Evaluating the Impact of Racetrack Memory Misalignment Faults on BNNs Performance

Leonard David Bereholschi[1], Mikail Yayla[1], Jian-Jia Chen[1], Kuan-Hsun Chen[2], and Asif Ali Khan[3]

[1] TU Dortmund University, Germany
{leonard.bereholschi, mikail.yayla, jian-jia.chen}@tu-dortmund.de
[2] University of Twente, the Netherlands
k.h.chen@utwente.nl
[3] Technical University of Dresden, Germany
{asif_ali.khan}@tu-dresden.de

Abstract. Racetrack memory (RTM) is a promising non-volatile memory (NVM) technology that offers exceptional density, power and performance benefits over other NVM and conventional memory technologies. RTM cells have the unique capability of storing hundreds of data bits per cell and are equipped with one or more access ports. However, accessing data in an RTM cell requires the data to be *shifted and aligned* to an access port, introducing performance and energy overheads and potentially leading to *misalignment faults*. A misalignment fault occurs when after the shift operation, the desired data is not properly aligned to an access port and incorrect data is read from the RTM cell. Countermeasures have been proposed to mitigate the effects of these faults on applications' accuracy, albeit at the cost of increased overhead. There is potential to balance the trade-offs between acceptable drops in accuracy and enhancements in performance, especially in error-resilient applications such as *Binarized Neural Networks* (BNNs). However, there exists no tool that enables effective exploration of this design space and assess the potential trade-offs.

This paper introduces NetDrift, a framework which facilitates investigation into the impact of RTM misalignment faults on BNNs accuracy at finer granularities. It enables controlled error injection in selected BNN layers with varying fault rates and simulates the impact of accumulated errors in weight tensors of several BNN models (FashionMNIST, CIFAR10, ResNet18) stored in RTM. The framework allows for tuning reliability for performance and vice versa, providing an estimate of the number of inference iterations required for a BNN model to drop below a certain lower threshold, with no protection, limited protection, and full protection, along with the associated impact on performance. The tool is openly available on Github[4].

Keywords: Binarized neural networks (BNNs) · Racetrack memory · Reliability · Accuracy · Misalignment Faults.

[4] https://github.com/LeonardDavid/NetDrift

1 Introduction

In recent years, machine learning has experienced remarkable progress, especially with the advancements in generative AI and large language models (LLM), revolutionizing numerous aspects of our lives. However, the complexity of these models demands extensive computational and memory resources, exposing the limitations of traditional technologies and computing paradigms. On the computing front, innovative system designs such as Google's tensor processing units (TPUs) [12], domain-specific accelerators [1, 2], near-memory, and in-memory computing systems [14], and chiplets [19] have been proposed to meet the compute demands. However, the memory subsystem unfortunately continues to rely on conventional SRAM and DRAM technologies, which face significant technological limitations. These technologies not only suffer from larger feature size but also exhibit substantial energy consumption [5]. To overcome these challenges, a multitude of novel nonvolatile memory (NVM) technologies including spin-transfer-torque memory (STT-RAM), phase-change memory (PCM), resistive RAM (ReRAM), magnetic RAM (MRAM), Ferroelectric FETs (FeFETs), and racetrack memory (RTM) have emerged [5]. Notably, RTMs stand out as particularly interesting, offering unprecedented densities and boasting latencies comparable to SRAM [4]. Since their inception in 2008 [21], RTMs have achieved substantial breakthroughs and have been deployed at various levels in the memory hierarchy [4].

Unlike conventional memory technologies, a single cell in RTM is a magnetic nanowire that can store hundreds of data bits, as illustrated in Fig. 2. Each RTM cell has one or more access ports (AP) that enable the read/write operations. During an RTM access, the desired data must be *shifted and aligned* to an AP position before it can be accessed. The shift operations not only impose performance and energy overhead but can also lead to potential *misalignment faults*, also referred to as position errors. Common position errors in RTM, such as stop-in-the-middle and out-of-step scenarios [26], occur when the adjacent domain, rather than the desired one, aligns with the AP position, resulting in the erroneous reading or writing of data.

RTM errors have been extensively studied and countermeasures like *position error correction codes* (PECCs) have been proposed to detect and correct misalignment faults [15, 26]. However, these schemes incur significant latency and energy overhead of up to 2× [26]. For applications where a slight drop in accuracy is acceptable, this overhead can be significantly reduced by selectively applying PECCs to only critical application regions. For instance, in image and video processing, edge pixels hold more significance compared to areas where surrounding pixel values are uniform. Even if a pixel in these regions is erroneously read, the overall impact on the image remains minimal. Similarly, certain machine learning models like *Binarized Neural Networks* (BNNs) [11], where weights and activations are represented in 1-bit, exhibit outstanding tolerance to bit errors [6, 8]. In such cases, the overhead of PECC schemes can be reduced by selectively protecting sensitive regions (e.g. BNN layers). To achieve this, new simulation tools are needed that are capable of identifying such important regions through

detailed analysis and characterization of BNNs at a finer granularity, and allow conducting design space exploration to strike a balance between accuracy and performance overhead.

To this end, this paper presents NetDrift, a framework that enables investigating the impact of RTM misalignment faults on the BNNs accuracy. The framework allows for different mappings of BNN weights to the RTM arrays, controlled error injection across various BNN layers with adjustable error rates, varying the RTM nanowire size, and simulating the accumulated error impact on the overall accuracy. Additionally, it evaluates the combined effects on accuracy and performance overhead incurred by a given PECC scheme, e.g., P-ECC, P-ECC-O [26], and GROGU [15], across three BNN protection levels – unprotected (no layers protected), partially protected (sensitive layers protected), and fully protected. We showcase the versatility of our framework by conducting thorough evaluations of RTM reliability and BNN performance across different RTM sizes, fault rates, datasets, and models.

The rest of the paper is structured as follows: Section 2 presents a background on RTMs, BNNs, and the state-of-the-art PECC schemes. In Section 3, we elaborate on our framework, including detailed descriptions of its individual modules. Section 4 presents our evaluation results and discussions, while Section 5 concludes the paper.

