MTJ-Based Hardware Synapse Design for Quantized Deep Neural Networks

Tzofnat Greenberg-Toledo, Ben Perach, Daniel Soudry, and Shahar Kvatinsky, *Senior Member, IEEE*

Abstract—**Quantized neural networks (QNNs) are being actively researched as a solution for the computational complexity and memory intensity of deep neural networks. This has sparked efforts to develop algorithms that support both inference and training with quantized weight and activation values without sacrificing accuracy. A recent example is the GXNOR framework for stochastic training of ternary and binary neural networks. In this paper, we introduce a novel hardware synapse circuit that uses magnetic tunnel junction (MTJ) devices to support the GXNOR training. Our solution enables processing near memory (PNM) of QNNs, therefore can further reduce the data movements from and into the memory. We simulated MTJ-based stochastic training of a TNN over the MNIST and SVHN datasets and achieved an accuracy of** 98.61% **and** 93.99%**, respectively.**

Index Terms—**Memristor, Magnetic Tunnel Junction, Deep Neural Networks, Quantized Neural Networks, MRAM.**

I. INTRODUCTION

DEEP neural networks (DNNs) are the state-of-the-art solution for a wide range of applications, such as image and natural language processing. The classic DNN approach requires frequent memory accesses and is compute-intensive, *i.e.*, it requires numerous multiply and accumulate (MAC) operations. For example, computing the first fully connected layer of VGG16 [1], which has 8192×4096 synapses, requires 32 million MAC operations, while storing and accessing 128MB of weights. As such, DNN performance is limited by computing resources and power budget. Therefore, efforts have been made to design dedicated hardware for DNNs [2–4]. Those solutions support training with high resolution such as 32-bit floating point. Still, DNN models are power hungry and not suited to run on low-power devices.

Ternary neural networks (TNNs) and binary neural networks (BNNs) are being explored as a way to reduce the computational complexity and memory footprint of DNNs. By reducing the weight resolution and activation function precision to quantized binary $\{-1, 1\}$ or ternary $\{-1, 0, 1\}$ values, the MAC operations are replaced by much less demanding logic operations, and the number of required memory accesses is significantly reduced. Such networks are also known as *quantized neural networks* (QNNs) [5]. This insight triggered recent research efforts to design novel algorithms that can support binary and/or ternary DNNs without sacrificing

T. Greenberg-Toledo, Ben Perach, Daniel Soudry, and S. Kvatinsky are with the Andrew and Erna Viterbi Faculty of Electrical Engineering, Technion – Israel Institute of Technology, Haifa, Israel, 3200003, e-mail: {stzgrin@tx},{daniel.soudry@ee},{shahar@ee}.technion.ac.il

accuracy. Recently, the GXNOR algorithm for training such networks was proposed [6]. This algorithm uses a stochastic update function to facilitate the training phase. Unlike other algorithms [5, 7, 8], GXNOR does not require keeping the full value (*e.g.*, in a floating point format) of the weights and activations. Hence, GXNOR enables further reduction of the memory capacity during the training phase.

Emerging memory technologies such as Spin-Transfer Torque Magnetic Tunnel Junction (STT-MTJ) can be used to design dedicated hardware to support in-situ DNN training, with parallel and energy efficient operations. Furthermore, the near-memory computation enabled by these technologies reduces overall data movement. The MTJ is a binary device, with two stable resistance states. Switching the MTJ device between resistance states is a stochastic process, but this property limits the use of STT-MTJ device as a memory cell. Previous works have used the stochastic behavior of the STT-MTJ, or other memristive technologies such as Resistive RAM (RRAM), to implement hardware accelerators for BNNs [9–12]. In [9], the focus was on the architecture level of BNN accelerators, but without supporting online training. Other works have implemented hardware for bio-inspired artificial neural networks (ANNs), using the spike-timing-dependent plasticity (STDP) training rule [10, 11]. Although STDP is widely used for bio-inspired ANN, common DNNs are trained with gradient-based optimization such as stochastic gradient descent (SGD) and adaptive moment estimation (ADAM) [13]. A recently proposed MTJ-based binary synapse composed of a single transistor and a single MTJ device (1T1R) [12] supports QNNs with binary weights still using real values to represent the activations. The implementation requires two update operations to execute the SGD updates. Using real-valued activation will require digital-to-analog converters, which will increase the area and power consumption of the proposed solution.

In this paper, we explore the stochastic behavior of the MTJ and use it to support GXNOR training. We leverage the stochastic process to support the GXNOR algorithm. We propose a four-transistor two-MTJ (4T2R) circuit for a ternary stochastic synapse and a two-transistor single-MTJ (2T1R) circuit for a binary stochastic synapse, where the intrinsic stochastic switching behavior of the MTJ is used to perform the GXNOR stochastic update function. Such a design enables highly parallel and energy efficient accurate in-situ computation. Our designed synapse can support various DNN optimization algorithms, such as SGD and ADAM, which are used regularly in practical applications.

We evaluated the TNN and BNN training using the MTJ-based synapse with PyTorch over the MNIST [14] and

SVHN [15] data-sets, where the circuit parameters were extracted from SPICE simulations using a GlobalFoundries 28nm FD-SOI process. Our results show that using the MTJ-based synapse for training yielded similar results as the ideal GXNOR algorithm, with a small accuracy loss of 0.7% for the TNN and 2.4% for the BNN. Moreover, the proposed hardware design is energy efficient, achieving $18.3\frac{TOPs}{W}$ for feedforward and $3\frac{TOPs}{W}$ for weight update.

This paper makes the following contributions:

- Exploit the MTJ stochastic properties to support QNN stochastic training based on the GXNOR framework. We showed that PNM of stochastic QNN training is possible using the MTJ-based synapse with small accuracy reduction.
- Suggested an MTJ-based ternary and binary synapse circuits. Those circuits 1) exploit the stochastic switching of the MTJ device to support stochastic weight update algorithm, 2) support in-situ weights update. The proposed solution can support standard optimization algorithms such as SGD and ADAM without reading the weight data out of the synapse array. 3) support near memory processing of the feedforward, and backpropagation computations.

The rest of the paper is organized as follows. In Sections II and III, background on MTJ and the GXNOR algorithm is given. Section IV describes the proposed MTJ-based ternary synapse. In Section V, we explain how to modify the proposed circuits to support BNNs. In Section VI, the proposed circuits are evaluated for their ability to support TNN training and their energy efficiency. We conclude in Section VII.

II. GXNOR Algorithm

In recent years, efforts have been made to make DNN models more efficient and hardware compatible. Compression methods have been explored, where the DNN weights and activation functions are constrained to discrete values such as binary $\{-1, 1\}$ or ternary $\{-1, 0, 1\}$. The MAC operations in TNNs and BNNs are replaced with the simpler XNOR logic operations, and the memory footprint of the network is reduced dramatically. The GXNOR algorithm is a framework for constraining the weights and activations to the quantized space while training the QNN [6]. This section describes the GXNOR framework and focuses on the framework's differences compared to regular DNN training.

Quantized Weights and Activations: The quantized space is defined by

$$Z_N = \{z_N^n | z_N^n = (\frac{n}{2^{N-1}} - 1), n = 0, 1..., 2^N\}, \quad (1)$$

where N is a non-negative integer which defines the space values and $z_N^n \in [-1, 1]$. For example, the binary space is given for $N = 0$ and the ternary space for $N = 1$. The quantized space resolution, the distance between two adjacent states, is given by

$$\Delta z_N = \frac{1}{2^{N-1}} \quad (2)$$

Feedforward: The quantized activation is a step function, where the number of steps is defined by the space. To support backpropagation through the quantized activations, the derivative of the activation function is approximated. In this work we used a simple window function which replaces the ideal derivative, given by a sum of delta functions. The window function is given by

$$\frac{\partial \varphi_r(x)}{\partial x} = \begin{cases} \frac{1}{2a}, & if \ r - a \leq x \leq r + a, \\ 0, & others \end{cases} \quad (3)$$

where r, a are positive parameters, defining the sparsity of the neurons and the window in the neighborhood of x, respectively. φ_r is the discrete activation function. Using the approximated derivative, the backpropagation algorithm, thus the error value, compute with no further changes.

