

ReSpawn: Energy-Efficient Fault-Tolerance for Spiking Neural Networks considering Unreliable Memories

Rachmad Vidya Wicaksana Putra*, Muhammad Abdullah Hanif†, Muhammad Shafique‡

*†*Technische Universität Wien (TU Wien), Vienna, Austria*

‡*New York University Abu Dhabi (NYUAD), Abu Dhabi, United Arab Emirates*

Email: {rachmad.putra, muhammad.hanif}@tuwien.ac.at, muhammad.shafique@nyu.edu

Abstract—Spiking neural networks (SNNs) have shown a potential for having low energy with unsupervised learning capabilities due to their biologically-inspired computation. However, they may suffer from accuracy degradation if their processing is performed under the presence of hardware-induced faults in memories, which can come from manufacturing defects or voltage-induced approximation errors. Since recent works still focus on the fault-modeling and random fault injection in SNNs, the impact of memory faults in SNN hardware architectures on accuracy and the respective fault-mitigation techniques are not thoroughly explored. Toward this, we propose ReSpawn, a novel framework for mitigating the negative impacts of faults in both the off-chip and on-chip memories for resilient and energy-efficient SNNs. The key mechanisms of ReSpawn are: (1) analyzing the fault tolerance of SNNs; and (2) improving the SNN fault tolerance through (a) fault-aware mapping (FAM) in memories, and (b) fault-aware training-and-mapping (FATM). If the training dataset is not fully available, FAM is employed through efficient bit-shuffling techniques that place the significant bits on the non-faulty memory cells and the insignificant bits on the faulty ones, while minimizing the memory access energy. Meanwhile, if the training dataset is fully available, FATM is employed by considering the faulty memory cells in the data mapping and training processes. The experimental results show that, compared to the baseline SNN without fault-mitigation techniques, ReSpawn with a fault-aware mapping scheme improves the accuracy by up to 70% for a network with 900 neurons without retraining.

Index Terms—Spiking neural networks, energy efficiency, fault tolerance, memory faults, approximation errors, manufacturing defects, fault-aware mapping, fault-aware training.

I. INTRODUCTION

Spiking neural networks (SNNs) have shown promising performance by achieving high accuracy with low energy consumption in an unsupervised learning scenario [1]–[4]. Currently, a large-sized SNN model is more favorable than the smaller ones since it can achieve higher accuracy, but at the cost of higher memory footprint and energy consumption [3], as illustrated in Fig. 1. To address these challenges, SNN accelerators have been developed to improve the performance and energy-efficiency of SNN-based applications [5]–[10]. However, these SNN accelerators may suffer from accuracy degradation when SNN processing is performed under the presence of *hardware-induced faults in memories*, which can come from the following sources.

- **Manufacturing defects:** The imperfections in the chip fabrication process can cause defects in memory cells, which

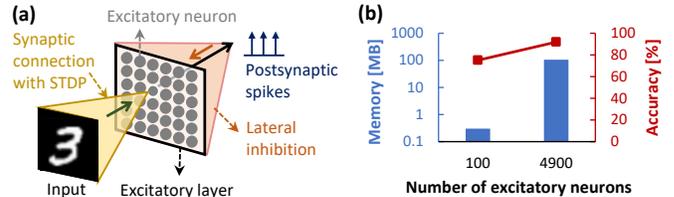


Fig. 1. (a) SNN architecture considered in this work, i.e., a single-layer fully-connected network. (b) An SNN model with a larger model size typically achieves higher accuracy than the smaller ones.

degrades the cell functionality in the form of faults, and thereby reducing the yield of chips [11].

- **Voltage-induced approximation errors:** The operational voltage of memories can be reduced to decrease the power and energy, at the cost of increased fault rates [12] [13].

Targeted Research Problem: *If and how can we mitigate the negative impacts of faults in the off-chip and on-chip weight memories on the accuracy, thereby improving the fault tolerance of SNNs and maintaining high accuracy.*

The efficient solution to this problem will enable reliable SNN processing in the presence of unreliable memories for energy-constrained devices, such as IoT-Edge. Furthermore, this will also improve the yield and reduce the per-unit-cost of the SNN accelerator.