2 Background

This section provides background on BNNs, RTMs, misalignment faults and PECC schemes. Additionally, it presents a summary of the relevant state-of-the-arts and a motivational example to emphasize the finer granularity analysis.

2.1 Binarized Neural Networks

Binarized neural networks (BNNs) [11] are a resource-efficient variant of NNs, in which the weights and the activations are binarized into 1-bit representations. Unlike the full-precision NNs, where one matrix multiplication must be performed for computing the output of each neuron, BNNs apply bitwise operators for computing the outputs of neurons. Notably, BNNs achieve comparable accuracy to full-precision NNs and requires considerably less resources, e.g. [11] achieves in practice 23× inference time improvement, compared to baseline NNs.

We assume for a certain layer a weight matrix $\mathbf{W}$ with dimensions $(\alpha \times \beta)$, where α is the number of neurons and β the number of weights of a neuron. The input matrix $\mathbf{X}$ has dimensions $(\gamma \times \delta)$, where $\beta = \gamma$ and δ is the number of convolution windows in the input. We leave out any layer indices for brevity. Every convolution of a conventional NN can be mapped to this matrix notation. Unlike the full-precision NNs, where one matrix multiplication must be performed for computing the output of each neuron, we can simply apply XNOR on the operands to compute the outputs.

$$2 \cdot \text{popcount}(XNOR(\mathbf{W}, \mathbf{X})) - \#bits > \mathbf{T}, \tag{1}$$

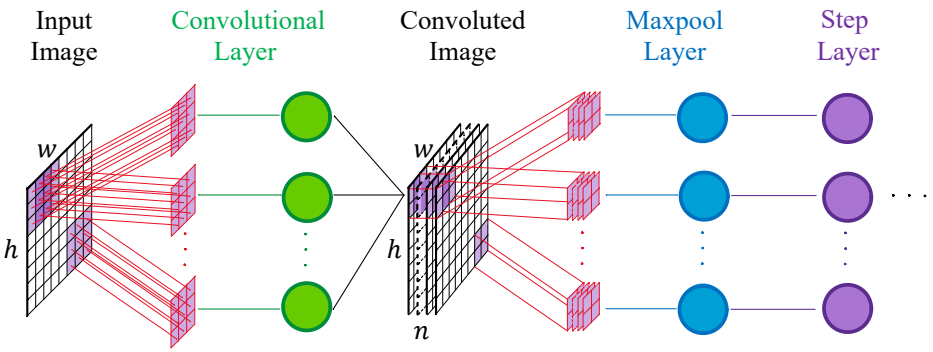

Fig. 1: Structure of a Convolutional BNN model demonstrating three major layer types: *Convolution*, *Maxpool*, and *Step* layer.

where $XNOR(\mathbf{W}, \mathbf{X})$ computes the XNOR of the rows in $\mathbf{W}$ with the columns in $\mathbf{X}$ (analogue to matrix multiplication), popcount counts the number of set bits in the XNOR result, $\#bits$ is the number of bits in the XNOR operands, and $\mathbf{T}$ is a vector of learnable threshold parameters, with one entry for each neuron. The thresholds are computed with the batch normalization parameters, i.e. $T = \mu - \frac{\sigma}{\psi}\eta$, where each neuron has a mean μ and a standard deviation σ over the result of the left side of Eq. (1), and ψ and η are learnable parameters (details about the batch normalization parameters can be found in [11, 22]). Finally, the comparisons against the thresholds produce binary values.

Target BNN models: In this paper, we consider convolutional BNNs and a binarized version of a Residual Network (ResNet18). In a standard convolutional BNN, there are four fundamental types of layers: *convolutional*, *maxpool*, *step*, and *fully-connected*. The *convolutional* layer (Cχ) conducts a 2D convolution of the input with 3×3 filters, with χ representing the number of neurons in the layer. The *maxpool* layer (MPχ) downsamples inputs by selecting the maximum in a 2×2 window, with χ indicating its output size. A *step* layer (S) incorporates batch normalization [22] followed by a binary activation function. We use signed integers for the threshold values in the batch normalization, and apply Hard-Tanh as our binary activation function. The *fully-connected* layer (FCχ), connects all neurons in the current layer with those in the next layer and also uses binary weights. Additionally, a *flattening* layer (FLAT) reshapes high-dimensional matrices into lower-dimensional matrices (3D into 1D in our models).

We also use a Residual network (ResNet18) adapted for the binarized version of NNs. In addition to the aforementioned layers, it also contains shortcut connections that allow parameters to skip multiple layers inside *Skip-connection blocks* (SCBs), in order to reach deeper into the network. The main advantage of this structure is that it mitigates the effects of the vanishing gradient problem, enabling the model to be applied to more complex datasets [7, 9].

2.2 Racetrack Memory

RTM is one of the many emerging NVM technologies that promises unprecedented density and energy benefits. A cell in RTM is a magnetic nanowire (also called track) that is composed of multiple tiny magnetic regions referred to as *domains*, which are separated by domain walls, and one or more access ports that are used to access data domains in the track, as shown in Fig. 2. Each domain in an RTM track represents a data bit and has its own magnetization direction. In order to access a domain in a track, a shift current is sent from one end of the track to shift the domains and align the corresponding data bit with a port position. To avoid data loss, each track has reserved domains at both ends of the nanowire that store data bits during the shift operation. The number of reserved domains in a track is dependent upon the number and position of access ports per track. In the case of evenly distributed access ports, the number of reserved domains is equal to the maximum shift distance. In this work, we assume one AP per track.