Weight Update: To support training with weights constrained to the discrete weight space (DWS), the GXNOR algorithm uses a stochastic gradient-based method to update the weights. First, a boundary function is defined to guarantee that the updated value will not exceed the $[-1, 1]$ range. The boundary function is

$$\varrho(\Delta W_{ij}^l(k)) =$$
$$\begin{cases} min(1 - W_{ij}^l(k), \Delta W_{ij}^l(k)), & if \ \Delta W_{ij}^l(k) > 0, \\ max(-1 - W_{ij}^l(k), \Delta W_{ij}^l(k)), & else \end{cases} \quad (4)$$

where W_{ij}^l is the synaptic weight between neuron j and neuron i of the following layer $(l + 1)$, ΔW_{ij}^l is the gradient based update value, and k is the update iteration. Then, the update function is

$$W_{ij}^l(k + 1) = W_{ij}^l(k) + \Delta w_{ij}^l(k), \quad (5)$$

where $\Delta w_{ij}^l(k) = \mathcal{P}\left(\varrho(\Delta W_{ij}^l(k))\right) \in \mathbb{Z}$ is the discrete update value, obtained by projecting $\varrho(\Delta W_{ij}^l(k))$ to a quantized weight space. $\mathcal{P}(\varrho)$ is a probabilistic projection function defined by

$$\mathcal{P}(\varrho) = \begin{cases} \kappa_{ij}\Delta z_N + sign(\varrho)\Delta z_N, & w.p. \ \tau(\nu_{ij}), \\ \kappa_{ij}\Delta z_N, & w.p. \ 1 - \tau(\nu_{ij}), \end{cases} \quad (6)$$

where κ_{ij} and ν_{ij} are, respectively, the quotient and the remainder values of $\varrho(\Delta W_{ij}(k))$ divided by Δz_N, and

$$\tau(\nu) = tanh\left(m \cdot \frac{|\nu|}{\Delta z_N}\right), \tau(\nu) \in [0, 1], \quad (7)$$

where m is a positive adjustment factor. Hence,

$$\Delta w_{ij}^l = \kappa_{ij}\Delta z_N + sign(\nu_{ij})Bern(\tau(\nu_{ij}))\Delta z_N, \quad (8)$$

where $Bern(\tau(\nu_{ij}))$ is a Bernoulli variable with parameter $\tau(\nu_{ij})$.

In this work, we focus on TNN and BNN. The binary weight space (BWS) is given by $N = 0$ and $\Delta z_0 = 2$. The ternary weight space (TWS) is given by $N = 1$ and $\Delta z_1 = 1$. Figure 1 illustrates examples of TNN and BNN weight update for $W = -1$ and $W = 0$.
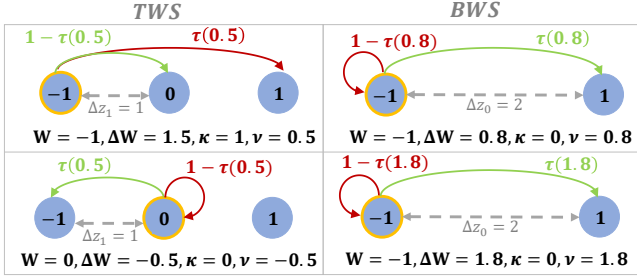
Fig. 1. TNN examples (TWS): Ternary weight update with $\Delta z = 1$. Given $W = -1$ and $\Delta W = 1.5$, $\kappa = 1$, $\nu = 0.5$ and the discrete update value is $\Delta w^l = 1 + Bern(\tau(0.5))$. For $W = 0$ and $\Delta W = -0.5$, $\kappa = 0$, $\nu = -0.5$ and the discrete update value is $\Delta w^l = -Bern(\tau(0.5))$. BNN examples (BWS): Binary weight update with $\Delta z = 2$. For $W = -1$ and $\Delta W = 0.8$, $\kappa = 0$, $\nu = 0.8$ and the discrete update value is $\Delta w^l = Bern(\tau(0.8))$. Given $W = -1$ and $\Delta W = 1.8$, $\kappa = 0$, $\nu = 1.8$ and the discrete update value is $\Delta w^l = Bern(\tau(1.8))$.

III. MAGNETIC TUNNEL JUNCTION

An MTJ device is composed of two ferromagnetic layers, a fixed magnetization layer and a free magnetization layer, separated by an insulator layer. The resistance of the device is defined by the relative magnetization of the free layer as compared to the fixed layer. A parallel magnetization state (P) leads to low resistance (R_{on}) and an anti-parallel state (AP) leads to high resistance (R_{off}). The device resistance can be switched by the current flow through the device. The switching probability of the MTJ device depends on the current pulse, when three work regimes are defined: 1) low current, 2) intermediate current, and 3) high current [11]. For fast switching time, this work focuses on the high current regime, where the current I is substantially higher than the critical current I_{c_0}, given by

$$I_{c_0} = \frac{2|e|}{\hbar} \frac{\alpha V (1 \pm P)}{P} \mu_0 M_s \frac{M_{eff}}{2}, \quad (9)$$

where α, M_s, V, P, M_{eff} are, respectively, the Gilbert damping, the saturation magnetization, the free layer volume, the spin polarization of the current, and the effective magnetization [16]. The switching time is

$$\tau = \frac{2}{\alpha \gamma \mu_0 M_s} \frac{I_{c_0}}{I - I_{c_0}} log\left(\frac{\pi}{2|\theta|}\right), \quad (10)$$

where γ is the gyromagnetic ratio, and θ is the initial magnetization angle [16], given by a normal distribution $\theta \sim \mathcal{N}(0, \theta_0)$, $\theta_0 = \sqrt{k_B T / (\mu_0 H_k M_s V)}$, where H_k is the shape anisotropy field.

A. Stochastic Weight Update Using MTJ

The MTJ conductivity represents the quantized synapse weight (see Section IV-A for details). Therefore, we control the MTJ switching process to support the GXNOR weight update. The update is done in the high current domain to guarantee fast update operation. To switch the MTJ device a voltage pulse, V_{up} is dropped over the device, for time interval, Δt. The resulted current direction sets the switching probability. Using

equation (10) and the voltage pulse, the switching probability of the MTJ is given by

$$P_{sw} = P(\Delta t > \tau) = 1 - erf\left(\frac{\pi}{2\sqrt{2}\theta_0 \exp\left(\frac{\Delta t V_{up}}{CR}\right)}\right), \quad (11)$$

where $C = \frac{2I_{c_0}}{\alpha \gamma \mu_0 M_s}$, and R is the resistance of the device.

IV. MTJ-BASED TERNARY SYNAPSES

In this section, we describe the proposed ternary synapse circuit to support stochastic GXNOR training. We leverage the stochastic behavior of the MTJ device to support the stochastic update function. In the following section (Section V), we explain how the proposed synapse can support binary weights as well.

A. Proposed Synapse Circuit and Synapse Array

Synapse Circuit: The schematic of the proposed ternary synapse is shown in Figure 2a. The ternary synapse is composed of two MTJ devices connected together in their fixed layer port. The free layer port of each MTJ is connected to two access transistors. This synapse is similar to our previous work [17, 18], where the RRAM is replaced by the MTJ device, and two synapse structure are added together to support the ternary weight. Compare to [17, 18], which supports full precision weight values, the following section describes how this designed is optimized to support quantized weights.

