (b) test accuracy) of the CNN in figure 6 shown as a function of training epochs for CIFAR10 data set taking cycle-to-cycle variations into account.
Lequeux et al experimentally demonstrated current-based control of multiple conductance states (measured through TMR) of such a tunnel-junction device [68]. Zhang et al, very recently, experimentally demonstrated current-controlled multiple conductance states in a chain of MTJ-s (as opposed to a single domain-wall device) and showed that the chain can act as a synapse unit for neural-network crossbars [23].

In the conductance response of our domain-wall device as shown in figure 3, the domain wall moves in multiples of 17 nm in response to current pulses with magnitudes in multiples of 100 μA and duration 3 ns. Such precise domain-wall motion has been experimentally demonstrated, very recently, by Zhang et al [69] and by Kumar et al [70].

5.6. Design of peripheral circuits

Figure 4(d) shows how the thresholding/quantizer (Q) and multiplier (X) circuits have been used in the crossbar array for update of the five LSBs of weights after every iteration. The design of the thresholding and the multiplier circuit can be seen in figure 13. Both circuits have been designed using transistor-based amplifiers; more details on the design and their input-vs-output relationships, as obtained through SPICE simulations, can be found in [17, 18]. The thresholding circuit’s design is shown in [17], and the multiplier circuit’s design is shown in [18].

The multiplier circuit in [17], however, cannot be used for this purpose since it contains two transistors only and can only carry out the multiplication when both the input and the common part of the weight update...
Figure 12. Classification accuracy ((a) train accuracy (b) test accuracy) of the CNN in figure 6 shown as a function of training epochs for CIFAR10 data set, while we vary the number of cycles/iterations after which the middle parts/middle bits of the synaptic weight are updated (five LSBs are updated every cycle).

are thresholded before the multiplication, which is the case in [17] but not in this work. Thresholding both the input and the common part of the weight update lead to significant reduction in classification accuracy since we train a CNN here unlike a FCNN (which is trained in [17]). So a more complicated multiplier circuit as shown in [18] needs to be used here (figure 13(b)).

The major delay during the weight update is caused by the multiplier and the thresholding circuit (the domain-wall motion within the synaptic device happens just within 3 ns, as mentioned before). The speed of these circuits ultimately depends on the switching time of the transistors used and the parasitic resistances and capacitances. Carrying out rigorous SPICE simulations to determine this delay time is a part of our future work; the focus of the current paper is on the modification of the existing CNN training algorithm in accordance with the requirements of domain-wall-synapse-based crossbar arrays.

The time for input-vector–weight-matrix multiplication ($t_0$ in subsection 4.2) is also difficult to calculate at the moment because similar delays due to parasitic resistances and capacitances occur there as well. Multiplier and thresholding circuits do not contribute to the delay here, but the ADC and the DAC (which interface between the digital unit and the crossbar) can cause significant delay. Calculating this $t_0$ will be a part of our future work. Together, write-time delay and $t_0$ determine the time taken for training, and thus the power consumed for training.
6. Conclusion

Thus, in this paper, we carry out device–circuit–system co-design and co-simulation of on-chip learning of a CNN using a domain-wall-synapse-based crossbar array. We use a combination of micromagnetic-physics-based synapse-device modeling (on micromagnetic package ‘mumax3’), SPICE simulation of a crossbar-array circuit using such synapse devices (on Cadence Virtuoso SPICE simulator), and system-level-coding using a high-level language (Python) for this purpose. We report very high classification accuracy numbers for data sets like MNIST and Fashion MNIST and fairly high classification accuracy numbers for a more complex data set like CIFAR-10.

We have also shown that device-to-device variations and cycle-to-cycle variations do not affect the classification accuracy. We also show that increasing the bit precision of the synaptic weights and increasing the weight-update frequency of the middle part of the synaptic weight (five LSBs are updated after every sample/cycle) improve the classification accuracy significantly.

Acknowledgments

This research was supported by Ministry of Human Resource Development, India, under the STARS scheme (MoE-STARS/STARS-1/304).

Data availability statement

All data that support the findings of this study are included within the article (and any supplementary files).