A. State-of-the-Art and Their Limitations

Standard fault-tolerance techniques for VLSI circuits (such as DMR [14], TMR [15], and ECCs [16])¹ are not effective for improving the resiliency of SNN systems, as they need redundant hardware and/or execution which incurs high overhead. To address this, state-of-the-art works have proposed fault tolerance for SNNs, which can be categorized as follows.

- **Fault modeling of SNNs:** It discusses (1) a set of possible faults that can affect SNN components, such as neurons and synapses [17], and (2) the fault modeling for analog neuron hardware [18] as well as its fault-tolerance strategy [19].
- **Studying the impacts of faults on accuracy:** It explores and discusses the impacts of bit-flips in weights [20] and full synapse failure (i.e., synapse removal) [21] [22] on the accuracy, considering different fault rates with random distribution.

¹DMR: Dual Modular Redundancy, TMR: Triple Modular Redundancy, and ECCs: Error Correcting Codes.

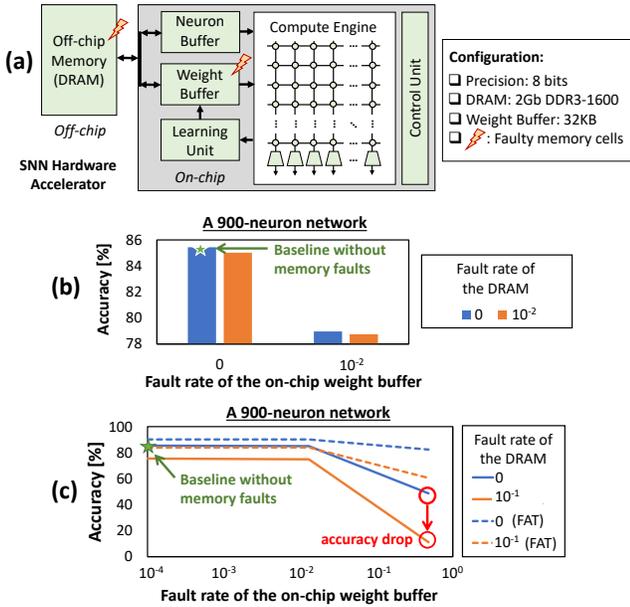


Fig. 2. (a) The SNN accelerator considered in this work with faults in the DRAM and the weight buffer. We employ 8-bit precision, a 2Gb DDR3-1600 DRAM, a 32KB weight buffer, and uniform random distribution of faults on each bank of the DRAM and the weight buffer in the form of bit-flip. (b) Significance of faults in the DRAM and the weight buffer on the accuracy. (c) Increasing fault rates in weight memories lead to accuracy degradation, and fault-aware training (FAT) can improve the SNN fault tolerance.

• **Retraining-based fault mitigation:** Work [22] incorporates additional components (i.e., astrocyte units) to the network for enhancing the retraining process for mitigating the faults.

Limitations: These state-of-the-art still focus on SNN fault modeling and fault injection without considering the SNN hardware architectures. Therefore, *the impact of bit-level faults in the off-chip and the on-chip memories on the accuracy, and the respective fault-mitigation techniques, are still unexplored.* Moreover, *current fault mitigation techniques still rely on the costly additional components and retraining process.*

To understand the characteristics of the SNN fault tolerance, we present an experimental case study in the following section.

B. Motivational Case Study and Research Challenges

To understand the impact of faults (from the manufacturing defects and the reduced-voltage operations in memories) on the accuracy, we perform experiments that explore different fault rates in the DRAM-based off-chip memory and the SRAM-based on-chip weight memory (weight buffer), while considering the typical architecture of SNN accelerators, as shown in Fig. 2(a). Further details on the experimental setup are presented in Section IV.

From the experimental results in Fig. 2(b)-(c), we make the following key observations.

- Different fault rates in the DRAM and the weight buffer cause an SNN system to obtain different accuracy scores.
- Faults in the weight buffer have a relatively higher impact on the accuracy degradation than the DRAM, since its size is significantly smaller than DRAM, and thereby having a higher probability to affect more weights.

- Fault-aware training (FAT) with progressive fault injection for neural networks [23] [24] can improve the SNN fault tolerance while incurring high training time and energy, as it considers a wide range of fault rates for the injection.

