NeuroScrub: Mitigating Retention Failures Using Approximate Scrubbing in Neuromorphic Fabric Based on Resistive Memories

Soyed Tuhin Ahmed, Michael Hefenbrock, Christopher Münch, Mehdi B. Tahoori
Karlsruhe Institute of Technology (KIT)
soyed.ahmed@kit.edu, michael.hefenbrock@kit.edu, christopher.muench@kit.edu, mehdi.tahoori@kit.edu

***Abstract*—Neuromorphic computation-in-memory fabric based on emerging non-volatile memories (NVM) is considered an attractive option to accelerate neural networks (NNs) in hardware as they provide high-performance, low-power, and reduced data movement. Although NVMs offer many benefits, they are susceptible to data retention faults, where previously stored data is not retained. This severely impacts the inference accuracy of mapped NNs. Traditionally, memory scrubbing with error-correcting codes (ECC) is employed to mitigate retention faults in conventional CMOS memories. This is not feasible in NVM-based neuromorphic fabric due to high overhead and inability to represent encoding or decoding in analog computing. In this work, we propose an approximate scrubbing technique for NVM-based neuromorphic fabric to mitigate uni-directional retention faults with minimal storage overhead. The training of the NNs adjusted accordingly to meet the requirements of the scrubbing scheme. On different benchmarks, the proposed scrubbing approach can improve the inference accuracy up to 85.51% over the lifetime with virtually zero storage overhead.**

I. Introduction

Complex computational tasks like image recognition and sensor data processing for *Internet-of-Things* (IoT) devices are the focus of modern computer architectures. *Neural networks* (NNs) offer a compelling solution to these problems due to their high performance in cognitive tasks. The inference of NNs is usually carried out in computation-centric architectures such as graphical processing units (GPUs) or special-purpose hardware such as *Tensor Processing Units* (TPUs). Although computation-centric architectures offer flexibility, they are highly inefficient due to data movement between the processing unit and the memory, resulting in the well-known *memory wall* [1].

Emerging *Non-Volatile Resistive Memories* (NVMs) offer a promising avenue to accelerate deep learning applications for neuromorphic computing-in-memory (CiM). There are several technologies like *Resistive RAM* (ReRAM) [2], *Phase-Change Memory* (PCM) [3], and *Spin Transfer Torque Magnetic RAM* (STT-MRAM) [4] that can be used to perform training and inference of NNs directly in memory. Compared to conventional static and dynamic RAM architectures, they allow for highly scalable designs, which are non-volatile and can be operated with low power consumption [5].

However, emerging NVM technologies also introduce additional sources of failures, which need to be considered. Particularly, the retention faults of NVM devices have a significant influence on the performance of NVM-based NN accelerators [6]. The retention faults are typically more likely to happen uni-directional, which means that a memory cell in one state is more likely to switch to the other state [7], [8]. Thus, the accuracy of a NN accelerator with NVM storage drops drastically over its lifetime if retention faults are not mitigated [9].

One way to mitigate this for STT-MRAMs is to increase thermal stability [10]. However, this leads to increased write energy and influences the resistive behavior of the *Low Resistance State* (LRS) and *High Resistance State* (HRS) of the memory cell. In PCMs, the unused resistance ranges between the resistive states (guard-band) can be increased to reduce the retention fault rate, but the write latency increases as a trade-off [11].

More generally, *Error-Correcting Code* (ECC)-based scrubbing is used to recover corrupted data in memory arrays [12]. ECC-based scrubbing adds redundant data bits to the memory and can detect and correct failures within a certain limit depending on the accepted hardware overhead. This overhead is not practical for neuromorphic computing, which needs large NVM arrays. Hence, a significant number of redundant cells will be required. Moreover, the encoding, decoding, and correction required for the ECC-based scrubbing cannot be implemented in analog neuromorphic computing.

We propose a scrubbing technique to counteract technology-dependent uni-directional retention faults (HRS to LRS or vice versa, depending on the particular technology) in NVM cells used for neuromorphic computing. The technique is based on the fact that neural networks can tolerate a certain amount of retention faults without suffering too much in accuracy (as demonstrated in [6]). The proposed scrubbing technique only corrects uni-directional faults in the scrub area and thereby only approximately restores the intended weight matrices in the NVM memory. The main contributions of this work are:

- We introduce an approximate scrubbing technique to mitigate retention failures. For this, the NVM-based crossbar used for the hidden layers of NNs is divided into two areas, where unstable and stable NN weights are mapped, respectively.
- We introduce a novel NN training technique that adjusts the weight matrix to our scrubbing requirement during training.