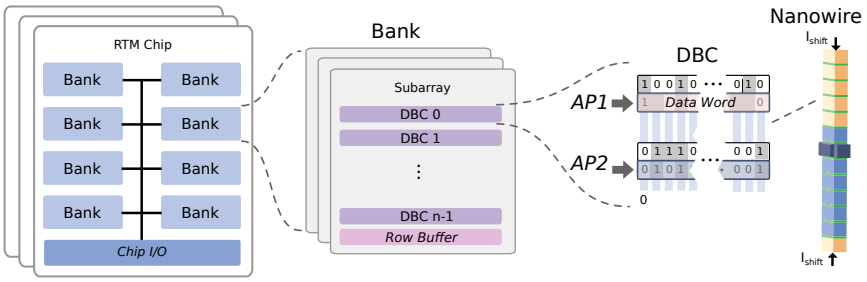

Fig. 2: RTM architectural overview

RTM, like other memory technologies, is organized into multiple banks, each containing one or more subarrays (see Fig. 2). Each subarray consists of one or more tiles, with each tile having multiple *domain wall clusters* (DBC). A DBC is a group of T nanowires, each containing K data domains (i.e., bits), and are equipped with one or more access ports for performing read/write operations. Typically, data is stored in a bit-interleaved manner across DBC tracks so that all T bits of a memory object can be accessed in parallel. To access a memory object, bits are shifted in a lock-step manner until they are properly aligned with the access port positions. Unlike domain wall RTMs, where data is stored in domains, the Skymiron-based RTM stores data in magnetic skyrmions and have been proposed as a more dense and more stable alternative [13]. However, due to the lack of its fault model, this work only focuses on domain walls. Nevertheless, the proposed framework is easily extendable to accommodate other fault models.

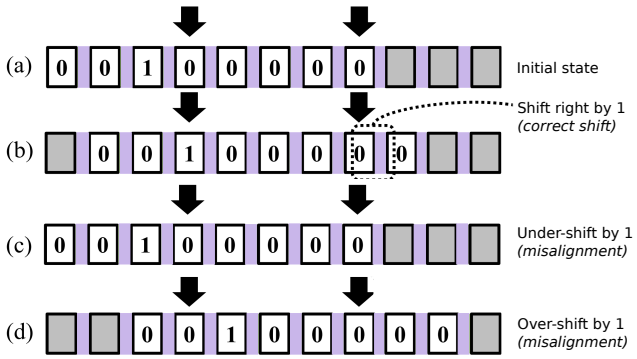

Fig. 3: Misalignment after a right shift by one position (a) Initial state before shifting (b) No error (c) Misalignment by one position (under-shift) (c) Misalignment by one position (over-shift). Adopted from [26].

2.3 Misalignment Faults

Misalignment faults (also known as position errors) may occur due to the fluctuation in the shifting current originated from variation in the operating conditions of the system or the non-linearities of the device itself [4]. The most common types of faults that can occur in an RTM are *stop-in-the-middle* and *out-of-step* misalignments. During an RTM access, the domain walls are shifted along the track until the desired location is reached. However, due to the aforementioned reasons, there exists certain probabilities that domains will be misaligned to the access ports. In the case of a stop-in-the-middle fault, which rarely occurs, the access port is aligned with a domain wall that separates two adjacent domains. The value read out is uncertain, it can either be random, or one of the values from the neighbouring domains. In the more common out-of-step misalignment, the domain is either under- or over-shifting by n positions, as illustrated in Fig. 3 for $n = 1$. Zhang et al. [26] characterized the misalignment probabilities as $P_1 = 4.55 \cdot 10^{-5}$ and $P_2 = 1.37 \cdot 10^{-21}$ for $n = 1$ and $n = 2$, respectively.

Conventional error correction codes are unfortunately not applicable to RTM misalignments because: (1) despite being correctable, misalignment faults may not always produce an immediate detectable error, (2) the misalignment fault changes the state of the entire nanowire, instead of a single bit. Many RTM-specific position error correction schemes are proposed of late. Zhang et al. [26] proposed P-ECC and P-ECC-O with various protection capabilities and overhead trade-offs. Ollivier et al. [20] proposed DECC to minimize the overhead of P-ECC and P-ECC-O by leveraging the transverse read access mode of RTM to quickly determine any potential misalignment. GreenFlag and Foosball make a correlation of the misalignment faults with the repeated and dropped bits in a communication channel and use VT codes to detect and correct RTM misalignment faults [3,17]. The most recent reliability technique, GROGU [15], proposed a novel scheme that is capable of handling misalignment by more positions with the least performance, area and energy overheads.

2.4 Motivation

To motivate the need for our framework, let us consider the example of binarized VGG7 and ResNet18 structures, where all weight tensors are stored in RTM. Figure 4 shows the accuracy of the two networks without employing any PECC scheme (unprotected) compared to when all layers in the networks are protected (fully-protected). As shown, the overall accuracy experiences a significant drop to an unacceptable level within just a few tens of iterations without protection. Conversely, with full protection, the networks maintain their baseline accuracy, albeit with an increase of, in some cases, over 100% performance overhead. The vast design space between these extremes offers opportunities to explore trade-offs in accuracy and performance. Our tool, as demonstrated in Section 4.4, reveals that selective protection of specific layers, identifiable using our framework through sensitivity analysis, can sustain accuracy for a longer duration with a $5-96\%$ performance overhead (for ResNet18), depending on the nanowire size and the employed protection scheme (GROGU [15] in this case).

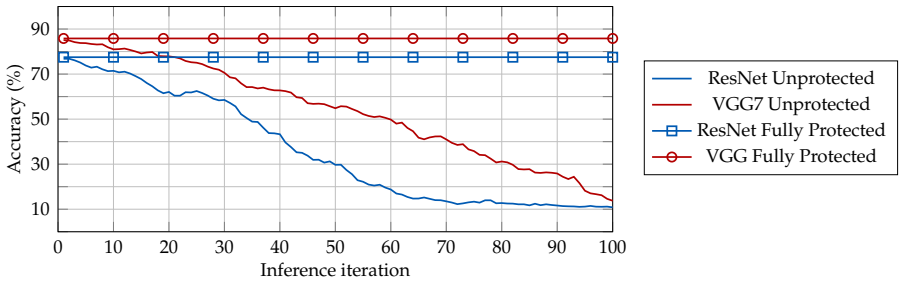

Fig. 4: Accuracy drop of ResNet and VGG7 models over time across inference iterations. The baseline accuracy is the same as in fully-protected configurations.