Research Challenges: The above observations expose key challenges that need to be solved for addressing the targeted research problem, as discussed in the following.

- *The fault-mitigation technique should minimize the impacts of faults in both, the DRAM and the weight buffer, thereby improving the SNN fault tolerance.*
- *It should employ a technique that does not rely on retraining, as retraining needs a full training dataset, which may not be available due to restriction policies (e.g., a company releases an SNN model, but makes the training dataset unavailable).*
- *It should incur low energy overhead at run-time, compared to the baseline (without fault-mitigation technique) to enable energy-efficient SNN applications.*

C. Our Novel Contributions

To address the above challenges, we propose **ReSpawn framework** that enables *energy-efficient fault-tolerance for Spiking neural networks considering unreliable memories.* To the best of our knowledge, this work is the first effort that mitigates the negative impacts of faults in the off-chip and on-chip weight memories of SNN accelerators. Following are the key steps of the ReSpawn framework (see Fig. 3):

- 1) **Analyzing the Fault Tolerance of the SNN Model** to characterize the accuracy values under the given fault rates. It is performed by adjusting the fault rates in the memories, while checking the obtained accuracy.
- 2) **Improving the Fault Tolerance of the SNN model** whose strategies depend on the availability of the training dataset.
 - *If the training dataset is not fully available, the **Fault-aware Mapping (FAM)** is employed through simple bit-shuffling techniques, that prioritize placing the bits with higher significance on the non-faulty memory cells.*
 - *If the training dataset is fully available, the **Fault-aware Training-and-Mapping (FATM)** is employed by including the information of the faulty memory cells in the data mapping and training processes. Here, the data mapping strategy follows the proposed FAM technique.*

Key Results: We evaluated our ReSpawn framework for accuracy and energy, using Python-based simulations on the GPGPU. The experimental results show that ReSpawn with fault-aware mapping improves the accuracy by up to 70% for a 900-neuron network without retraining on the MNIST dataset.

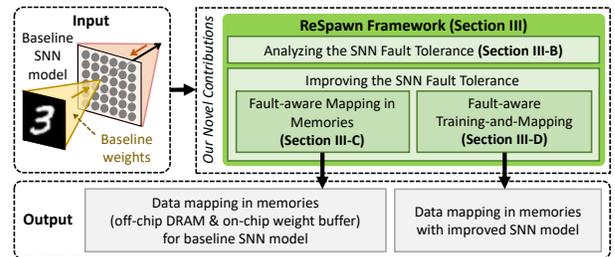


Fig. 3. An overview of our novel contributions (shown in the green boxes).

II. PRELIMINARIES

A. Spiking Neural Networks (SNNs)

The components of an SNN model consist of spike coding, neuron model, network architecture, and learning rule. Spike coding represents the input into spikes. Different types of spike coding have been proposed, such as rate, rank-order, phase, burst, and time-to-first spike [25]–[28], and here we select the rate coding due to its robustness for data representation. For the neuron model, we choose the Leaky Integrate-and-Fire (LIF) as it has low complexity [29]. Its membrane potential increases if a presynaptic spike comes, and it decreases if no presynaptic spike is observed. If the potential reaches the threshold (V_{th}), a postsynaptic spike is generated, and afterward, it is back to the reset potential (V_{reset}). We consider the network architecture of Fig. 1(a), i.e., a single-layer fully-connected network, since it shows the state-of-the-art accuracy for the unsupervised learning scenario [3]. In such an architecture, the spikes generated from each neuron are passed to other neurons for providing inhibition, which promotes competition among neurons to recognize the given input. We employ the spike-timing-dependent plasticity (STDP) for the learning rule, due to its capability for learning input features under unsupervised settings. We consider the unsupervised SNNs since they can learn features from the unlabeled data, which is highly desired for real-world applications (gathering unlabeled data is easier and cheaper than labeled ones) [3].

B. Fault Modeling for Memories

We focus on the fault modeling for the DRAM and the SRAM weight buffer, since we aim at accurately exploring the impacts of hardware-induced faults in weight memories across the hierarchy of an SNN accelerator, as shown in Fig. 2(a). These faults can come from *manufacturing defects*, due to the imperfections in the fabrication process [11] [30]–[33], and *reduced-voltage operation*, which is performed for decreasing the operational power/energy [12] [13].