- We present extensive experimental results on various datasets to confirm that our proposed scrubbing technique can improve the final inference accuracy up to 85.51% over the device lifetime without notable memory overhead.

The rest of the paper is structured as follows. Section II covers the background and related work to our approach. In Section III, the main ideas and motivation of our proposed scrubbing and training approach are presented. Finally, Section IV presents our simulation framework and results, while Section V concludes the paper.

II. Background

A. Retention Failures in NVMs

Retention failures are a major reliability issue in NVMs. Ideally, after an NVM cell is written, its content is expected to be stored until the next write operation. However, due to external influences, the cell may lose its datum or the written resistance drifts in multi-level NVMs. Additionally, most NVMs have an asymmetrical flip behavior, making it more probable for an NVM cell to change to a specific state over time [7], [13].

In the case of the *Magnetic Tunnel Junction* (MTJ), which is the storage element of STT-MRAM, the energy barrier from the HRS to the LRS is lower than the barrier from LRS to HRS, making it more probable for an MTJ to switch from the HRS to the LRS than the other way around [8]. Additionally, increasing the temperature lowers the effective energy barrier, which in turn increases the possibility of retention faults [14].

Changing the dimension of an MTJ is a typical design choice to vary the thermal stability. Decreasing the thermal stability also enables smaller access transistors resulting in a denser memory array. However, as the energy barrier is directly related to the cell's thermal stability, this results in a higher probability of retention faults [15]. Similarly, when the dimension of ReRAM devices is scaled-down, their reliability degrades and they perform unpredictably [16].

B. Neural Networks (NNs)

Neural networks (NNs) are biologically inspired computation graphs, constructed out of so-called neurons that form their fundamental building blocks. A neuron expresses a composition of a linear function of its inputs and a nonlinear function referred to as an activation function. Most commonly, the computational results of several neurons are combined in so-called layers. The most basic ones being densely connected layers can be expressed as

$$\mathrm{Layer}(\boldsymbol{x}) = \phi\left(\boldsymbol{W}\boldsymbol{x} + b\right)$$

where the vector $\boldsymbol{x} = [x_1, \cdots, x_i, \cdots]^T$ is the vector of inputs x_i, $\phi(\cdot)$ referring to the activation function (applied element-wise) and $\boldsymbol{W}$ and b denoting the weight matrix and the bias vector, respectively. Aside from simple layers of neurons, various other layers, such as normalization layers, have been proposed [17]. Through a set of layers, an NN $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is constructed as a composition of them, i.e.,

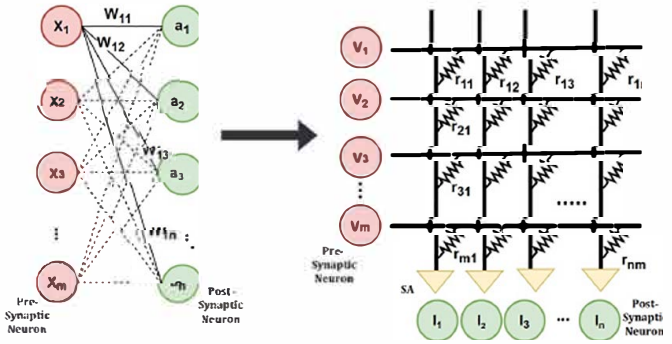

Fig. 1. (a) A single layer of a fully connected NN, (b) crossbar array with resistive NVM cells, receives inputs as an analog voltage. The trained weights w of the NN are mapped to the crossbars array as resistive states r of the NVM cells. A sense amplifier (SA) determines each neuron fires or not.