3 NetDrift Overview

This section presents our framework (NetDrift) which enables profiling BNNs on unreliable RTMs. Our framework is based on the SPICE-Torch error injection tool [25] that models injecting conventional memory faults, such as like bit-flips in BNN weights during inference and assessing their resilience. Contrary to existing frameworks for conventional permanent and transient faults, NetDrift specifically focuses on the RTM misalignment faults, comprising a mapping module, a fault injector, and accounting for parameters such as RTM track size and misalignment fault rate. Figure 5 illustrates a high-level overview of the operational steps of NetDrift.

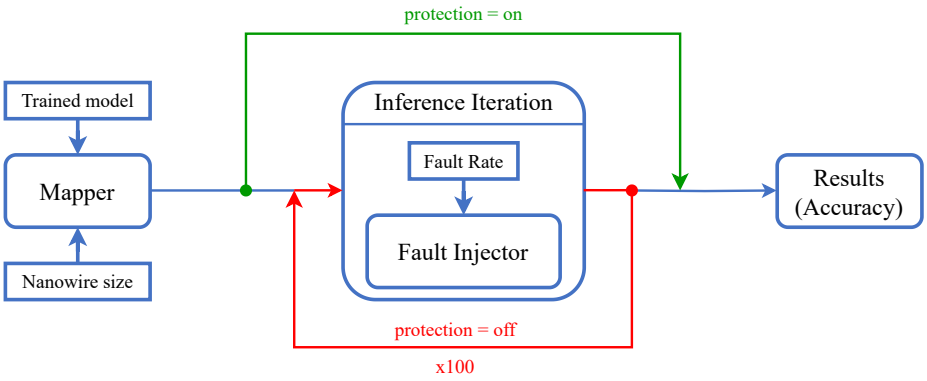

Fig. 5: A high-level overview of our NetDrift simulation framework.

3.1 Mapping of BNN Weights to RTM

We train different BNN models without any misalignments and map the weight tensors to RTM nanowires. The binary weight tensors (one per network layer) are split into blocks of size K, where K is the number of bits in the nanowire (typically $K \leq 64$). Hence, multiple nanowires are used to store a single weight tensor. The framework currently allows mapping the weight matrices to RTM in both row-major and column-major orders, but an arbitrary mapping can also be implemented. We assume a single access port per track, which requires resetting the AP to the beginning of the nanowire after every inference iteration. Nevertheless, the framework can also be used with the zig-zagged RTM accesses using two APs to avoid the long jumps after every iteration [18].

3.2 Fault Injection

We presume that all BNN weights are stored in RTM, and that an inference does not require any weight updates (i.e. write operations). In the course of an inference, the weights are fetched (i.e. read) from the memory, and each access mandates shifting the nanowire to the desired port position. The shift direction and distance depend upon the requested data and the current position of the AP. As input, the fault injector takes a fault rate and the BNN layers that need to be protected, along with the chosen PECC scheme. By default, the framework inserts faults in all layers and does not employ any protection. For each shift operation, the fault injector inserts a misalignment fault by one position, according to the provided fault rate. Note that while the fault injection rate governs the rate of fault injection, the control of under-shift and over-shift rates is not explicitly determined, following a uniform distribution.

Please note that, once a fault is injected, the values in the shifted nanowires remain misaligned so that the impact of faults is accumulated across multiple inference iterations. For example, if a nanowire in iteration i over-shifts by one position to the right, and the same nanowire in iteration $i+1$ over-shifts to the right again, the overall misalignment accumulates to over-shift by *two* positions to the right. Conversely, subsequent misalignments, such as an over-shift followed

by an under-shift (or vice versa), counteract each other, potentially bringing the nanowire back to the correct position.

3.3 Error Detection and Correction

NetDrift monitors the injected faults by the fault injector, along with the total shifts count, to estimate the performance overhead of the given PECC scheme. When protection is activated on a layer, the framework simulates the process as follows: The fault injector triggers as if it would inject a fault in the nanowire, tallying the fault count for the protection overhead estimation. However, no error is injected into the nanowire, thus necessitating no correction. For all correctable faults by the selected PECC, this yields similar outcomes to implementing the PECC scheme directly within the simulator.

3.4 Applicability to High-precision Networks

This work is also applicable to high-precision neural networks. Here, we briefly explain how to modify the framework to implement and test such neural networks. First, the mapping module takes the weight matrices of the custom (trained) NN model along with the nanowire size, bit-slices them, and stores them in the RTM nanowires. The fault injector treats every nanowire independent and can be used out of the box without requiring any modifications. If needed, it can also be extended or used as a reference. In the provided GitHub repository (see link in the abstract), we explain how certain modules of the framework can be modified to achieve this. Finally, the accuracy of the framework in the presence of the misalignment faults is evaluated.

4 Evaluation

This section presents our experimental setup, including our datasets and BNN architectures. Additionally, it presents our evaluation results for conducting sensitivity analysis on various BNN models and our design space exploration to balance the accuracy and performance tradeoffs.

4.1 Experimental setup

We use a Linux server with two GTX1080 cards, each having 8GB of VRAM, to run our experiments. The server can handle four concurrent and independent runs. A run means executing a BNN model in our NetDrift framework for 100 consecutive inference iterations.

As explained in Section 3.2, the errors also accumulate over multiple inference iterations. In our evaluations, misalignment faults are accumulated over the course of 100 consecutive inference iterations. We focus only on the impact faults have on the accuracy of the models described in Section 4.2. The evaluated misalignment fault rates $p \in \{10^{-4}, 4.55 \times 10^{-5}, 10^{-5}\}$ are taken from [15, 26]. The simulator generates results including details about each inference iteration, such as accuracy after each iteration and the total (accumulated) misalignment faults in each layer. Since the models have different sizes (as outlined in the

following Section 4.2), *one* inference iteration takes on average 17 seconds for VGG3 (FMNIST), 39 seconds for VGG7 (CIFAR), and around 7 minutes for ResNet18 (ImageNette).