Faults from Manufacturing Defects: The SNN hardware accelerators are fabricated using a sophisticated manufacturing process. Hence, there is a chance of imperfections that result in defects in the fabricated chips. Moreover, the technology scaling, which is employed for improving the performance and efficiency of the chips, may increase fault rates related to both, permanent faults (e.g., stuck-at faults) and transient faults (e.g., bit-flip) at random locations of a chip. Therefore, the faults from manufacturing defects can be modeled using a uniform random distribution, which has also been considered in previous works [34] [35].

Faults from Reduced-Voltage Operations: The reduction of operational voltage is a widely-used approach to reduce the operational power/energy of DRAM and SRAM-based buffer, at the cost of increased fault rates, as shown in Fig. 4. For DRAM, we follow the fault model from [23], i.e., the faults are modeled by considering the *weak cells* (i.e., cells that fail when the DRAM voltage is reduced), and the probability of a fault in any weak cell. Here, the faults have a uniform random

distribution across a DRAM bank. Meanwhile, for SRAM, we follow the fault model from [13], i.e., the faults have a uniform random distribution across an SRAM bank. The selection of the uniform random distribution as the fault model for DRAM and SRAM, is motivated by the following reasons: (1) it produces faults with high similarity to the real reduced-voltage DRAM [23] and the real reduced-voltage SRAM [13]; and (2) it offers fast software-based fault injection.

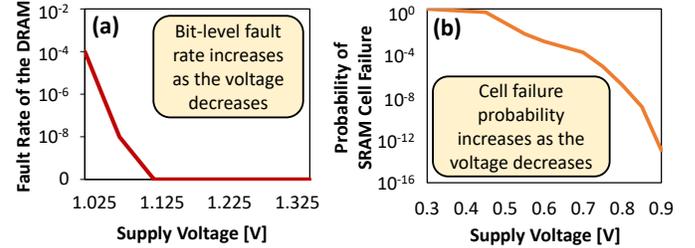


Fig. 4. (a) DRAM fault rates and the corresponding DRAM voltage values, based on the study of [12]. (b) SRAM cell failure probability (P_{cell}) and the corresponding SRAM voltage values for a 28nm CMOS technology, based on the study of [13]. The yield of non-faulty cells is defined as $Y = (1 - P_{cell})^M$ with M denotes the total memory bit-cells.

III. RESPAWN FRAMEWORK

A. Overview

We propose the ReSpawn framework for energy-efficient fault-tolerance for SNNs processing on unreliable off-chip and on-chip weight memories. The key steps of our ReSpawn are shown in Fig. 5, and discussed in the following sections.

- 1) **Analyzing the SNN Fault Tolerance (Section III-B):** It aims at understanding the interaction between the fault rates and the accuracy, by exploring different combinations of fault rates in DRAM and weight buffer, while observing the accuracy scores. This information is then leveraged for improving the SNN fault tolerance.
- 2) **Improving the SNN Fault Tolerance** through different strategies, depending on the availability of training dataset.
 - **Fault-aware Mapping (Section III-C):** *This strategy is performed if the training dataset is not fully available.* It employs efficient bit-shuffling techniques, that place the significant bits on the non-faulty memory cells and the insignificant bits on the faulty ones. We propose two FAM techniques to offer accuracy-energy trade-offs.
 - **FAM1:** It considers the fault map from each memory as an individual fault map, and devises a mapping pattern for each fault map.
 - **FAM2:** It merges multiple fault maps from different memories to an integrated fault map, and devises a mapping pattern for it accordingly.
 - **Fault-aware Training-and-Mapping (Section III-D):** *This strategy is performed if the training dataset is fully available.* It uses the information of the faulty memory cells in the data mapping and training processes. Here, the data mapping strategy also follows the proposed FAM techniques (i.e., FAM1 and FAM2).