ORCID iDs

Debanjan Bhowmik https://orcid.org/0000-0003-1136-8778

References

[1] LeCun Y, Bengio Y, Bengio Y and Hinton G 2015 Deep learning Nature 521 436
[2] Wulf W and McKee S A 1996 Hitting the memory wall: implications of the obvious ACM SIGARCH Comput. Archit. 23 20–4
[3] Musisi-Nkambwe M, Afshari S, Barnaby H, Kozicki M and Sanchez Esqueda I 2021 The viability of analog-based accelerators for neuromorphic computing: a survey Neuromorph. Comput. Eng. 1 012003
[4] Tsai H, Ambrogio S, Narayanan P, Shelby R M and Burr G W 2018 Recent progress in analog memory-based accelerators for deep learning J. Phys. D: Appl. Phys. 51 283001

Figure 13. (a) Design of the thresholding circuit, as also shown in [17] (b) design of the multiplier circuit, as also shown in [18].
Neuromorph. Comput. Eng. 2 (2022) 024006

V B Desai et al

[5] Sebastian A, Le Gallo M, Burr G W, Kim S, BrightSky M and Eleftheriou E 2018 Tutorial: brain-inspired computing using phase-change memory devices J. Appl. Phys. 124 111101

[6] Chakraborty I, Ali M, Ankit A, Jain S, Roy S, Sridharan S, Agrawal A, Raghunathan A and Roy K 2020 Resistive crossbars as approximate hardware building blocks for machine learning: opportunities and challenges Proc. IEEE 108 2276–310

[7] Luo Y and Yu S 2020 Accelerating deep neural network in situ training with non-volatile and volatile memory based hybrid precision synapses IEEE Trans. Comput. 69 1113–27

[8] Burr G W, Narayanan P, Shelby R M, Sidler S, Boybat I, di Nolfo C and Leblebici Y 2015 Large-scale neural networks implemented with non-volatile memory as the synaptic weight element: comparative performance analysis (accuracy, speed, and power) IEDM Tech. Dig.

[9] Wang W, Song W, Yao P, Li Y, Van Nostrand J, Qiu Q, Jelmini D and Yang J 2020 Integration and co-design of memristive devices and algorithms for artificial intelligence iScience 23 12

[10] Covi E, Donati E, Liang X, Kappel D, Heidari H, Payyand M and Wang W 2021 Adaptive extreme edge computing for wearable devices Front. Neurosci. 15 5

[11] Li H et al 2021 Memristive crossbar arrays for storage and computing applications Adv. Intell. Syst. 3 2100017

[12] Sengupta A and Roy K 2016 A vision for all-spin neural networks: a device to system perspective IEEE Trans. Circuits Syst. I 63 2267–77

[13] Sengupta A, Shim Y and Roy K 2016 Proposal for an all-spin artificial neural network: emulating neural and synaptic functionalities through domain wall motion in ferromagnets IEEE Trans. Biomed. Circuits Syst. 10 1152–60

[14] Sengupta A and Roy K 2017 Encoding neural and synaptic functionalities in electron spin: a pathway to efficient neuromorphic computing Appl. Phys. Rev. 4 044105

[15] Bhownik D, Saxena U, Dankar A, Verma A, Kaushik D, Chatterjee S and Singh U 2019 On-chip learning for domain wall synapse based fully connected neural network J. Magn. Magn. Mater. 489 165271

[16] Kaushik D, Singh U, Sahu U, Sreedevi I and Bhownik D 2020 Comparing domain wall synapse with other non volatile memory devices for on-chip learning in analog hardware neural network AIPT Adv. 10 025111

[17] Kaushik D, Sharda J and Bhownik D 2020 Synapse cell optimization and back-propagation algorithm implementation in a domain wall synapse based crossbar neural network for scalable on-chip learning Nanotechnology 31 364004

[18] Dey N, Sharda J, Saxena U, Kaushik D, Singh U and Bhownik D 2019 On-chip learning in A conventional silicon MOSFET based analog hardware neural network IEEE Biomedical Circuits and Systems Conf. (BioCAS) pp 1–4

[19] Roy K et al 2015 Exploring spin transfer torque devices for unconventional computing IEEE J. Emerg. Sel. Top. Circuits Syst. 5 5–16

[20] Zhang D, Hou Y, Wang C, Chen J, Zeng L and Zhao W 2018 Hardware acceleration implementation of sparse coding algorithm with spintronic devices IEEE/ACM Int. Symp. Nanoscale Architectures (NANOARCH) pp 1–7

[21] Guo Z et al 2021 Spintronics for energy-efficient computing: an overview and outlook Proc. IEEE 109 1398–417

[22] Zhang D et al 2016 All spin artificial neural networks based on compound spintronic synapse and neuron IEEE Trans. Biomed. Circuits Syst. 10 828–36