For error detection and correction, various PECC schemes are available, as discussed in Section 2.3, which can be used with NetDrift to estimate performance overhead. We use the state-of-the-art GROGU scheme [15] that operates by conducting a transverse read and a standard read operation to detect a misalignment by $\pm w$ positions, where w is the window size for the transverse read operation [15]. Assuming each of these read operations takes one cycle, fault detection requires two cycles and occurs after every shift operation in the protected layers. Upon detecting an error, GROGU proceeds with correction, involving shift and write operations. On average, GROGU typically requires around four cycles to correct a fault. While other schemes like P-ECC and P-ECC-O [26] exist, with varying detection and correction overhead cycle counts generally exceeding those of GROGU, in this section we focus exclusively on GROGU for the sake of brevity.

4.2 Datasets and BNN Models

We evaluate our framework on several commonly used datasets, for which we used BNN models based on VGG-type architectures [23] (in the case of CIFAR10 and Fashion-MNIST datasets) and ResNet18 for ImageNette.

Fashion-MNIST: The Fashion-MNIST [24] dataset consists of $70,000$ grayscale images and labels from 10 classes, representing different clothing articles. The size of each image is 28×28 pixels in 1 channel, with 0 representing the *brightest* and 255 the *darkest* values. We use $60,000$ images for training and the remaining $10,000$ for testing. Table 1.1 defines the BNN structure of the VGG3 model used for this dataset, using the notations from Section 2.1. The baseline accuracy for this model is 91.08%.

CIFAR10: The CIFAR10 [16] dataset contains $60,000$ colour images (3 channels), each with a size of 32×32. It is split into $50,000$ training and $10,000$ test images, which are classified in 10 different classes representing means of transportation (i.e. airplane, ship, truck, automobile) and animals (i.e. bird, cat, dog, deer, frog, horse). For this dataset we used a bigger VGG7 model, with its BNN structure illustrated in Table 1.2. It achieves a baseline accuracy of 85.82%.

ImageNette: ImageNette [10] is the subset of the larger ImageNet dataset. It classifies 10 various classes of 1000 images each, at a resolution of 64×64. Compared to the larger original dataset, ImageNette is more manageable and time efficient for running multiple inference iterations, necessary for our evaluations. The BNN model used for the ImageNette dataset is a ResNet18 structure with a baseline accuracy of 77.50%, following the schematics of Table 1.3. It consists of convolutional layers, divided into multiple *Skip-connection blocks* (SCBs), which contain a shortcut connection that skips some of the layers inside the block. The blocks are made up of 4 convolutional layers, with the number χ after $SCB\chi$ denoting the amount of neurons in both of the convolutional layers.

Table 1: Structure of the BNN Models.

1. VGG3

In → C64 → MP14 → S → C64 → MP7 → S → FLAT → FC2048 → S → FC2048

2. VGG7

In → C64 → S → C64 → MP16 → S → C256 → S → C256 → MP8 → S → C512 → S → C512 → MP4 → S → FLAT → FC1024 → S → FC1024 → 10

3. ResNet18

In → C64 → SCB64 → SCB128 → SCB256 → MP2 → SCB512 → MP4 → FC10

4.3 Layer-wise Sensitivity Analysis

To give an overview on the individual impact of each layer, we present the accuracy drops in a heatmap as Figure 6, where darker reddish spots mean a bigger accuracy drop from the baseline. The rows represent the ordered layers of the evaluated model, while the columns denote different RTM nanowire sizes (varied from 2 to 64). The heatmap is generated as follows: For a given RTM nanowire size and a given fault rate, we inject faults in merely one layer (i.e., keeping all other layers protected) and observe the accuracy over 100 inference iterations. Each cell in the heatmap shows the accuracy after these 100 iterations. We repeat this experiment for each network layer (different rows), with different RTM nanowire sizes (columns) and different fault rates (different blocks in the heat map). This allows us to evaluate the sensitivity of each layer individually. Due to space constraints, we chose to only show the heatmaps for the VGG7 and ResNet18 models, with the realistic fault rates of $p \in \{10^{-4}, 4.55 \times 10^{-5}, 10^{-5}\}$.

Firstly, we can observe that, for smaller nanowire sizes, the probability of faults is smaller due to fewer shifts. For instance, when the nanowire has a size of 2 bits, the average accuracy drop across all layers is only around -1%. In most cases, this drop is deemed acceptable, thus obviating the need for error correction. However, common nanowire sizes tend to be 32 and 64, and using only two locations per nanowire significantly underutilizes the RTM. The accuracy drop increases, along with the increase of the nanowire sizes. For a nanowire size of 64, the accuracy drop can be as high as -75% (in VGG7) and -57% (in ResNet18) in selected layers, necessitating protection of RTM accesses.

Additionally, we also observe that certain network layers exhibit higher susceptibility to misalignment faults compared to others. The sensitivity of these layers varies across network models, yet the relative accuracy drop remains consistent for different block sizes, as illustrated in Figure 6.

4.4 Design Space Exploration

Applying a PECC scheme can mitigate or even entirely prevent accuracy drops resulting from misalignment faults. However, this comes at the cost of performance overhead, namely an increase in inference execution time, as the employed

a. VGG7 (CIFAR10)

Layer	10^{-4}						4.55×10^{-5}						10^{-5}					
	64	32	16	8	4	2	64	32	16	8	4	2	64	32	16	8	4	2
1	-0.40	-0.40	-0.51	-0.16	-0.45	-0.31	-0.42	-0.32	-0.13	0.00	-0.11	0.00	-0.13	-0.23	0.00	0.00	0.00	0.00
2	-69.8	-63.3	-45.9	-33.2	-11.1	-5.61	-62.1	-50.1	-30.2	-18.9	-5.87	-2.36	-19.8	-17.1	-4.00	-2.64	-0.53	-0.50
3	-67.4	-64.1	-53.3	-20.8	-7.07	-2.5	-65.7	-40.6	-28.9	-8.38	-2.38	-0.74	-16.2	-11.4	-2.91	-0.89	-0.90	-0.68
4	-75.8	-75.8	-74.4	-43.9	-15.5	-1.56	-75.8	-74.0	-37.3	-14.0	-2.35	-0.79	-44.4	-18.1	-5.45	-3.23	-0.50	-0.14
5	-65.2	-65.2	-58.7	-43.6	-12.3	-1.93	-59.9	-49.8	-29.3	-13.9	-2.94	-0.72	-22.9	-8.07	-2.75	-2.52	-0.46	-0.36
6	-75.8	-75.4	-62.0	-27.7	-5.37	-1.56	-75.7	-65.7	-31.5	-6.96	-2.42	-0.80	-24.7	-7.58	-3.87	-1.57	-0.57	-0.26
7	-75.8	-75.8	-67.7	-20.7	-2.93	-0.45	-75.8	-63.0	-19.4	-3.91	-0.77	-0.14	-18.4	-3.21	-1.01	-0.31	-0.15	-0.06
8	-34.7	-23.7	-20.3	-7.64	-1.58	-0.97	-20.3	-16.1	-7.96	-1.47	-0.84	-0.82	-6.52	-1.56	-0.54	0.69	-0.70	-0.41

b. ResNet18 (ImageNette)