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \left(\mathrm{Layer}^L \circ \cdots \circ \mathrm{Layer}^l \circ \cdots \circ \mathrm{Layer}^1\right)(\boldsymbol{x}).$$

For brevity, the parameter vector $\boldsymbol{\theta}$ summarizes all parameters, e.g., the weight matrices $\boldsymbol{W}^l$ and biases b^l of all layers $l = 1, \cdots L$. To train an NN on some data $\mathcal{D} = \{(\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)})\}_{n=1}^{N}$, consisting of inputs $\boldsymbol{x}$ and expected output $\boldsymbol{y}$ pairs from a given task, its parameters $\boldsymbol{\theta}$ are minimized with respect to some loss function $\mathrm{Loss}(\boldsymbol{\theta}, \mathcal{D})$, i.e.,

$$\min_{\boldsymbol{\theta}} \quad \mathrm{Loss}(\boldsymbol{\theta}, \mathcal{D})$$

The loss function is task-specific, and expresses the quality of the match between $\boldsymbol{y}$ and $f_{\boldsymbol{\theta}}(\boldsymbol{x})$.

Usually, NNs have fixed-point weights and activation that requires a higher memory size to store the trained weights and are computationally expensive. *Binary Neural Networks* (BNNs) use binary ($+1$ or -1) weights and activation functions during their inference and require only 1-bit to store a single trained weight [18]. By reducing the bit-width of the weights from a multi-bit fixed point to a single-bit per weight, we can directly map an NN layer to an NVM crossbar. This allows us to use their in-memory computation capabilities to perform the neuromorphic operations directly within the memory. Fig. 1 shows how a fully connected neural network is mapped to an NVM-based crossbar structure.

A one-time write operation is used to map the trained weights of the BNN to the crossbar arrays, where each connection of two layers is represented by one crossbar. Resistance values r of the NVM cells represent the NN weights $\boldsymbol{w}$. More specifically, in this paper synaptic $+1$ (unstable) and -1 (stable) weights are represented with HRS and LRS cells, respectively. Note that this can be decided based on the NVM technology and direction of state change. The NN inputs for the inference are applied as an analog voltage to every row of the crossbar array. Multiple worldlines rows are activated concurrently and a weighted sum operation is performed in parallel. Each of the input voltage is multiplied by each NVM's conductance, and the resulting current is accumulated column-wise on the bitline. Each neuron's activation is computed using

a sense amplifier (SA), which determines if the neuron fires or not depending on the bitline current [1]. To calculate the result of a neuron, first all rows with an input value of $+1$ are enabled and the number of HRS cells are evaluated. In the second step, all rows with an input of -1 are enabled and the number of LRS cells are evaluated. Adding the HRS result to the LRS result yields the result of the operation.

C. Related Work

Checksum-based error correction is proposed for RRAM-based crossbars in [19]. However, memory overhead and power consumption do not scale well with the size of the crossbar. In [20], an ECC-based scrubbing technique for STT-MRAM is proposed, which has a 12.5% storage overhead. An adaptive scrubbing technique to mitigate retention faults in the cache based on STT-MRAM is proposed in [15]. They grouped the memory cells based on their retention time and adjusted the scrubbing interval with respect to the operating temperature. A training adaptation to reduce the number of HRS in the crossbar so that the number of uni-directional switching can be reduced is proposed in [6]. They also proposed a hybrid crossbar array with mixed retention cells to mitigate retention failures, but these crossbars are difficult to manufacture.

Blind scrubbing has previously been used in FPGA devices to mitigate single-event upset (SEU) errors [21]. This scrubbing technique does not require expensive checks for error detection. Instead, it blindly overwrites the specified memory region at a pre-specified frequency. Usually, blind scrubbing requires storing information about the scrub region. In this work, we employ a blind scrubbing technique while keeping the scrubbing cost to a minimum.

III. Mitigating Retention Failures through Blind Scrubbing

A. Main Idea and Motivation

As discussed before, the direction of the uni-directional state change is technology dependant. In MTJ based NVM crossbars, state changes from $+1 \dashrightarrow -1$ are far more common than retention faults in the other direction $-1 \dashrightarrow +1$. Therefore, these uni-directional retention faults accumulate over time and severely impact the inference accuracy after a certain period.

We propose an approximate scrubbing technique to prevent the accumulation of uni-directional faults. The main idea behind the approach is to define a scrub region $\mathcal{S}$ where most unstable $+1$ weights are stored. This region can then be frequently scrubbed to restore the respective values. The scrubbing frequency is optimized based on the used NVM technology, device parameters and environment such as operating temperature.