[23] Zhang K et al 2022 High on/off ratio spintronic multi-level memory unit for deep neural network Adv. Sci. 2022 109

[24] Grollier J, Querlioz D and Stiles M D 2016 Spintronic nanodevices for bioinspired computing Proc. IEEE 104 2024–39

[25] Grollier J, Querlioz D, Camsari K Y, Everschor-Sitte K, Fukami S and Stiles M D 2020 Neuromorphic spintronics Electron. 3 360–70

[26] Saxena U, Kaushik D, Bansal M, Sahu U and Bhowmik D 2018 Low-energy implementation of feed-forward neural network with back-propagation algorithm using a spin–orbit torque driven skyrmionic device IEEE Trans. Magn. 54 1–5

[27] Sahu U, Pandey A, Goyal K and Bhowmik D 2019 Spike time dependent plasticity (STDP) enabled learning in spiking neural networks using domain wall based synapses and neurons IEEE Adv. 9 125339

[28] Akinola O, Hu X, Bennett C H, Marinella M, Friedman J S and Incorvia J A C 2019 Three-terminal magnetic tunnel junction synapse circuits showing spike-time-dependent plasticity J. Phys. D: Appl. Phys. 52 494010

[29] Bennett C H, Hassan N, Hu X, Incorvia J A C, Friedman J S and Marinella M J 2019 Semi-Supervised Learning and Inference in Domain-Wall Magnetic Tunnel Junction (DW-MTJ) Neural Networks Spintronics XI vol 11090 (International Society for Optics and Photonics)

[30] Alamdar M et al 2021 Domain wall-magnetic tunnel junction spin–orbit torque devices and circuits for in-memory computing Appl. Phys. Lett. 118 112401

[31] Misba W A, Kaais T, Bhattacharya D and Atulasimha J 2021 Voltage-controlled energy-efficient domain wall synapses with stochastic distribution of quantized weights in the presence of thermal noise and edge roughness IEEE Trans. Electron Dev. 69 1638–60

[32] Misba W A, Lozano M, Querlioz D and Atulasimha J 2021 Energy efficient learning with low resolution stochastic domain wall synapse based deep neural networks (arXiv:2111.07284)

[33] Wang H, Kang W, Pan B, Zhang H, Deng E and Zhao W 2021 Spintronic computing-in-memory architecture based on voltage-controlled spin–orbit torque devices for binary neural networks IEEE Trans. Electron Dev. 68 4944–50

[34] Bhattacharya T, Li S, Huang Y, Wang C, Wang W, Zhao W and Suri M 2019 Low-power (1T1N) skyrmionic synapses for spiking neuromorphic systems IEEE Access 7 5034–44

[35] LeCun Y 1998 MNIST database https://yann.lecun.com/exdb/mnist/

[36] Lecun Y, Bottou L, Bengio Y and Haffner P 1998 Gradient-based learning applied to document recognition Proc. IEEE 86 2278–2324

[37] Krizhevsky A 2012 The cifar-10 dataset https://www.cs.toronto.edu/~kriz/cifar.html

[38] Krizhevsky A, Sutskever I and Hinton G E 2012 ImageNet classification with deep convolutional neural networks Proc. IEEE 100 2276–310

[39] Ankit A, Jain S, Roy S, Sridharan S, Agrawal A, Raghunathan A and Roy K 2020 Resistive crossbars as approximate hardware building blocks for machine learning: opportunities and challenges Proc. IEEE 108 2276–310

[40] Ankit A et al 2019 PUMA: a programmable ultra-efficient memristor-based accelerator for machine learning inference Proc. 24th Int. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS’19) pp 715–31

[41] Ankit A et al 2020 PANTHER: a programmable architecture for neural network training harnessing energy-efficient ReRAM IEEE Trans. Comput. 69 1128–42

[42] Ryu K-S, Thomas I, Yang S-H and Parkin S 2013 Chiral spin torque at magnetic domain walls Nat. Nanotechnol. 8 527