Synapse Weight: Table I lists the different values of the synapse weight, W. This weight is defined and stored as the combination of the two MTJs resistances. The zero state in our ternary synapse has two representations, as opposed to one in a regular ternary synapse. Moreover, thanks to the bi-stability of the MTJ, the proposed synapse value is limited to $\{-1, 0, 1\}$; thus, the boundary function in (4) is enforced by the hardware synapse.

Synapse Array: The synapse circuit described in Fig 2a is the basic cells of an array structure, as shown in Figure 2b. The synapses are arranged in a $M \times N$ array, where each synapse is indexed by (n, m). Each synapse in column $n \in [1, N]$ is connected to four inputs $\{u_{n1}, \bar{u}_{n1}, u_{n2}, \bar{u}_{n2}\}$ where all the input voltages are shared among all synapses in the same column. Likewise, each synapse in row $m \in [1, M]$ is connected to control signals $\{e_{m(1,n)}, \bar{e}_{m(1,p)}, e_{m(2,n)}, \bar{e}_{m(2,p)}\}$. The control signals are shared among all synapses in the same row. The synapse located in (m, n) produce an output current I_{mn} which contrib to current through output row m. The operations on the synapse array are done in the analog domain, accumulated according to Kirchoff's current law (KCL), where the GXNOR output is represented by the current.

B. Training TNN

1) Gated XNOR and Feedforward: To perform the gated-XNOR logic operation between the synapse and activation values [6], we denote the input neuron values as the voltage sources. The logic values $\{-1, 0, 1\}$ are represented by $u \in \{-V_{rd}, 0, V_{rd}\}$, where V_{rd} is set to guarantee the low
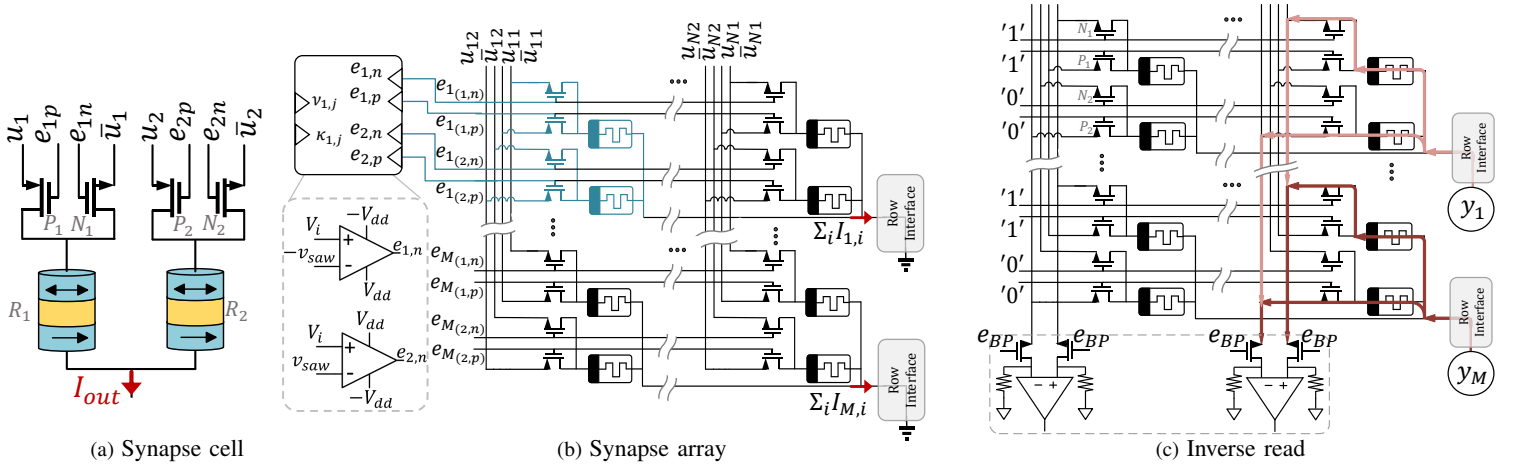
Fig. 2. (a) Schematic of the 4T2R MTJ-based synapse. (b) The synapse array. The MTJ symbol is replaced by the general memristor symbol. A single synapse cell is marked in blue. The control signals are generated using a voltage comparator. (c) The data flow of the inverse read operation.

TABLE I
TERNARY SYNAPSE STATES AND OUTPUT CURRENT

Weight	R_1	R_2	I_{out}
1	R_{on}	R_{off}	$\frac{R_{off}-R_{on}}{R_{off}R_{on}}u$
0_s	R_{off}	R_{off}	0
0_w	R_{on}	R_{on}	0
-1	R_{off}	R_{on}	$-\frac{R_{off}-R_{on}}{R_{off}R_{on}}u$

current regime of an MTJ, so the switching probability is negligible. During this operation, $u_1 = u$ and $\bar{u}_2 = -u$ are connected and the synapse output node is grounded. The result is the output current sign,

$$I_{out} = (G_1 - G_2)u, \qquad (12)$$

where $G_{\{1,2\}}$ are the conductance of the two MTJs. As listed in Table I, the polarity of I_{out} depends on the input voltage and the synapse weight. If $u = 0$ or $W = \{0_w, 0_s\}$, the output current is $I_{out} \approx 0$. If the weight and input have the same polarity, then $sign(I_{out}) = 1$ else $sign(I_{out}) = -1$.

To perform feedforward with the GXNOR operation, we need to compute

$$O_m = \sum_{n=1}^{N} GXNOR(w_{mn}, a_n), \forall m \in [1, M] \qquad (13)$$

where O_n is row n result. To that aim, each column voltage is mapped to the corresponding input activation ($u_m = V(a_m)$ $\forall m \in [1, M]$) and the array row outputs are connected to ground potential and the output currents from all synapses are summed based on KCL. Thus, the current through row i is

$$I_{row,i} = \sum_{j=1}^{N} (G_{ij,R_1} - G_{ij,R_2})u_j = \\ \frac{R_{off} - R_{on}}{R_{off}R_{on}} (N_{+1,i} - N_{-1,i})V_{rd}, \qquad (14)$$

where $G_{j,R_{\{1,2\}}}$ is the conductivity of each MTJ, N is the number of synapses per row, $N_{+1,i}$ is the total number of positive products in row i, and $N_{-1,i}$ is the total number of negative products in row i.

2) Synapse Weight Update: Unlike updating regular DNN, the proposed synapse supports the quantized update scheme suggested in [6]. The synapse weight update is done in the high current domain, guaranteed by the update input voltage V_{up} for the update period marked by T_{up}. T_{up} is set to guarantee that if $\Delta t = T_{up}$ then Eq. (11) results $P_{sw} = P(T_{up}) \approx 1$. As mention in Section III-A, the weight update is influenced by the current direction and the time interval in which the current flows through the MTJs.

To update the ternary synapse weight we need to update both MTJ devices while supporting the GXNOR update scheme. We used the control signals to set the current direction and the voltage pulse time interval. We consider two update cases: 1) supporting general optimization algorithms, and 2) supporting the SGD algorithms.