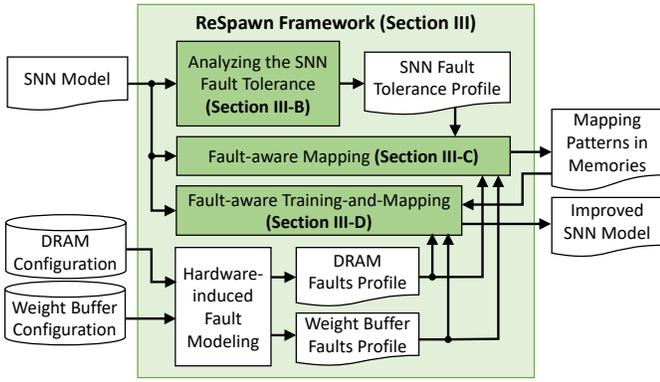


Fig. 5. An overview of our ReSpawn framework, whose novel mechanisms are shown in the green boxes.

B. SNN Fault Tolerance Analysis

Understanding the fault tolerance of the given SNN model is important, because the information from the analysis will be beneficial, especially for performing efficient fault-mitigation techniques. Therefore, our ReSpawn framework analyzes the fault tolerance of the SNN model to observe the interaction between memory faults and accuracy. It is performed by exploring different combinations of fault rates in the DRAM and the weight buffer, while observing the obtained accuracy. For instance, if we consider a network with 900 neurons, our ReSpawn will explore different combinations of fault rates in DRAM and weight buffer, and the experimental results are shown in Fig. 6. These results show two different regions, i.e., where fault rates in memories cause the network to achieve acceptable accuracy, as shown by label-**(A)**, and where fault rates in memories cause the network to suffer from notable accuracy degradation, as shown by label-**(B)**. These regions provide insights regarding the tolerable fault rates that should be considered to effectively improve the SNN fault tolerance.

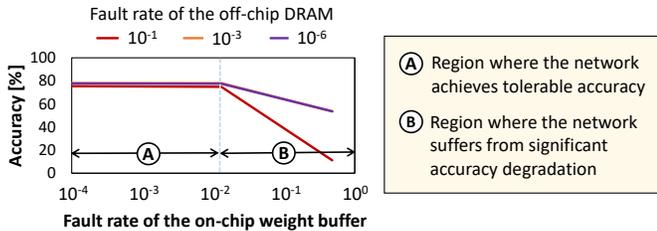


Fig. 6. The experimental results for a 900-neuron network, considering different fault rates for DRAM and weight buffer.

C. Fault-aware Mapping (FAM)

Faulty cells in memories that come from manufacturing defects and reduced-voltage operations, can be characterized at design time. Therefore, their locations are known before the deployment. ReSpawn framework leverages the information of faulty cells in DRAM and weight buffer to effectively map the weights on memory fabrics, thereby minimizing the impact of faulty cells on the significant bits. It is performed through fault-aware mapping (FAM), that employs simple bit-shuffling techniques for placing the significant bits on the non-faulty memory cells and the insignificant bits on the faulty ones.

Furthermore, we observe that, a data word may have a single faulty bit or multiple faulty bits, depending on whether this word occupies a memory segment that has a single faulty cell or multiple faulty cells, as illustrated in Fig. 7(a). Therefore, we propose a mapping strategy that can address both, the single fault-per-word and multiple faults-per-word scenarios.

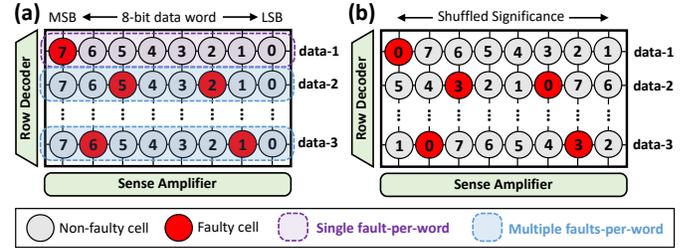


Fig. 7. (a) Illustration of possible locations of faulty cells in memories. (b) The proposed bit-shuffling technique, which is based on the right circular-shift.

The proposed memory mapping strategy is performed by the following steps, which are also illustrated in Fig. 7(b).