Layer	10^{-4}						4.55×10^{-5}						10^{-5}					
	64	32	16	8	4	2	64	32	16	8	4	2	64	32	16	8	4	2
1	-0.53	0.02	-0.23	-0.60	0.00	0.11	-0.51	-0.46	-0.79	-0.63	-0.63	0.00	-0.71	-0.46	-0.63	-0.58	0.00	0.00
2	-10.4	-9.68	-8.26	-6.50	-3.62	-1.73	-9.47	-9.17	-6.75	-3.67	-1.78	-0.12	-5.60	-3.08	-2.29	-1.07	-1.04	-0.35
3	-12.0	-11.7	-11.3	-7.80	-2.63	-1.86	-9.73	-6.29	-4.20	-4.28	-1.60	-1.27	-6.26	-2.82	-0.61	-0.86	-0.96	-0.58
4	-30.5	-23.0	-22.5	-22.6	-5.07	-3.49	-29.7	-24.9	-4.74	-9.60	-1.50	-0.96	-11.9	-9.78	-2.65	-1.68	-0.58	-0.61
5	-12.9	-16.6	-13.4	-5.86	-5.07	-0.97	-20.7	-13.1	-6.32	-7.13	-5.12	-0.96	-3.28	-1.88	-1.30	-0.99	-0.74	-0.23
6	-57.0	-48.1	-47.7	-46.5	-16.9	-3.31	-48.6	-41.3	-33.6	-14.4	-5.27	-0.91	-40.5	-20.8	-7.13	-2.09	-0.71	-0.61
7	-52.1	-49.3	-45.2	-40.7	-13.7	-3.11	-51.4	-46.8	-29.8	-20.5	-4.94	-0.89	-32.2	-27.7	-6.70	-2.06	-1.47	-0.81
8	-19.1	-18.0	-16.3	-8.69	-3.47	-0.97	-11.9	-10.0	-7.59	-2.34	-0.99	-0.28	-8.68	-2.95	-1.45	-1.45	-0.33	0.00
9	-18.1	-17.1	-11.5	-6.78	-2.50	-1.07	-20.4	-9.83	-9.37	-1.98	-0.89	-0.30	-6.29	-2.29	-1.02	-0.12	-0.48	-0.18
10	-10.9	-7.19	-5.27	-4.87	-1.53	-0.13	-7.61	-7.00	-6.14	-1.93	-0.84	-0.23	-3.23	-2.60	-0.99	-0.89	-0.81	-0.38
11	-46.5	-42.7	-33.5	-19.6	-9.45	-1.15	-35.0	-28.3	-18.6	-6.82	-2.90	-0.51	-17.9	-7.33	-4.30	-1.35	0.05	-0.07
12	-12.5	-11.6	-9.27	-5.27	-1.53	-0.61	-12.4	-8.79	-5.30	-1.98	-0.74	-0.05	-5.17	-2.19	-1.30	-0.91	-0.35	-0.10
13	-8.53	-5.35	-4.77	-3.64	-1.38	-0.56	-6.04	-4.86	-2.42	-1.63	-0.71	-0.40	-2.88	-1.58	-0.28	-0.58	-0.33	-0.76
14	-9.56	-5.89	-3.80	-2.70	-0.33	0.08	-6.49	-5.88	-2.72	-0.61	-0.18	-0.48	-2.29	-1.07	-0.20	0.13	-0.43	-0.46
15	-7.82	-5.40	-5.00	-3.62	-1.28	-0.23	-5.78	-5.37	-2.95	-2.37	-0.96	-0.33	-1.81	-0.63	-0.56	-0.66	-0.74	-0.10
16	-52.8	-46.0	-36.2	-7.84	-5.73	-1.61	-44.7	-31.0	-17.6	-7.26	-2.72	-0.66	-18.8	-5.88	-2.67	-0.79	-0.79	-0.46
17	-45.2	-35.4	-25.6	-12.2	-3.82	-0.48	-34.9	-24.7	-13.7	-4.63	-1.63	-0.58	-11.6	-4.25	-1.42	-1.27	0.05	-0.33
18	-3.21	-2.06	-2.29	-1.38	-0.33	-0.41	-1.98	-1.75	-1.09	-0.48	-0.25	-0.23	-0.84	-0.18	-0.05	-0.53	-0.28	-0.07
19	-51.6	-40.3	-22.8	-9.30	-2.27	-0.46	-31.1	-21.5	-8.71	-2.06	-0.99	-0.18	-7.54	-2.77	-1.37	-0.02	-0.05	-0.10
20	-56.8	-51.3	-30.2	-7.31	-0.46	-0.20	-45.0	-31.8	-7.31	-0.18	-0.07	-0.02	-6.70	-0.48	0.00	0.05	-0.07	-0.15
21	-61.4	-62.1	-43.2	-25.0	-21.4	-2.93	-59.5	-55.5	-32.5	-14.2	-3.74	-1.07	-38.8	-3.95	-4.15	-2.04	-0.89	-0.76

Fig. 6: Heatmaps representing the accuracy drop each layer causes individually, for different block sizes and error rates. The baseline accuracy for VGG7 is 85.82% and for ResNet18 is 77.50%.

PECC scheme requires a certain number of cycles to detect and correct misalignment faults. This subsection explores this design space by evaluating the following configurations and presenting results analysis.