Support of General Optimization Algorithms: To support general optimization algorithms the update value, ΔW is computed outside the synapse array. The array columns are updated iteratively, *i.e.*, a single synapse array column is updated at each iteration. During this operation, the input voltages are set to $u_1 = u_2 = V_{up} > 0$ for all the synapse columns. To support the probabilistic projection (Section II), the control signals are given by

$$\begin{cases} e_{1,p} = -e_{2,p} = -sign(\Delta W_{ij})V_{dd}, & if\ \kappa_{ij} \neq 0 \\ e_{1,p} = e_{2,p} = V_{dd}, & else \end{cases} \qquad (15)$$

$$e_{1,n} = \begin{cases} -sign(\Delta W_{ij})V_{dd}, & 0 < t < |\nu_{ij}|T_{up} \\ -V_{dd}, & |\nu_{ij}|T_{up} < t < T_{up} \end{cases} \qquad (16)$$

$$e_{2,n} = \begin{cases} sign(\Delta W_{ij})V_{dd}, & 0 < t < |\nu_{ij}|T_{up} \\ -V_{dd}, & |\nu_{ij}|T_{up} < t < T_{up}. \end{cases} \qquad (17)$$

where ΔW, ν, and κ were defined in Section II. Hence, the MTJ is updated proportionally to $\kappa_{ij} = \lfloor \Delta W_{ij} \rfloor$ and

$\nu_{ij} = Remainder(\Delta W_{ij})$, meaning that for a single synapse, one MTJ is updated using a pulse width of $\Delta t = |\kappa_{ij}|T_{up}$ and the other with $\Delta t = |\nu_{ij}|T_{up}$. We assume that the κ and ν data are inputs to the synapse array. Using this work scheme, the synapse weight is updated as follows. κ_{ij} is an integer, so if $\kappa_{ij} \neq 0$, then the MTJ switching probability is approximately 1 and can be described as an indicator variable $sign(\kappa_{ij})\mathbb{1}_{\kappa \neq 0}$. ν_{ij} is a fraction, so the switching probability of the MTJ with respect to ν_{ij} is a Bernoulli variable with probability $P_{sw}(\nu_{ij})$. Thus, the MTJ-based synapse update is given by $\Delta w_{ij} = sign(\Delta W_{ij})(\mathbb{1}_{\kappa \neq 0} + Bern(P_{sw}(\nu_{ij})))$; see examples in Section IV-C.

Support of Stochastic Gradient Descent: This update scheme is similar to the update scheme proposed in [4]. When the SGD algorithm is used to train the network, all the synapses in the array are updated in parallel. To support SGD training, minor changes need to be made to the general update scheme. Using SGD, the update is given by the gradient value, and is equal to $\Delta W = u^T y$, where y is the error propagated back to the layer, using the backpropagation algorithm, and u is the input. For TNN and BNN the input activations are $u \in \{-1, 0, 1\} = \{-V_{up}, 0, V_{up}\}$ and $u \in \{-1, 1\} = \{-V_{up}, V_{up}\}$, respectively; thus, $\Delta W_{i,j} = y_i u_j = sign(u_j)y_i$ or $\Delta W_{i,j} = 0$ for $u = 0$. In this scheme, the voltage sources keep the activation values, so $u_1 = u_2 = u$ (whereas in the general scheme the voltage sources are set to $u_1 = u_2 = V_{up}$). The control signals are a function of the error y, whereas in ADAM and other optimization algorithms they are a function of the update value ΔW. The control signal functionality for SGD is

$$\begin{cases} e_{1,p} = -e_{2,p} = -sign(y_i)V_{dd}, & if \ \kappa_{ij} \neq 0 \\ e_{1,p} = e_{2,p} = V_{dd}, & else \end{cases} \quad (18)$$

$$e_{1,n} = \begin{cases} -sign(y_i)V_{dd}, & 0 < t < |\nu_{ij}|T_{up} \\ -V_{dd}, & |\nu_{ij}|T_{up} < t < T_{up} \end{cases} \quad (19)$$

$$e_{2,n} = \begin{cases} sign(y_i)V_{dd}, & 0 < t < |\nu_{ij}|T_{up} \\ -V_{dd}, & |\nu_{ij}|T_{up} < t < T_{up}. \end{cases} \quad (20)$$

The functionality of the control signals remains unchanged, the voltage source is selected according to y, and the voltage sign and the effective update duration are set as a function of κ and ν, the integer and remainder values of y, respectively. Therefore, the update equation is given by

$$\Delta w_{ij} = sign(y_i)sign(u_j)(\mathbb{1}_{\kappa \neq 0} + Bern(P_{sw}(\nu_{ij}))) \quad (21)$$

3) Inverse Read: To train the TNN, backpropagation of the error must be performed. Thus, an inverse matrix vector multiplication $W^T y$ is supported. Similarly to [4], we use the output row interface as an input. This allows reusing the same synapse array. Due to the synapse structure, the data is separated into two columns, as shown in Figure 2c. The output current, $I_{i,R_1} - I_{i,R_2}$, is converted to voltage using a voltage comparator.

C. Ternary Synapse Update Examples

To clarify the update scheme proposed in this paper, two examples of synapse updates are given.

1) Example 1*:* Figure 3a shows the case where a synapse weight is -1, and the update value is 1.5. Thus, $k = 1$ and $v = 0.5$. In that case, $\lfloor \Delta W \rfloor \neq 0$ and $sign(\Delta W) = 1$. Hence, $e_{1,p} = -e_{2,p} = -V_{dd}$; therefore, P_1 is ON and P_2 is OFF for time interval T_{up}. Hence, $P_{sw,1} \approx 1$. $e_{1,n} = -V_{dd}$ for T_{up} and $e_{2,n}$ is ON for $0.5T_{up}$, as given by

$$e_{2,n} = \begin{cases} V_{dd} & 0 < t < 0.5T_{up} \\ -V_{dd} & 0.5T_{up} < t < T_{up}. \end{cases} \quad (22)$$

Therefore, R_2 will switch with probability $P_{sw,2} = P(\frac{0.5T_{up}V_{up}}{R_{on}})$. In this example, the synapse weight will be updated from $-1 \rightarrow 0$ with probability

$$\begin{aligned} P_{-1\rightarrow 0} &= P_{-1\rightarrow 0_w} + P_{-1\rightarrow 0_s} = \\ &P_{sw,1}(1 - P_{sw,2}) + (1 - P_{sw,1})(1 - P_{sw,2}) \\ &\approx (1 - P_{sw,2}), \end{aligned} \quad (23)$$

and might switch to 1 with probability

$$P_{-1\rightarrow 1} = P_{sw,1}P_{sw,2} \approx P_{sw,2}. \quad (24)$$

Note that when $W = -1$, $\{R_1, R_2\} = \{R_{off}, R_{on}\}$. Thus, if $\Delta W < 0$, the current flow direction will be from R_2 to R_1 and the MTJ cannot switch.

2) Example 2*:* Figure 3b shows the case where a synapse weight is 0_w, and the update value is -0.5. Thus, $k = 0$ and $v = -0.5$. Hence, $\lfloor \Delta W \rfloor \neq 0$ and $sign(\Delta W) = -1$. Consequently, $e_{1,p} = e_{2,p} = V_{dd}$, so both P_1 and P_2 are closed for T_{up}. $e_{2,n} = -V_{dd}$ for T_{up} and $e_{1,n}$ is open for $0.5T_{up}$, as given by

$$e_{1,n} = \begin{cases} V_{dd} & 0 < t < 0.5T_{up} \\ -V_{dd} & 0.5T_{up} < t < T_{up}. \end{cases} \quad (25)$$

Therefore, R_1 will switch with probability $P_{sw,1} = P(\frac{0.5T_{up}V_{up}}{R_{on}})$. In this example, the synapse weight is updated from $0_w \rightarrow -1$ with probability $P = P_{sw,1}$. Although theoretically no current should flow through R_2, with probability $P_{sw,2} \approx 0$ it might switch from R_{on} to R_{off} due to leakage currents. It is important to note that the switching probability is a function of the resistance; therefore, the switching probability of $0_s = \{R_{off}, R_{off}\}$ is lower than $0_w = \{R_{on}, R_{on}\}$.

V. SUPPORT FOR BINARY NEURAL NETWORKS

In this section, we discuss how the proposed ternary synapse can support BNN. We address the changes that must be applied to the circuits to support binary weights.

A. MTJ-Based Binary Synapses

To support BNN instead of TNN, the GXNOR operation is replaced by a simple XNOR operation and the quantized space resolution is $\Delta z_0 = 2$.