- **Step-1: Identifying the faulty cells in the given memories.** This step aims at obtaining the information of faulty cells in each memory, such as fault rate and fault map. The faulty cells in the on-chip buffer from manufacturing defects can be detected using the standard post-fabrication testing [34], and the faulty cells in the DRAM can be detected through measurements, e.g., using SoftMC tool [36]. Meanwhile, the faulty cells from reduced-voltage operations can also be detected through measurements on the DRAM [36] and on the on-chip buffer [13]. In this manner, collecting the faulty cell information is feasible as it follows the standard post-fabrication testing and measurements.
- **Step-2: Identifying the maximum fault rate allowed in a data word.** This step aims at determining which memory cells, that can be used for storing a data word, by considering the interaction between fault rates and accuracy from SNN fault tolerance analysis in Section III-B. For instance, we allow a maximum of 2 faulty bits for an 8-bit data word.
- **Step-3: Identifying the memory segment with the highest number of subsequent non-faulty cells for storing a data word.** It aims at maximizing the possibility of placing the significant bits on the non-faulty cells. Therefore, we also examine the corner case (i.e., the left-most memory cell with the right-most memory cell) as possible subsequent non-faulty cells, as shown in the second row of Fig. 7(b).
- **Step-4: Performing circular-shift technique for each data word.** It aims at efficiently performing bit-shuffling. Here, we always employ right circular-shift to simplify the control.

Since the FAM technique leverages the information of fault maps from multiple memories, we propose two variants of FAM techniques to offer different accuracy-energy trade-offs, which are discussed in the following.

1) **FAM for Individual Fault Map (FAM1):** This technique considers an individual fault map from each memory (i.e., DRAM or weight buffer). Therefore, the FAM1 devises multiple mapping patterns, i.e., one pattern for DRAM, and

another one for weight buffer, as illustrated in Fig. 8. This FAM1 technique offers high resiliency against faults from each memory, as each mapping pattern minimizes the negative effect of faults on the significant bits. However, it needs to perform a specialized data mapping for each memory.

2) *FAM for Integrated Fault Map (FAM2)*: This technique merges multiple fault maps from multiple memories (i.e., DRAM and weight buffer) as an integrated fault map. Hence, the FAM2 only devises a single mapping pattern for both, DRAM and weight buffer, as illustrated in Fig. 9. This FAM2 technique potentially offers better efficiency than the FAM1, due to its simpler mapping technique. However, it has less resiliency than the FAM1 as the generated mapping pattern may be sub-optimal for each memory, because some insignificant bits may be placed in non-faulty cells and some significant bits in faulty ones.

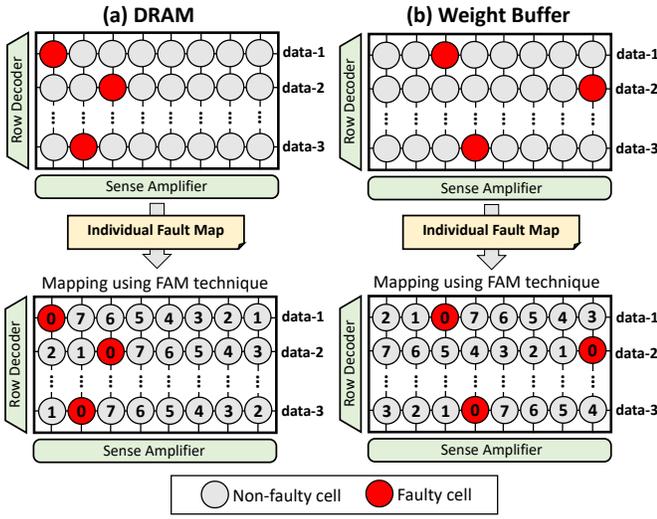


Fig. 8. (a) For DRAM, the FAM1 only considers the DRAMs' fault map. (b) For weight buffer, the FAM1 only considers the buffers' fault map.

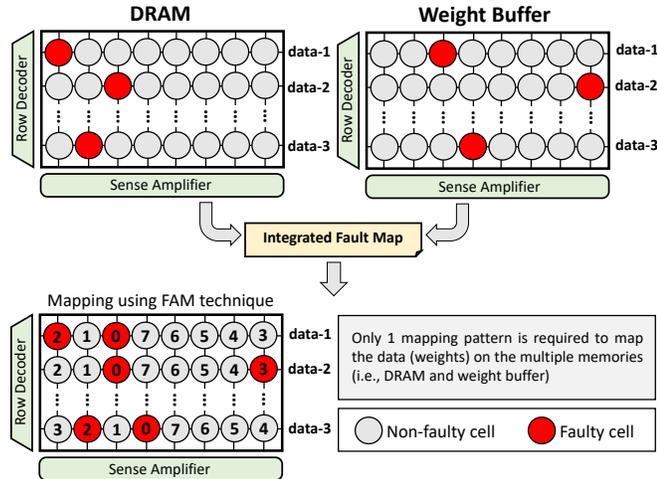


Fig. 9. The FAM2 technique considers the integrated fault map for devising the mapping pattern for both, DRAM and weight buffer.