The scrubbing controller receives the information about the scrub region's shape and scrub frequency. It re-writes that region (row/column-wise) to $+1$ weights. As a result, the accumulation of errors can be prevented. Hence, degradation of the inference accuracy of the NN can be mitigated either completely or partially. While some -1 weights that may have been stored in the scrubbing region $\mathcal{S}$ will be overwritten

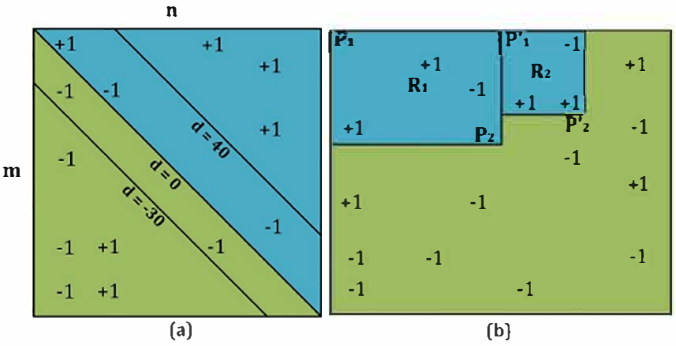

Fig. 2. (a) A Crossbar ($m = n$) showing different possible scrub and non-scrub areas with different diagonals d. (b) A Crossbar ($m < n$) showing rectangular shaped (R_1 and R_2) scrub area. Each scrub area requires storing two points (P_1 and P_2).

to $+1$s, our hypothesis is that a few of these faults will not significantly impair the inference accuracy. For the non-scrubbing region $\mathcal{S}'$, no scrubbing is performed since weights in that region are considered stable.

There are two key challenges associated with our proposed technique. Firstly, the challenge is to define the shape of the scrub region while maintaining low scrubbing costs. Each scrub area requires storing information about the size of the scrub region $\mathcal{S}$. To avoid significant memory overhead, the space complexity of the definition of $\mathcal{S}$ should ideally be kept constant, i.e., $O(1)$. Secondly, the NN training should be encouraged to store most of the $+1$ weights in $\mathcal{S}$ while minimizing the number of -1 weights. Conversely, since the weights in the non-scrubbing region $\mathcal{S}'$ are not scrubbed, it should mainly contain the stable -1 weights.

B. Proposed Scrubbing Technique

As described before, our objective is to divide the crossbar array into scrub area $\mathcal{S}$ and non-scrub area $\mathcal{S}'$, which contain most of the $+1$ and -1 weights, respectively, while keeping the cost of scrubbing to a minimum. Different shapes can thereby define the scrub areas. Two examples of $O(1)$ scrub area descriptions can be seen in Fig. 2. To define a diagonal scrubbing region as shown in Fig. 2(a), we define a diagonal d in the crossbar array, which separates the scrub area $\mathcal{S}$ from the non-scrub area $\mathcal{S}'$. The value of d is considered a hyperparameter during training. Since this definition depends only on a single integer d, the storage overhead is $O(1)$. The top-right area above d thereby defines the scrub area $\mathcal{S}$, while the bottom-left defines $\mathcal{S}'$. Increasing or decreasing d can be visualized as the diagonal moving towards top-right or bottom-left, respectively, as depicted in Fig. 2(a). The diagonal shaped scrubbing region leads to a greater variation in synaptic weights compared to the rectangular scrubbing region depicted in Fig. 2 (b). Although, the number of rectangles can be increased to increase variability, but it will require more storage.

C. Proposed Training Technique

In NN training, the loss function expresses the preference for solutions. Due to the employed scrubbing, we prefer

solutions with $+1$ and -1 in the respective regions (above or below the diagonal), as these entries will be less affected by retention faults. A penalty function augmenting the original training objective can be designed to express this preference.

As training NNs is most commonly described as a minimization problem, the penalty function should exhibit its minimum value at the most preferred configuration, i.e., where all $+1$ and -1 weights are in the correct region. Consequently, the more weights are placed outside of the respective region, the higher the penalty. Note that many such penalty functions can be designed that satisfies this property. For example

$$P(\mathrm{W}) = \sum_{w \in \mathcal{S}} (1 - w) + \sum_{w \in \mathcal{S}'} (1 + w).$$

The penalty function can now be combined with the original training objective, i.e., loss function, to form the new training objective

$$\mathrm{Loss}'(\theta, \mathcal{D}) = \mathrm{Loss}(\theta, \mathcal{D}) + \sum_{l=1}^{L} P(\mathrm{W}^l).$$

The definition of the loss function can be dynamically adjusted during training to influence the training target. For example, the loss function can be augmented by the penalty (as described in Algorithm 1) when the network starts to store weights of undesired values in the non-scrub and scrub area.