1) Proposed Synapse Circuit and Synapse Array:

(a) Synapse update where $W = -1$ and $\Delta W = 1.5$

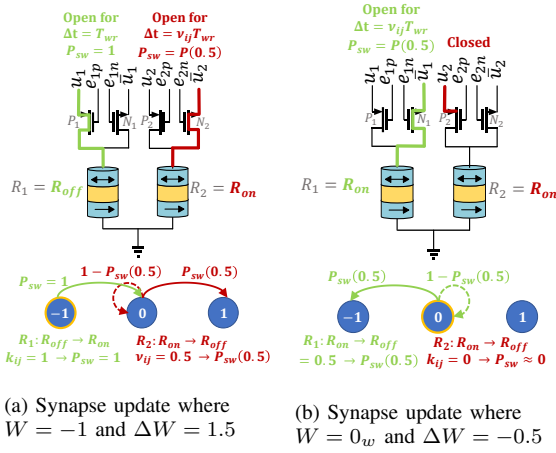(b) Synapse update where $W = 0_w$ and $\Delta W = -0.5$

Fig. 3. Examples of synapse update. The blue circle represents the logic state of the weights, where the initial state is marked by an orange outline.

Synapse Circuit: To support BWS, a 2T1R synapse is used as illustrated in Figure 4a. This synapse structure is similar to the synapse circuit suggested in our previous work, which used a resistive RAM device (RRAM) instead of an MTJ device [19, 20]. To reuse the ternary synapse proposed in this work to support binary weights, one "branch" of the synapse array is deactivated (see Figure 4b). To represent ± 1 values, a reference resistor $R_{ref} = (\frac{G_{on}+G_{off}}{2})^{-1}$ is added per synapse, and is connected in parallel to $\bar{u}$ of the corresponding synapse.

It seems that the ternary synapse could have been separated to two binary synapses with $e_{1,n} = e_{2,n}$ and $e_{1,p} = e_{2,p}$. Unfortunately, due to the use of the comparator, the ternary array cannot support the inverse read from all the columns; thus, it cannot support the backpropagation when the ternary synapse is split to two binary synapses (see Figure 4c). The 2T1R synapse can be used to design a dedicated engine for BNN; such a design does not need the comparators.

Synapse Weight: Table II defines the values of the weights when a 2T1R synapse is used. MTJ resistance of R_{on} leads to $W = 1$ and resistance of R_{off} leads to $W = -1$. To compute the XNOR operation between the weights and activation $,u$, the synapse current is compared to the reference value $I_{ref} = -uG_{ref} = -u\frac{G_{on}+G_{off}}{2}$. The result of the XNOR operation is given in the right column of Table II. While other methods to support binary weights can be considered (for example, using the resistance threshold value to separate the ± 1 weight values), this solution was chosen due to the low ratio between R_{off} and R_{on}, which is a common property of MTJ devices.

Synapse Array: If the proposed synapse array is used, each weight can use only one branch of the ternary synapse; thus the synapse can represent only a single bit, and half of the array is deactivated using binary mode. Similarly to the method in [19], the reference resistors added to each row are located together (see Figure 4b) and are active only during the feedforward phase of the BNN ($e_{br} =' 1'$).

B. Training BNN

1) XNOR and Feedforward: As in the GXNOR operation, we denote the input neuron values as the voltage sources. The logic values $\{-1, 1\}$ are represented by $u \in \{-V_{rd}, V_{rd}\}$. The result of each XNOR operation is

$$I_{out} = Gu, \tag{26}$$

where G is the conductance of the MTJ. During feedforward, the control signal $e_{br} =' 1'$, and hence the reference resistors are connected and the current through each row is

$$I_{row,i} = \sum_{j=1}^{N} G_{ij}u_j + \sum_{j=1}^{N} G_{ij}\bar{u}_j = \\ \frac{R_{off} - R_{on}}{2R_{off}R_{on}}(N_{+1,i} - N_{-1,i})V_{rd}, \tag{27}$$

where G_{ij} is the MTJ conductivity of synapse j in row i, M is the number of synapses per row, $M_{+1,i}$ is the total number of positive products in row i, and $M_{-1,i}$ is the total number of negative products in row i.

2) Weight Update: In a manner similar to the TNN update scheme suggested in the paper, the MTJ device of each binary synapse is updated to support the GXNOR algorithm [6]. Figure 1 illustrates two update examples for the binary weights using the GXNOR algorithm.

The control signal must have the following functionality: if $\kappa_{ij} = \lfloor \Delta W_{ij}/2 \rfloor \neq 0$, a switch will occur with probability $P_{sw} \approx 1$; otherwise the switching probability is a function of $\nu_{ij} = reminder(\Delta W_{ij}/2)$.

The control signals are set as follows. First, the reference resistors are disconnected, and thus

$$e_{br} =' 0'. \tag{28}$$

The row control signals are

$$\begin{cases} e_{2,n} = -V_{dd}, \\ e_{2,p} = V_{dd}, \end{cases} \tag{29}$$

so branch 2 of each synapse is deactivated. Signals $e_{1,p}$ and $e_{1,n}$, which control the weight update, are given by

$$e_{1,p} = \begin{cases} -sign(\Delta W_{ij})V_{dd}, & 0 < t < \psi T_{up} \\ V_{dd}, & \psi T_{up} < t < T_{up} \end{cases} \tag{30}$$

$$e_{1,n} = \begin{cases} -sign(\Delta W_{ij})V_{dd}, & 0 < t < \psi T_{up} \\ -V_{dd}, & \psi T_{up} < t < T_{up} \end{cases} \tag{31}$$

where $\psi = \max(|\kappa_{ij}|, |\nu_{ij}|)$.

3) Inverse Read: To compute the value of each multiplication, the current read from the activated synapse must be compared to the reference value $I_{ref} = -yG_{ref} = -y\frac{G_{on}+G_{off}}{2}$. As in the feedforward solution, a reference resistor is added per synapse in the column, and voltage y is applied across it. The resistors are located together as illustrated in Figure 4c and are connected to the row only if $e_{B_{BP}} =' 1'$. Thus, the current comparator will compute

$$\sum_{i=1}^{M}(I_{i,R_1} - I_{i,ref}) = \sum_{i=1}^{M}\left(G_{ij} - \frac{G_{on} + G_{off}}{2}\right)y_i, \tag{32}$$

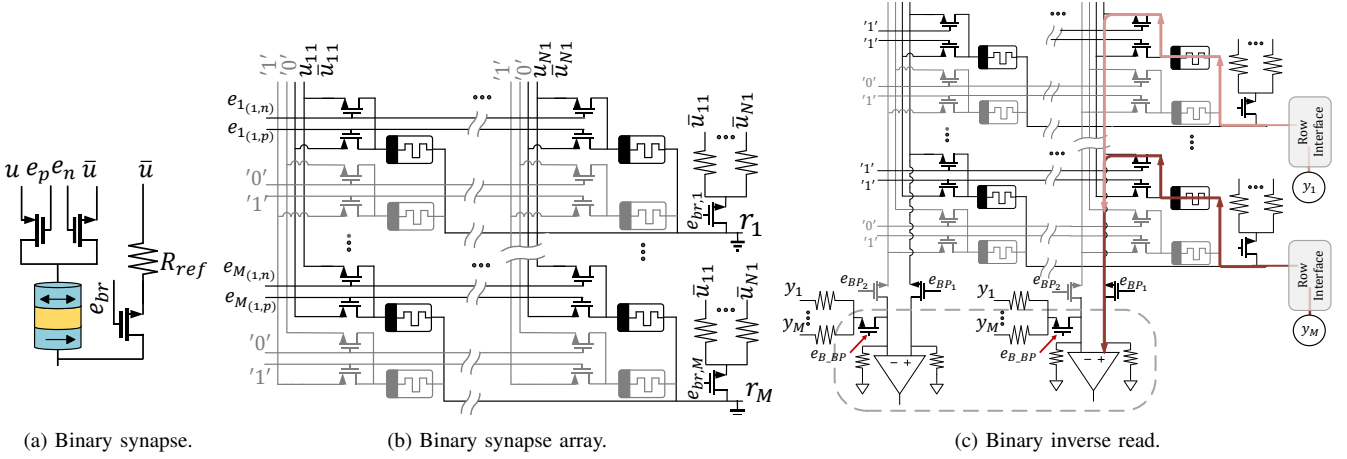where N is the number of synapses per column.