[43] Bhownik D, Nowakowski M E, You L, Lee O, Keating D, Wong M, Bokor J and Sahlehddin S 2015 Deterministic domain wall motion orthogonal to current flow due to spin orbit torque Sci. Rep. 5 11823
[44] Miron I M et al 2011 Fast current-induced domain-wall motion controlled by the Rashba effect Nat. Mater. 10 419
[45] Emori S, Martínez E, Lee K-J, Lee H-W, Bauer U, Ahn S-M, Agrawal P, Bono D C and Beach G S D 2014 Spin Hall torque
magnetometry of Dzyaloshinskii domain walls Phys. Rev. B 90 184427
[46] Martínez E, Emori S, Perez N, Torres L and Beach G S D 2014 Current-driven dynamics of Dzyaloshinskii domain walls in the
presence of in-plane fields: full micromagnetic and one-dimensional analysis J. Appl. Phys. 115 213909
[47] Lo Conte R et al 2015 Role of B diffusion in the interfacial Dzyaloshinskii–Moriya interaction in Ta/Co90Fe8B10/MgO nanowires 
Phys. Rev. B 91 014433
[48] Vansteenkiste A, Leliært J, Dvornik M, Helsen M, García-Sanchez F and Van Waeyenberge B 2014 The design and verification of
mumax3 AIP Adv. 4 107133
[49] Liu L, Fa C-F, Li Y, Tseng H W, Ralph D C and Buhrman R A 2012 Spin–torque switching with the giant spin Hall effect of 
tantalum Science 336 555–8
[50] Liu L, Q, Moriyama T, Ralph D C and Buhrman R A 2011 Spin–torque ferromagnetic resonance induced by the spin Hall effect 
Phys. Rev. Lett. 106 036601
[51] Liu L, Lee O J, Gudmundsen T J, Ralph D C and Buhrman R A 2012 Current-induced switching of perpendicularly magnetized
magnetic layers using spin torque from the spin Hall effect Phys. Rev. Lett. 109 096602
[52] Berger A, Edwards E R J, Nembach H T, Karis O, Weiler M and Silva T J 2018 Determination of the spin Hall effect and the spin 
diffusion length of Pt from self-consistent fitting of damping enhancement and inverse spin–orbit torque measurements Phys. 
Rev. B 98 024202
[53] Qu D, Huang S Y, Miao B F, Huang S X and Chien C L 2014 Self-consistent determination of spin Hall angles in selected 5D
metals by thermal spin injection Phys. Rev. B 89 140407
[54] Khvalkovskiy A et al 2009 High domain wall velocities due to spin currents perpendicular to the plane Phys. Rev. Lett. 102 067206
[55] Chantlhoukula A et al 2011 Vertical-current-induced domain-wall motion in MgO-based magnetic tunnel junctions with low 
current densities Nat. Phys. 7 626–30
[56] Bromberg D M, Moneck M T, Sokalski V M, Zhu J, Pileggi L and Zhu J G 2014 Experimental demonstration of four-terminal
magnetic logic device with separate read- and write-paths IEDM Tech. Dig.
[57] Zhu J-G and Park C 2006 Magnetic tunnel junctions Mater. Today 9 36
[58] Hu X, Timm A, Brigner W H, Incorvia J A C and Friedman J S 2014 SPICE-only model for spin-transfer torque domain wall MTJ
logic IEEE Trans. Electron Dev. 66 2817–21
[59] Xiao H, Rasul K and Vollgraf R 2017 Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms 
arXiv:1708.07747
[60] Cireşan D, Meier U and Schmidhuber J 2012 Multi-column deep neural networks for image classification Proc. IEEE Conf. 
Computer Vision and Pattern Recognition vol 1 pp 3642–9
[61] Jarrett K, Kavukcuoglu K, Ranzato M A and LeCun Y 2009 What is the best multi-stage architecture for object recognition? Proc. 
IEEE Int. Conf. Computer Vision
[62] Klambauer G, Mayr T U A and Hochreiter S 2017 Self-normalizing neural networks Advances in Neural Information Processing 
Systems (NeurIPS) pp 972–81
[63] Kingma D P and Ba J 2015 Adam: a method for stochastic optimization Int. Conf. Learning Representations
[64] Lequeux S, Sampaoi J, Cros V, Yakushi K, Fukushima A, Matsumoto R, Kubota H, Yuasa S and Grolier J 2016 A magnetic
synapse: multilevel spin–torque memristor with perpendicular anisotropy Sci. Rep. 6 31510
[65] Zhang S et al 2019 A spin–orbit–torque memristive device Adv. Electron. Mater. 5 1800782
[66] Kumar D, Jing C H, JianPeng C, Jin T, Ter L, Shia R and Piramanayagam S N 2020 Ultra-low power domain wall device for
spin-based neuromorphic computing (arXiv:2007.12357)