Algorithm 1 dynamically adjusts the loss function. (Non-) *Scrub Area Coverage SAC* (SAC') is given as a percentage of the desired weights in $\mathcal{S}$ ($\mathcal{S}'$). The parameter th is the corresponding augmentation threshold in percent.

for $epoch = 1$ to $epochs$ **do**
  if $SAC < th$ or $SAC' < th$ **then**
    $\mathrm{Loss}'(\theta, \mathcal{D}) \leftarrow \mathrm{Loss}(\theta, \mathcal{D}) + \sum_{l=1}^{L} P(\mathrm{W}^l)$
  else
    $\mathrm{Loss}'(\theta, \mathcal{D}) \leftarrow \mathrm{Loss}(\theta, \mathcal{D})$
  end if
end for

IV. Simulation Results

A. Simulation Setup

In this work, we focus on uni-directional faults in MTJ-based crossbars, but the proposed methods can be applied to mitigate retention faults in other emerging NVM-based crossbars.

The probability of the retention faults depends on the thermal stability factor Δ of the MTJs. In general, the lower the thermal stability factor Δ of the MTJs, the higher the switching probability and the more uni-directional faults. Higher uni-directional fault rates can lead to higher degradation in inference accuracy over the device lifetime $t = end$, or the expected time before the next weight matrix update [6].

Our training, fault-injection, and scrubbing simulation flow are shown in Fig. 3. The fault is modeled as described in [6]. We have trained a six-layer (2048 neurons) NN with three different datasets: MNIST, Fashion-MNIST, and CIFAR-3. We

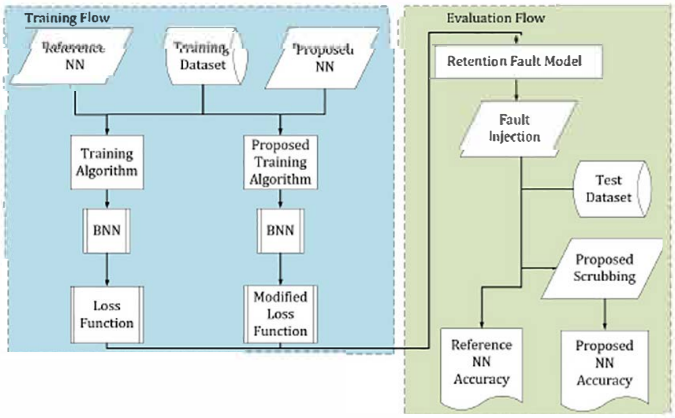

Fig. 3. Overview of NN training and evaluation flow.

have used a constant learning rate, the ADAM optimizer [22], and the cross-entropy loss function.

MNIST (handwritten digits) and Fashion-MNIST [23] have 28×28 gray-scale images representing handwritten digits ranging from 0 to 9 (10 classes), and 10 classes of clothes. Both datasets have 50K training images and 10K test images. We did not use any data-augmentation or pre-processing for these datasets. CIFAR-3 is a subset of the CIFAR-10 dataset with only three output classes compared to ten classes in CIFAR-10. It has a 15K training set and a 3K test set. Each image is colored with 32×32 pixels and represents either an airplane, a bird, or a cat. We refer to that dataset as CIFAR-3 in the following. The training and test images are resized to 28×28 as a pre-processing step.

We binarized our NN with the algorithm from [18], but initialized it with a pre-trained floating-point NN. The proposed cost function described in Section III-C is used during floating-point and BNN training.

The trained weight of the binary NN is mapped to six different MTJ-based NVM crossbars for inference. The shape of all hidden layers is $[2048 \times 2048]$, and they are mapped to MTJ-based crossbars H1, H2, H3, and H4 with a dimension (m and n) of $[2048 \times 2048]$. Due to hardware constraints, only a limited number of wordlines are activated concurrently and the partial current sum is accumulated to get the total current sum for each step described in the last paragraph of Section II-B. The activation function of each post-synaptic neuron is scaled and shifted by BatchNorm parameters, which are deterministic during inference.

The hidden layers have more parameters compared to the first and last layer. The first layer's input and the last layer's activation are non-binary. As a result, this work is focused on retention failures in hidden layers only. We assume that the input and the output layer weights are mapped to a crossbar array with a high thermal stability factor Δ.