Fig. 4. (a) Binary synapse, a reference resistor is added for each synapse. This concept is similar to our previous work for RRAM synapses [19]. (b) Binary synapse array. To support binary weights with the ternary synapse, only one "branch" of the synapse is used; the deactivated branch is marked in grey. (c) The data flow of the inverse read operation.

TABLE II
BINARY SYNAPSE STATES AND OUTPUT CURRENT

Weight	R	I_{out}	$I_{out} - uG_{ref}$
1	R_{on}	$G_{on}u$	$u\frac{R_{off}-R_{on}}{2R_{off}R_{on}}$
-1	R_{off}	$G_{off}u$	$-u\frac{R_{off}-R_{on}}{2R_{off}R_{on}}$

VI. EVALUATION AND DESIGN CONSIDERATIONS

This section presents the evaluation of the MTJ-based training performance. The synapse circuit and array are evaluated and the circuit parameters and behavior were extracted and used for the training simulations. In this section, the software and the MTJ-based implementations of the GXNOR algorithm are referred to as GXNOR and MTJ-GXNOR, respectively.

A. Circuit Evaluation

The synapse circuit was designed and evaluated in Cadence Virtuoso for the GlobalFoundries 28nm FD-SOI process. The MTJ device is based on device C from [16] and its parameters are listed in Table III. To achieve higher switching probability, the magnetization saturation ($\mu_0 M_s$) was changed according to [21]. The read voltage, V_{rd}, was set to guarantee a low-current regime and negligible switching probability for the feedforward and inverse read operations. Likewise, the update voltage, V_{up}, was set to guarantee a high-current regime. The update time period was set to match $P_{sw}(T_{up}) \approx 1$.

1) MTJ Switching Simulation: To evaluate the MTJ transition resistance and the impact of the MTJ transient response on the synapse circuit operation, we ran a Monte-Carlo simulation of the MTJ operation. The simulation numerically solves the Landau–Lifshitz–Gilbert (LLG) [11, 22] differential equation (assuming the MTJ is a single magnetic domain) with the addition of a stochastic term for the thermal fluctuations [23] and Slonczewski's STT term [24]. For each iteration of the Monte-Carlo simulation, a different random sequence was introduced to the LLG equation and the resulting MTJ resistance
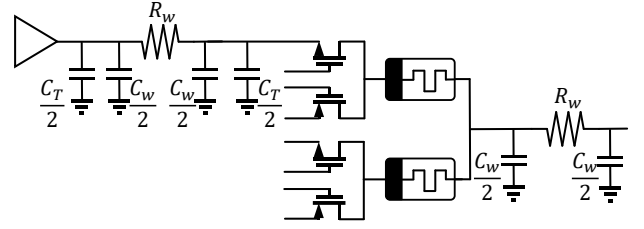

Fig. 5. The circuit schematic model considers the parasitic capacitance and resistance of the wires and transistors, which are dependent on the cell location within the synapse array.

trace was retrieved. The equation was solved using a standard midpoint scheme [25] and was interpreted in the sense of Stratonovich, assuming no external magnetic field [26] and a voltage pulse waveform. The resistance of the MTJ was taken as $R_{on}\frac{1+P^2}{1+P^2\cos\theta}$ [27], where θ is the angle between magnetization moments of the free and fixed layers and P is the spin polarization of the current. To approximate the time-variation resistance of an MTJ during the switch between states, all the traces from the Monte-Carlo simulation were aligned using the first time that the resistance of the MTJ reached $\frac{R_{on}+R_{off}}{2}$. After the alignment, a mean trace was extracted and used for the fit. This fit was used as the time-variation resistance when the MTJ made a state switch.

2) Circuit Schematic Model: The transistor and the interconnect affect the circuit performance and operations. Therefore, we adopt the circuit model illustrated in Figure 5, which considers the parasitic resistance and capacitance. We considered the corner cases (*i.e.*, the synapses located at the four corners of the synapse array) to evaluate the effect of the wires and transistors on operation results, latency and power consumption. For the following circuit simulations, we considered the worst case where the wire resistance and capacitance are the most significant (*i.e.*, for an array of size $M \times N$, the synapse located at [M,1]).

TABLE III
CIRCUIT PARAMETERS [16]

MTJ, device C [16]			
Parameter	**Value**	**Parameter**	**Value**
a[nm]	50	Temp. [K]	300
b[nm]	20	$R_{on}[\Omega]$	1500
t_f[nm]	2.0	$R_{off}[\Omega]$	2500
$\mu_0 M_s$[T][1]	0.5	α	0.01
CMOS			
Parameter	**Value**	**Parameter**	**Value**
V_{DD}[V]	1	W/L$_{PMOS}$	33
V_{SS}[V]	-1	W/L$_{NMOS}$	20
V_{up}[V]	1	T_{up}[ns]	2
V_{rd}[V]	0.1	T_{rd}[ns]	0.5

[1] To achieve higher switching probability, the value
of $\mu_0 M_s$ was changed according to [21].

TABLE IV
ACCURACY OF STATE-OF-THE-ART ALGORITHMS

Methods	Datasets	
	MNIST	**SVHN**
BNNs [7]	98.6%	97.20%
BWNs [8]	98.82%	97.70%
GXNOR TNN [6]	99.32%	94.12%
GXNOR BNN [6]	98.54%	91.68%
MTJ-GXNOR TNN	**98.61%**	**93.99%**
MTJ-GXNOR Bin-Activation	**98.6%**	**93.62%**
MTJ-GXNOR BNN Full	**97.84%**	**89.46%**

B. MTJ-GXNOR Training Simulation

To evaluate the training performance of the MTJ-based synapse, we simulated the training of two TNN and BNN architectures using the MTJ-based synapse over the MNIST and SVHN datasets [14, 15] in PyTorch. The network architecture for MNIST is "32C5-MP2-64C5-MP2-512FC-SVM," and for SVHN it is "2×(128C3)-MP2-2×(256C3)-MP2-2×(512C3)-MP2-1024FC-SVM" [6]. All the training parameters were set to match the settings in [6]. The synapse circuit parameters were extracted from the SPICE simulations. Table IV lists the test accuracy of MTJ-GXNOR as compared to GXNOR and other state-of-the-art algorithms. BNNs [7] and BWNs [8] constrain the weights and activation to the ternary and binary spaces. However, in contrast to GXNOR, these networks keep the full-precision weights during the training phase, which increases the frequency of memory access and requires supporting full-precision arithmetic. The results of the TNN training using the MTJ-based synapse (MTJ-GXNOR TNN) are similar to the results of the GXNOR training. When the ternary synapse is used, the activation can be constrained to binary values using a sign function [6], although the weights cannot be constrained to the binary space. Therefore, we also explore a mixed precision network that uses binary activations with ternary weights (MTJ-GXNOR Bin-Activation). When trained on the SVHN dataset, the test accuracy of MTJ-GXNOR BNN is lower than that of GXNOR BNN, while the test accuracy of MTJ-GXNOR Bin-Activation is closer to that of GXNOR TNN.

TABLE V
TEST ACCURACY VS. PROCESS
VARIATION FOR MNIST

RSD	**Resistance Variation**	θ_0 **Variation**
0%	98.61%	98.61%
1%	98.13%	97.98%
5%	98.13%	97.92%
10%	98.1%	97.98%
30%	98.15%	98.05%
35%	97.94%	98.05%

TABLE VI
TEST ACCURACY VS. θ_0

θ_0[rad]	**Test Accu.**
0.0913	94.28%
0.1141	94.98%
0.2739	97.39%
0.345	**98.61%**

C. Sensitivity to Process Variation

Variation in the device parameters and environment may affect the performance of the proposed circuits. In this section, we evaluate the sensitivity of the TNN training performance to process variation.