- *UNPROT:* This configuration is fully unprotected, where faults are injected into all network layers, and the accuracy results are reported.
- *EVEN:* In this configuration, protection is naively applied to all odd layers while leaving *even* layers unprotected, i.e., 50% of layers are protected.
- *EQUAL:* Similar to *EVEN*, this configuration also protects 50% of the layers. However, the layers are greedily selected based on their importance, i.e., based on the results of the sensitivity analysis in Section 4.3.
- *CUSTOM:* In this configuration, protection is exclusively applied to all critical layers — those with the most significant impact on accuracy — while leaving all other layers unprotected, usually $< 25\%$ of all layers.
- *FULL:* This configuration represents complete protection, maintaining baseline accuracy by protecting all layers.

For the RTM nanowire size of 32, Figure 7 shows the accuracy drop across inference iterations for different BNN models and configurations. The red and blue lines are the same as in Figure 4, namely the *FULL* configuration (maintaining baseline accuracy) and the *UNPROT* configuration, respectively. In all three models, the accuracy of the *UNPROT* configuration drops below 30% in less than 10 to 20 iterations. The *FULL* configuration maintains the baseline accuracy but at the cost of 87% and 109% performance overhead increase in ResNet and VGG, respectively.

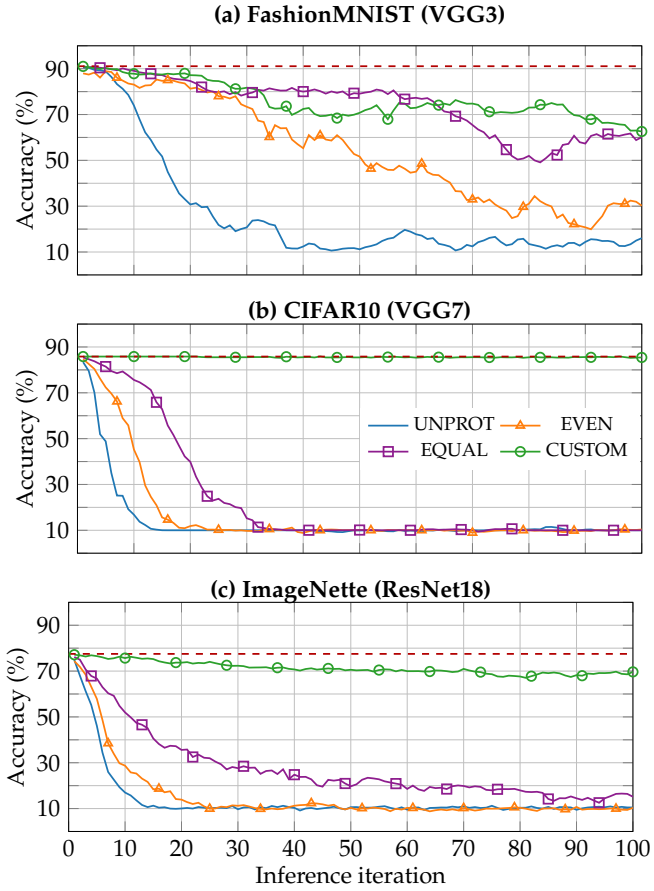

Fig. 7: Accuracy course over 100 inference iterations for different BNN models. The size of the nanowire tracks are 32 bits and the fault rate is $p = 10^{-4}$.

To reduce this overhead, *EVEN, EQUAL* and *CUSTOM* selectively apply error protection to the network layers. The EVEN configuration slightly delays the accuracy drop but, since it is naively protecting network layers without considering their importance, in general it follows a similar trend as the unprotected configuration and increases the performance overhead by 51% (ResNet), and 105% (VGG), compared to the baseline *UNPROT*. The carefully chosen *EQUAL* configuration significantly delays the accuracy drop with a 72% increase in the

Model	VGG3 (FMNIST)						VGG7 (CIFAR10)						ResNet18 (ImageNette)					
Size	64	32	16	8	4	2	64	32	16	8	4	2	64	32	16	8	4	2
UNPROT	10.3	16.1	17.3	35.9	79.5	90.2	10.0	10.0	10.0	10.3	13.7	70.1	9.5	10.4	10.1	10.1	10.8	51.2
EVEN	20.7	30.2	67.4	81.8	88.0	89.7	9.9	10.3	10.0	11.8	34.5	76.6	9.3	10.3	10.3	10.2	24.0	51.6
EQUAL	38.3	60.1	54.0	74.0	90.1	90.5	10.0	10.0	10.4	29.4	63.5	79.4	9.8	15.1	18.7	25.5	55.1	73.5
CUSTOM	61.8	62.5	84.3	86.1	90.1	90.9	85.5	85.4	85.4	85.3	79.2	85.4	67.7	69.6	70.1	73.3	75.3	76.3
FULL	91.0	91.0	91.0	91.0	91.0	91.0	85.8	85.8	85.8	85.8	85.8	85.8	77.5	77.5	77.5	77.5	77.5	77.5

Table 2: Accuracies after 100 inference iterations, for different RTM nanowire sizes and configurations. The fault rate is $p = 10^{-4}$.

overhead for ResNet, and only 6% for VGG. Note that while both *EVEN* and *EQUAL* protect the same number of layers, the overhead is different because the weight tensor sizes of the selected layers are varied, requiring different number of memory accesses, RTM shifts and misalignment. The *CUSTOM* configuration further reduces the accuracy drop: after 100 iterations, it is within 10% difference to the baseline, but at the cost of relatively higher performance overhead of 87% and 109% for ResNet and VGG respectively.

For RTM nanowire sizes 2 to 64, the accuracy after 100 inference iterations is reported in Table 2. Notably, for a nanowire size of two, the accuracy drop after 100 iterations is insignificant, especially in the VGG model. However, with increasing nanowire sizes, only the carefully protected configurations retain their accuracy and usability, while the accuracy in other configurations, such as *UNPROT* and *EVEN*, drops to unacceptable levels.