B. Analyzing Inference Accuracy with Scrubbing

We conducted two sets of experiments for each dataset: reference and proposed. The training algorithm of the reference NN is unaltered, and it is not scrubbed during the simulation. The proposed NN model is trained with Algorithm 1, and a

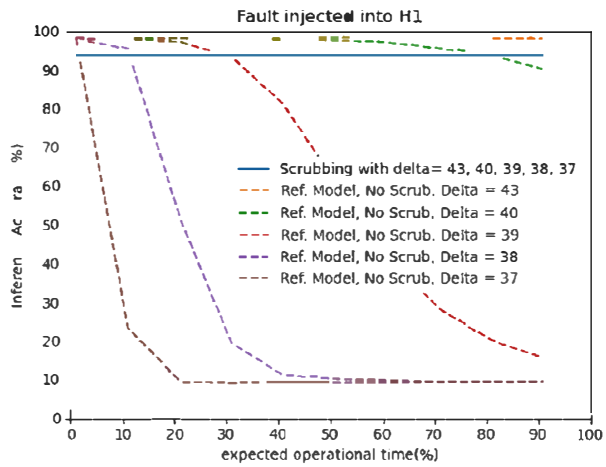

Fig. 4. Impact of different thermal stability factor on inference accuracy with MNIST dataset.

value of $th = 100$ and $d = 9$ are chosen before training. There are no weights with undesired values in S and S' after training the proposed NN model, and it is scrubbed during the simulation.

We have simulated the retention faults in ten steps, each corresponding to 10% of the expected device lifetime. The results are summarized in Table I.

Retention faults are stochastic, which is effectively random behavior. This leads the NN to predict the wrong class using faulty weights. As a consequence, the inference accuracy can vary with the same amount of failures. The results that are shown in Table I are averaged over ten runs of independent stochastic fault injections.

Our proposed scrubbing technique improves the inference accuracy by more than 82.89%, 74.46%, and 30.80% in MNIST, Fashion-MNIST, and CIFAR-3 datasets, respectively, over the expected device lifetime. Although the initial inference accuracy at $t = 0$ is slightly better for the reference model than for our proposed technique, the inference accuracy is significantly better for our proposed scrubbing technique until the end of the expected device lifetime ($t = \text{end}$). Please note that CIFAR-3 has only three output classes and the probability that the output is correct at random is 33%, which is more than three times that of MNIST and Fashion-MNIST (10%). As a result, the inference accuracy degrades to around 33% under retention faults, and our proposed scrubbing technique saves more than 30.80%.

The inference accuracy of the proposed method does not change with Δ as opposed to the reference model, as shown in Fig. 4. When the thermal stability factor Δ is above 40, the reference model performs better, but our proposed scrubbing method performs significantly better in the long run when the thermal stability is below 40.

C. Analyzing Inference accuracy with Different Scrub Area

The scrub area can be changed before training and after training by choosing different diagonal values d. In the previous section, a diagonal of 9 was chosen before training and used during evaluation.

Increasing or decreasing the diagonal d' from the value specified before training d will make the scrub area smaller or bigger. The smaller the scrub area ($d' > d$), the more synaptic $+1$ weights are subject to retention faults. On the other hand, with a larger scrub area ($d' < d$), the more synaptic -1 weights are written to synaptic $+1$ weights due to scrubbing, as shown in Table II.

Due to the approximate nature of the NN, the inference accuracy only degrades noticeably after the scrub area is made significantly bigger ($d' \ll d$) or smaller ($d' \gg d$) compared to the original scrub area defined before training. This shows the robustness of the proposed approach.

Choosing a too high value for the NN training hyper-parameter d before training will lead to a massive $t = 0$ inference accuracy degradation. The inference accuracy drops to 75.76%, and 93.10% for MNIST when a value $d = 1000$ and $d = 100$ was chosen, respectively.

D. Relaxing the requirement of scrub and non-scrub areas

In Section IV-B, we analyzed the inference accuracy when the scrub and non-scrub area only contain their desired values. However, this requirement can be relaxed without severely impacting the inference accuracy up to a certain point, as shown in Table III.