1) Resistance Variation and θ Distribution Variation: Two cases of process variation were considered: 1) resistance variation; and 2) variation in θ distribution. Variation in the device resistance and θ distribution may lead to different switching probability per MTJ device. To evaluate the sensitivity of the training to the device-to-device variation, we simulated the MNIST-architecture training with variations in the resistance and θ distributions. Several Gaussian variabilities were examined with different relative standard deviations (RSD). Table V lists the training accuracy for resistance variation and θ variation. According to [11], the resistance RSD was found to be approximately 5%, while our simulations show that the training accuracy is robust to the resistance variation even for higher RSD values (*e.g.* only 0.46% accuracy degradation for RSD= 30%). The training accuracy is more sensitive to variations in θ. Nevertheless, high standard deviation of θ values results in better training accuracy. We concluded that the performance of the MTJ-GXNOR algorithm improves for higher variations in θ. Table VI lists the training results for different θ_0 values; the θ_0 value used in this work is marked in bold. Larger θ_0 values, which correspond to higher randomness of the MTJ switching process, yield better accuracy.

2) Sensitivity to Voltage Non-Ideality: The operation most sensitive to voltage variation is the weight update operation, where the update probability is a function of the voltage drop across the MTJ device. Therefore, in this section we evaluate the test accuracy obtained for variation in the voltage source. Increasing the voltage leads to higher switching probability and θ_0 variance. Hence, increasing the voltage magnitude increases the randomness of the MTJ switching. Therefore, the voltage magnitude can be used to improve the stochastic switching process and to improve the network training performance when using an MTJ device with low θ_0 variance. In the case simulated in this work, increasing the voltage magnitude above $V_{up} = 1.1V$ only slightly improves test accuracy; hence, in this work we set $V_{up} = 1V$ to constrain the power consumption of our design.

3) Sensitivity to Temperature: The ambient temperature affects the switching behavior of the MTJ [28]. When the

TABLE VII
TEMPERATURE EFFECT ON TEST ACCURACY FOR MNIST

T[K]	260	273	300	333	373
$R_{off}[\Omega]$	2750	2650	2500	2150	2000
$\theta_0[rad]$	0.3187	0.3266	0.345	0.3617	0.3827
Test Accuracy(%)	**98.14**	**98.32**	**98.66**	**98.82**	**98.88**

TABLE VIII
AREA AND POWER

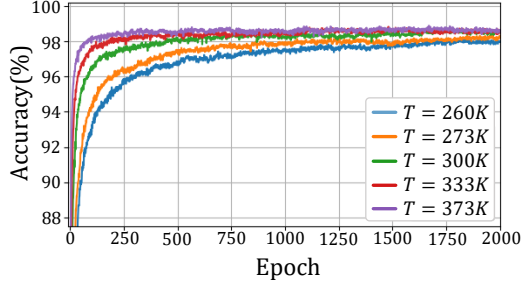| Cell	Area	Power	
		XNOR+sum	**Update**
Single Synapse	$3.63\mu m^2$	$1.89\mu W$	$2.72\mu W$
64×64 Syn. array	$0.015mm^2$	$7.31mW$	$1.64mW$
128×128 Syn. array	$0.059mm^2$	$28.5mW$	$3.25mW$

Fig. 6. Test accuracy during the training phase for temperature range $[273K, 373K]$. Increasing the temperature leads to larger θ_0 variance; thus, it increases the randomness of the MTJ switching time. Therefore, higher temperature leads to faster convergence.

temperature increases, the R_{off} resistance decreases. The R_{on} resistance value has a much weaker temperature dependency and it is nearly constant [29]. The transistors can be described as variable current sources, where for high temperatures the drivability of the MOS transistor is degraded because the electron mobility decreases. Hence, the ambient temperature has opposite effects on the R_{off} of the MTJ and the drivability of the MOS transistor, which affect the switching probability. Additionally, the initial magnetization angle, θ, depends on the temperature by the normal distribution $\theta \sim \mathcal{N}(0, \theta_0)$, where the standard deviation is $\theta_0 = \sqrt{k_B T/(\mu_0 H_k M_s V)}$. Hence, θ_0 increases for higher temperature.

As mentioned in Section VI-C, the training performance is highly dependent on the variance of θ. Thus, to evaluate the sensitivity of the MTJ-GXNOR training to the ambient temperature, we focused on θ_0, and left the resistance and drivability evaluation for future work. To estimate the sensitivity of the MTJ-based synapse to the temperature, we simulated MTJ-based training with different temperatures in the range $[260K, 373K]$, where the resistances are extrapolated to emulate the temperature dependence. Table VII lists the test accuracy obtained for different temperatures. Although better accuracy is obtained for higher temperatures, the training phase and network accuracy are robust to temperature variations. Figure 6 shows the test accuracy over the training phase for the MNIST network. Higher temperatures, which correspond to higher θ_0, increase the convergence rate of the network while the network converges to similar test accuracy for all the temperatures in the examined range. Further research on how the ambient temperature affects the synapse circuit and array is left for future work.

D. Performance Evaluation

In this section, the power and area of TNNs execution is presented.

1) TNN Power and Area: The power consumption and area were evaluated for a single synapse and synapse array, including the interconnect parasitics. The results are listed in Table VIII. During the read operation, all the synapses are read in parallel; therefore, the feedforward power is higher than the write power, where the columns are updated serially.

2) Potential Performance: QNNs were proposed as a way to reduce the overall power consumption and complexity of the full precision DNNs; hence, we evaluated the energy efficiency (in units of $\frac{TOPs}{W}$) of our design. For the feedforward phase in a 128×128 synapse array, $128 \times (128 + 128)$ GXNOR and accumulate operations are done in parallel (1OP= $1b$ GXNOR/Accumulate/update). Therefore, the synapse array can reach $2299\frac{TOPs}{W}$ in this phase. When performing update, we count each update as a single operation; the energy efficiency when updating the weights is thus $39\frac{TOPs}{W}$. During the update phase the voltage source is set to guarantee a high current domain; the energy efficiency of the update operation is therefore bounded by the MTJ device properties.

3) System Performance (Test Case): To evaluate the performance of the synapse array when integrating our design to a full system, we consider the following (but not the only possible) setup, when the performance will change for different setups. A 128×128 synapse array is used. The synapse array is used as an analog computation engine and as memory for the weights; hence, the input and output to the array are converted to using 1-bit DAC and 8-bit ADC (for the array dimensions the output of each row is in $[-128, 128]$ and a 9 bit ADC is needed. To reduce the overhead, we assumes for this discussion that $[-127, 127]$ is sufficient, hence, a 8 bit ADC is sufficient). In the inverse read phase, we consider the bit-streaming method as suggested in [30] to compute the multiplication with the full-precision error data; thus, only a 1-bit DAC a 8-bit ADC are needed. To generate the control signals, an 8-bit DAC, and voltage comperators are needed. The power and area of those components are listed in Table IX. The respective energy efficiency in the feedforward and update phases is $18.3\frac{TOPs}{W}$ and $3\frac{TOPs}{W}$, where the power consumption of the data converters limits the overall performance. For the bit-streaming method [30] with 8-bit precision for the error data, the energy efficiency of the inverse read operation is $1.43\frac{TOPs}{W}$.