Analysis summary: NetDrift simulates the impact of RTM misalignment faults on BNNs accuracy and enables balancing the tradeoffs between the accuracy drop due to misalignment faults and the performance overhead associated with PECC schemes. The framework allows for conducting sensitivity analyses on a given BNN model to identify crucial layers that have a substantial impact on accuracy. Leveraging this information, selective application of error protection, such as in the *EQUAL* and *CUSTOM* configurations, becomes possible, enabling a collaborative reduction in both accuracy drop and performance overhead.

5 Conclusion

In this paper, we present NetDrift, a framework to simulate the effects of RTM misalignment faults on the accuracy of BNNs across multiple inference iterations. We evaluate various misalignment probabilities across different BNN models, exploring scenarios with no protection against misalignment faults, partial protection, and full protection. We show that NetDrift enables identifying critical network layers and exploring the trade-offs in accuracy and performance overhead. The framework also supports all necessary parameters to evaluate the impact on the energy consumption and to optimise for it. We also illustrate that despite the inherent resilience of BNNs, their accuracy suffers significantly in the absence of PECC for misalignment fault probabilities exceeding approximately 10^{-5}. In the future, we plan to extend NetDrift to high-precision networks for more comprehensive sensitivity analysis and simulate the fault model of Skymiron-based RTMs once it is available.

Acknowledgements

This work is partially funded by the German Research Council (DFG) through the CO4RTM (450944241), OneMemory (405422836), and ARTS-NVM (502308721).

References

1. Reuther et al., A.: Survey and benchmarking of machine learning accelerators. In: IEEE high performance extreme computing conference (HPEC). pp. 1–9 (2019)
2. Markidis et al., S.: Nvidia tensor core programmability, performance & precision. In: IEEE International Parallel and Distributed Processing Symposium Workshops. pp. 522–531 (2018)
3. Archer, S., Mappouras, G., Calderbank, R., Sorin, D.: Foosball coding: Correcting shift errors and bit flip errors in 3d racetrack memory. In: IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 331–342 (2020)
4. Bläsing, R., Khan, A.A., Filippou, P.C., Garg, C., Hameed, F., Castrillon, J., Parkin, S.S.P.: Magnetic racetrack memory: From physics to the cusp of applications within a decade. Proceedings of the IEEE **108**(8), 1303–1321 (2020)
5. Boukhobza, J., Rubini, S., Chen, R., Shao, Z.: Emerging nvm: A survey on architectural integration and research challenges. ACM Transactions on Design Automation of Electronic Systems (TODAES) **23**(2), 1–32 (2017)
6. Buschjäger, S., Chen, J.J., Chen, K.H., Günzel, M., Hakert, C., Morik, K., Novkin, R., Pfahler, L., Yayla, M.: Margin-maximization in binarized neural networks for optimizing bit error tolerance. In: Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 673–678 (2021)
7. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
8. Hirtzlin, T., Bocquet, M., Klein, J.O., Nowak, E., Vianello, E., Portal, J.M., Querlioz, D.: Outstanding bit error tolerance of resistive ram-based binarized neural networks. In: IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS). pp. 288–292 (2019)
9. Hochreiter, S.: The vanishing gradient problem during learning recurrent neural nets and problem solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **6**(02), 107–116 (1998)
10. Howard, J.: Imagenette. `https://github.com/fastai/imagenette/` (2019), [Online; accessed 25-April-2023]
11. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks. In: Advances in neural information processing systems. pp. 4107–4115 (2016)
12. Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al.: In-datacenter performance analysis of a tensor processing unit. In: International Symposium on Computer Architecture. pp. 1–12 (2017)
13. Kang, W., Wu, B., Chen, X., Zhu, D., Wang, Z., Zhang, X., Zhou, Y., Zhang, Y., Zhao, W.: A comparative cross-layer study on racetrack memories: Domain wall vs skyrmion. J. Emerg. Technol. Comput. Syst. **16**(1) (2019)
14. Khan, A.A., De Lima, J.P.C., Farzaneh, H., Castrillon, J.: The landscape of compute-near-memory and compute-in-memory: A research and commercial overview. arXiv preprint arXiv:2401.14428 (2024)

15. Khan, A.A., Ollivier, S., Hameed, F., Castrillon, J., Jones, A.K.: Downshift: Tuning shift reduction with reliability for racetrack memories. IEEE Transactions on Computers **72**(9), 2585–2599 (2023)
16. Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep. (2009)
17. Mappouras, G., Vahid, A., Calderbank, R., Sorin, D.J.: Greenflag: Protecting 3d-racetrack memory from shift errors. In: 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 1–12 (2019)
18. Multanen, J., Hepola, K., Khan, A.A., Castrillon, J., Jääskeläinen, P.: Energy-efficient instruction delivery in embedded systems with domain wall memory. IEEE Transactions on Computers **71**(9), 2010–2021 (2022)
19. Naffziger, S., Beck, N., Burd, T., Lepak, K., Loh, G.H., Subramony, M., White, S.: Pioneering chiplet technology and design for the amd epyc™ and ryzen™ processor families : Industrial product. In: ACM/IEEE International Symposium on Computer Architecture (ISCA). pp. 57–70 (2021)
20. Ollivier, S., Kline, D., Kawsher, R., Melhem, R., Banja, S., Jones, A.K.: Leveraging transverse reads to correct alignment faults in domain wall memories. In: International Conference on Dependable Systems and Networks. pp. 375–387 (2019)
21. Parkin, S., Hayashi, M., Thomas, L.: Magnetic Domain-Wall Racetrack Memory. Science **320**, 190–194 (2008)
22. Sari, E., Belbahri, M., Nia, V.P.: How does batch normalization help binary training? arXiv:1909.09139 (2019)
23. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (2015)
24. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms (2017)
25. Yayla, M., Thomann, S., Wei, M.L., Yang, C.L., Chen, J.J., Amrouch, H.: Hw/sw codesign for robust and efficient binarized snns by capacitor minimization. arXiv preprint arXiv:2309.02111 (2023)
26. Zhang, C., Sun, G., Zhang, X., Zhang, W., Zhao, W., Wang, T., Liang, Y., Liu, Y., Wang, Y., Shu, J.: Hi-fi playback: Tolerating position errors in shift operations of racetrack memory. In: 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA). pp. 694–706 (2015)