When only the non-scrub area contains undesired weights, the inference accuracy at the end of the device lifetime degrades slightly. We found that a higher number of undesired

TABLE I

EVALUATION OF DIFFERENT CROSSBARS (ONE AT A TIME) WITH OUR PROPOSED SCRUBBING TECHNIQUE AND THE REFERENCE MODEL FOR MNIST, FASHION-MNIST, AND CIFAR-3 DATASET. $\Delta = 25$ IS CHOSEN.

Dataset	Crossbar	accuracy at t=0		accuracy at t=end	
		Proposed	Ref.	Proposed	Ref.
MNIST	H1	94.21%	98.35%	94.21%	8.70%
	H2	94.21%	98.35%	94.21%	10.81%
	H3	94.21%	98.35%	94.21%	10.83%
	H4	94.21%	98.35%	94.21%	11.32%
Fashion-MNIST	H1	84.46%	90.68%	84.46%	7.98%
	H2	84.46%	90.68%	84.46%	9.52%
	H3	84.46%	90.68%	84.46%	8.79%
	H4	84.46%	90.68%	84.46%	10.00%
CIFAR-3	H1	64.2%	69.27%	64.2%	33.4%
	H2	64.2%	69.27%	64.2%	32.27%
	H3	64.2%	69.27%	64.2%	33.33%
	H4	64.2%	69.27%	64.2%	31.8%

TABLE II

THE EFFECT OF CHANGING DIAGONAL ON INFERENCE ACCURACY AFTER TRAINING. EVALUATED FOR MNIST DATASET AND H1 CROSSBAR WITH $\Delta = 25$.

Diagonal	% of $+1$ in S'	% of -1 weights in S	Accuracy at $t = end$
100	8.73%	0%	92.56%
50	3.98%	0%	94.05%
9(original)	0%	0%	94.21%
-50	0%	5.44%	92.70%
-100	0%	9.51%	85.44%

TABLE III
THE RESULT FOR THE FASHION-MNIST DATASET WITH THERMAL STABILITY FACTOR (Δ) $= 25$ WHEN ONLY NON-SCRUB AND BOTH AREAS CONTAIN UNDESIRED WEIGHTS.

Crossbar	% of $+1$ weights in S'	% of -1 in S	Accuracy $t = 0$	Accuracy $t = end$
Relaxed in Non-Scrub Area Only				
H1	1.21%	0%	87.16%	86.37%
H2	2.65%	0%	87.16%	87.03%
H3	0.80%	0%	87.16%	86.91%
H4	1.99%	0%	87.16%	87.13%
Relaxed in Scrub and Non-Scrub Area				
H1	0.45%	1.54%	85.81%	80.61%
H2	0.04%	2.81%	85.81%	85.67%
H3	1.23%	3.16%	85.81%	75.58%
H4	2.34%	4.52%	85.81%	75.95%

weights in the non-scrub area does not always lead to a lower final inference accuracy.

Allowing undesired weights in both non-scrub and scrub areas does not guarantee a higher initial inference than allowing only undesired weights in the non-scrub area. In our case, we get a slightly lower initial inference accuracy ($87.16\% \rightarrow 85.81\%$) for the Fashion-MNIST dataset. Also, the inference accuracy degrades more at $t = end$ compared to the case of only undesired weights in the non-scrub area. This is due to the combined effect of not protecting synaptic $+1$ in the non-scrub and writing -1 weights in the scrub area to $+1$.

There is a trade-off between the initial $t = 0$ inference accuracy and the final $t = end$ inference accuracy. Although allowing some undesired weights in the scrub or non-scrub areas increases the initial $t = 0$ inference accuracy, it can reduce the inference accuracy at the end of the device lifetime, especially when both scrub and non-scrub areas have undesired weights. For example, the inference accuracy of H3 (75.58%) and H4 (75.95%) at the end of device lifetime is lower in comparison (84.46%) to the result from Section IV-B for the Fashion-MNIST dataset. The results suggest that the scrub and non-scrub areas should not be relaxed at the same time and should not contain more than 3% of undesired weights, i.e., training the NN with Algorithm 1 and threshold $th = 97$.