E. Comparison to Previous Work

Previous works that propose in-situ hardware implementations of BNN and TNN support only inference. In [32], a CMOS-based computation-near-memory (CNM) engine was designed and fabricated. The design's energy efficiency during

TABLE IX
TEST CASE MODEL

Component	Number	Power [mW]
ADC 8-bit [30]	8	16
DAC 8-bit [31]	2×128	5.52
DAC 1-bit [30]	2×128	1

inference is $532\frac{TOPs}{W}$. They assumed that the binary activation can be done immediately after the convolution, thus eliminating the ADC. Similar assumption for our setup will increase the inference energy efficiency of our design to $180\frac{TOPs}{W}$. BNN inference without the need for an ADC is also supported in [33], where energy efficiency of $1326\frac{TOPs}{W}$ was reported. In that work, a RRAM device is used instead of an MTJ. The RRAM-based synapse can use smaller access transistors than the MTJ-based device, which is current driven. Moreover, a 1T1R synapse is sufficient when supporting only inference, thus reducing the complexity and overall power consumption of each synapse.

In [12], a 1T1R and 1R structure were proposed, and the stochastic behavior of the MTJ was also leveraged to support in situ BNN training. Two update operations are required for the 1T1R and four for the 1R, whereas our synapse can perform positive and negative operations in parallel, thus requiring only one update operation. Moreover, full precision activations are used in [12]. As a result, a high resolution DAC is still necessary to convert the input value to voltages. Using the DAC circuits listed at Table IX, the 8-bit DAC consume $5\times$ more energy than the 1-bit DAC. The power consumption and complexity of the synapse array are greater as a result.

VII. CONCLUSIONS

In this paper, we proposed a novel MTJ-based synapse circuit and showed that a QNN, and especially a TNN and BNN, can be trained using the MTJ-based synapse, without sacrificing accuracy. The proposed circuit enables in-situ, highly parallel and energy efficient execution of weight-related computation. Such a circuit can accelerate TNN inference and training execution on low-power devices, such as IoT and consumer devices. To fulfill the potential of the MTJ-based synapse, the next step is to integrate it into a full system design. Further research on the effects of environmental factors, such as temperature and magnetic fields, on circuit performance will be explored in future work.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[2] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A Machine-Learning Supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622, Dec 2014.

[3] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 541–552, 2017.

[4] D. Soudry, D. D. Castro, A. Gal, A. Kolodny, and S. Kvatinsky, "Memristor-Based Multilayer Neural Networks With Online Gradient Descent Training," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, pp. 2408–2421, Oct 2015.

[5] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *arXiv:1609.07061*, 2016.

[6] L. Deng, P. Jiao, J. Pei, Z. Wu, and G. Li, "GXNOR-Net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework," *Neural Networks*, vol. 100, pp. 49–58, 2018.

[7] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," *arXiv:1602.02830*, 2016.

[8] M. Courbariau, Y. Bengio, and J.-P. Davi, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," in *Advances in Neural Information Processing Systems 28*, pp. 3123–3131, 2015.

[9] X. Sun, S. Yin, X. Peng, R. Liu, J. s. Seo, and S. Yu, "XNOR-RRAM: A Scalable and Parallel Resistive Synaptic Architecture for Binary Neural Networks," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1423–1428, March 2018.

[10] S. Yu, B. Gao, Z. Fang, H. Yu, J. Kang, and H.-S. P. Wong, "Stochastic learning in oxide binary synaptic device for neuromorphic computing," *Frontiers in Neuroscience*, vol. 7, p. 186, 2013.

[11] A. F. Vincent, J. Larroque, N. Locatelli, N. B. Romdhane, O. Bichler, C. Gamrat, W. S. Zhao, J. O. Klein, S. Galdin-Retailleau, and D. Querlioz, "Spin-transfer torque magnetic memory as a stochastic memristive synapse for neuromorphic systems," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 9, no. 2, pp. 166–174, 2015.

[12] A. Mondal and A. Srivastava, "In-situ Stochastic Training of MTJ Crossbar Based Neural Networks," in *Proceedings of the International Symposium on Low Power Electronics and Design*, ISLPED '18, (New York, NY, USA), pp. 51:1–51:6, ACM, 2018.

[13] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

[15] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[16] A. Vincent, N. Locatelli, J.-O. Klein, W. Zhao, S. Galdin-Retailleau, and D. Querlioz, "Analytical macrospin modeling of the stochastic switching time of spin transfer-torque devices," *IEEE Trans. Electron Devices*, vol. 62, no. 1, pp. 164–170, 2015.

[17] D. Soudry, D. Di Castro, A. Gal, A. Kolodny, and S. Kvatinsky, "Memristor-based multilayer neural networks with online gradient descent training," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, pp. 2408–2421, Oct 2015.

[18] T. Greenberg-Toledo, R. Mazor, A. Haj-Ali, and S. Kvatinsky, "Supporting the momentum training algorithm using a memristor-based synapse," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, pp. 1571–1583, April 2019.

[19] E. Rosenthal, S. Greshnikov, D. Soudry, and S. Kvatinsky, "A fully analog memristor-based neural network with online gradient training," *Proceedings - IEEE International Symposium on Circuits and Systems*, vol. 2016-July, pp. 1394–1397, 2016.

[20] D. Soudry, D. D. Castro, A. Gal, A. Kolodny, and S. Kvatinsky, "Memristor-based multilayer neural networks with online gradient descent training," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 10, pp. 2408–2421.

[21] H. Kubota, A. Fukushima, K. Yakushiji, S. Yakata, S. Yuasa, K. Ando, M. Ogane, Y. Ando, and T. Miyazaki, "Reduction in switching current using a low-saturation magnetization Co–Fe–(Cr, V)–B free layer in MgO-based magnetic tunnel junctions," *Journal of Applied Physics*, vol. 105, pp. 7–117, 2009.

[22] T. L. Gilbert, "A phenomenological theory of damping in ferromagnetic materials," *IEEE Transactions on Magnetics*, vol. 40, pp. 3443–3449, Nov 2004.

[23] J. L. García-Palacios and F. J. Lázaro, "Langevin-dynamics study of the dynamical properties of small magnetic particles," *Phys. Rev. B*, vol. 58, pp. 14937–14958, Dec 1998.

[24] J. Slonczewski, "Current-driven excitation of magnetic multilayers," *Journal of Magnetism and Magnetic Materials*, vol. 159, no. 1, pp. L1 – L7, 1996.

[25] M. d'Aquino, C. Serpico, G. Coppola, I. D. Mayergoyz, and G. Bertotti, "Midpoint numerical technique for stochastic Landau-Lifshitz-Gilbert

dynamics," *Journal of Applied Physics*, vol. 99, no. 8, p. 08B905, 2006.

[26] Z. Diao, Z. Li, S. Wang, Y. Ding, A. Panchula, E. Chen, L.-C. Wang, and Y. Huai, "Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory," *Journal of Physics: Condensed Matter*, vol. 19, no. 16, p. 165209, 2007.

[27] J. C. Slonczewski, "Conductance and exchange coupling of two ferromagnets separated by a tunneling barrier," *Phys. Rev. B*, vol. 39, pp. 6995–7002, Apr 1989.

[28] X. Bi, H. Li, and X. Wang, "STT-RAM Cell Design Considering CMOS and MTJ Temperature Dependence," *IEEE Transactions on Magnetics*, vol. 48, pp. 3821–3824, Nov 2012.

[29] X. Liu, D. Mazumdar, W. Shen, B. D. Schrag, and G. Xiao, "Thermal stability of magnetic tunneling junctions with MgO barriers for high temperature spintronics," *Applied Physics Letters*, vol. 89, no. 2, p. 023504, 2006.

[30] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 14–26, 2016.

[31] M. Saberi, R. Lotfi, K. Mafinezhad, and W. A. Serdijn, "Analysis of power consumption and linearity in capacitive digital-to-analog converters used in successive approximation adcs," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, pp. 1736–1748, Aug 2011.

[32] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "an always-on 3.8μj/86% cifar-10 mixed-signal binary cnn processor with all memory on chip in 28nm cmos," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*.

[33] E. Giacomin, T. Grinberg, S. Kvatinsky, and P.-e. Gaillardon, "A Robust Digital RRAM-based Convolutional Block for Low Energy Image Processing and Vision Applications," in *Design Automation Conference*, 2018.