V. CONCLUSION

In this work, we proposed an approximate scrubbing technique to mitigate uni-directional retention faults in resistive NVM-based NN accelerators. Our proposed technique defines a diagonal, which divides the crossbar into the scrub and non-scrub area. Unstable HRS cells are mapped to the scrubbing region and a scrubbing controller re-writes the scrubbing region periodically to mitigate retention faults. We introduced a training adaptation to minimize the number of LRS and HRS cells in the scrubbing and non-scrubbing regions, respectively. Our proposed scrubbing technique improves the inference accuracy over the expected device lifetime up to 85.51% with virtually zero memory overhead. It enables higher memory density of the crossbar, and a reduction in write latency and energy of certain NVMs without trading-off retention time.

REFERENCES

[1] S. Yu, "Neuro-inspired computing with emerging nonvolatile memorys," *Proceedings of the IEEE*, vol. 106, no. 2, pp. 260–285, 2018.

[2] L. Ni *et al.*, "Distributed in-memory computing on binary rram crossbar," *J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, pp. 36:1–36:18, Mar. 2017. [Online]. Available: http://doi.acm.org/10.1145/2996192

[3] G. W. Burr *et al.*, "Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element," *IEEE Transactions on Electron Devices*, vol. 62, no. 11, pp. 3498–3507, 2015.

[4] D. Zhang *et al.*, "Stochastic spintronic device based synapses and spiking neurons for neuromorphic computation," in *Proc. IEEE/ACM Int. Symp. Nanoscale Architectures (NANOARCH)*, Jul. 2016, pp. 173–178.

[5] D. Soudry *et al.*, "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, Oct. 2015.

[6] C. Münch, R. Bishnoi, and M. B. Tahoori, "Tolerating retention failures in neuromorphic fabric based on emerging resistive memories," in *2020 25th Asia and South Pacific Design Automat. Conf. (ASP-DAC)*. IEEE, 2020, pp. 393–400.

[7] K. Hofmann *et al.*, "Comprehensive statistical investigation of stt-mram thermal stability," in *Symposium on VLSI Technology: Digest of Technical Papers*, June 2014, pp. 1–2.

[8] K. Tsunoda *et al.*, "Area dependence of thermal stability factor in perpendicular stt-mram analyzed by bi-directional data flipping model," in *IEEE International Electron Devices Meeting*, Dec 2014, pp. 19.3.1–19.3.4.

[9] C. Münch, R. Bishnoi, and M. B. Tahoori, "Reliable in-memory neuromorphic computing using spintronics," in *Proceedings of the Asia and South Pacific Design Automation Conference*. ACM, 2019.

[10] A. Nigam *et al.*, "Delivering on the promise of universal memory for spin-transfer torque RAM (stt-RAM)," in *Proc. IEEE/ACM Int. Symp. Low Power Electronics and Design*, Aug. 2011, pp. 121–126.

[11] M. Zhang *et al.*, "Quick-and-dirty: Improving performance of mlc pcm by using temporary short writes," in *2017 IEEE International Conference on Computer Design (ICCD)*, 2017, pp. 585–588.

[12] R. H. Morelos-Zaragoza, *The art of error correcting coding*. John Wiley & Sons, 2006.

[13] J. Park *et al.*, "Investigation of state stability of low-resistance state in resistive memory," *IEEE Electron Device Letters*, vol. 31, no. 5, pp. 485–487, 2010.

[14] Z. Diao *et al.*, "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.

[15] N. Sayed *et al.*, "Process variation and temperature aware adaptive scrubbing for retention failures in stt-mram," in *2018 23rd Asia and South Pacific Design Automat. Conf. (ASP-DAC)*, 2018, pp. 203–208.

[16] P. Hazra and J. K. B, "Scaling of resistive random access memory devices beyond 100 nm2: Influence of grain boundaries studied using scanning tunneling microscopy," *Nanotechnology*, vol. 29, p. 495202, 12 2018.

[17] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[18] M. Courbariaux *et al.*, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[19] A. Das and N. A. Touba, "Selective checksum based on-line error correction for rram based matrix operations," in *2020 IEEE 38th VLSI Test Symposium (VTS)*, 2020, pp. 1–6.

[20] X. Guo *et al.*, "Sanitizer: Mitigating the impact of expensive ecc checks on stt-mram based main memories," *IEEE Transactions on Computers*, vol. 67, no. 6, pp. 847–860, 2018.

[21] F. Brosser *et al.*, "Assessing scrubbing techniques for xilinx sram-based fpgas in space applications," in *Int. Conf. Field-Program. Technol. (FPT)*, 2014, pp. 296–299.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[23] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.