ReSpawn also considers optimizing the energy of DRAM and SRAM buffer accesses to maximize the energy efficiency potential, since memory accesses typically dominate the total

energy of SNN processing [37]. The DRAM mapping is performed by maximizing the DRAM row buffer hits [38], multi-bank burst feature [39], and subarray-level parallelism [40] [41], while considering the proposed FAM techniques (the algorithm is presented in Alg. 1). Meanwhile, the SRAM buffer mapping is performed by maximizing the bank-level parallelism [39] while considering the proposed FAM techniques (the algorithm is presented in Alg. 2).

Algorithm 1 The proposed mapping for a DRAM chip

INPUT: (1) DRAM ($DRAM$), number of bank-per-chip (D_{ba}), number of subarray-per-bank (D_{su}), number of row-per-subarray (D_{ro}), number of column-per-row (D_{co});
(2) Fault rate of a DRAM column (D_{rate_col}), maximum tolerable fault rate for a DRAM column ($D_{rate_col_{max}}$);
(3) Weight bits ($weight_b$);
(4) Fault-aware mapping (FAM); // either FAM1 or FAM2
OUTPUT: DRAM ($DRAM$);

BEGIN

Process:
1: **for** $ro = 0$ to $(D_{ro} - 1)$ **do**
2: **for** $su = 0$ to $(D_{su} - 1)$ **do**
3: **for** $ba = 0$ to $(D_{ba} - 1)$ **do**
4: **for** $co = 0$ to $(D_{co} - 1)$ **do**
5: **if** $D_{rate_col} \leq D_{rate_col_{max}}$ **then**
6: $DRAM[ba, su, ro, co] \leftarrow FAM(weight_b)$;
7: **end if**
8: **end for**
9: **end for**
10: **end for**
11: **end for**
12: **return** $DRAM$;
END

Algorithm 2 The proposed mapping for SRAM buffer

INPUT: (1) SRAM ($SRAM$), number of bank (S_{ba}), number of row-per-bank (S_{ro}); // the number of column-per-row = the bitwidth of a word
(2) Fault rate of an SRAM row (S_{rate_row}), maximum tolerable fault rate for an SRAM row ($S_{rate_row_{max}}$);
(3) Weight bits ($weight_b$);
(4) Fault-aware mapping (FAM); // either FAM1 or FAM2
OUTPUT: SRAM ($SRAM$);

BEGIN

Process:
1: **for** $ro = 0$ to $(S_{ro} - 1)$ **do**
2: **for** $ba = 0$ to $(S_{ba} - 1)$ **do**
3: **if** $S_{rate_row} \leq S_{rate_row_{max}}$ **then**
4: $SRAM[ba, ro] \leftarrow FAM(weight_b)$;
5: **end if**
6: **end for**
7: **end for**
8: **return** $SRAM$;
END

Note: The proposed FAM techniques (FAM1 and FAM2) do not require retraining, thereby making them suitable for energy-efficient and fault-tolerant SNN processing, especially in the case where the training dataset is not fully available. Consequently, these techniques can also improve the yield and reduce the per-unit-cost of SNN accelerators (hardware chips).

D. Fault-aware Training-and-Mapping (FATM)

If the training dataset is fully available, users can decide if they want to perform fault-mitigation techniques without training, like our FAM techniques (Section III-C),

or fault-aware training (FAT)². Our experimental results in Fig. 2(c) show that, the FAT technique can improve the SNN fault tolerance. Toward this, *ReSpawn framework also provides FAT-based solutions to improve the SNN fault tolerance on top of the proposed FAM techniques; so-called fault-aware training-and-mapping (FATM)*. For conciseness, the FAT with FAM1 mapping is referred to as the FATM1, and the FAT with FAM2 mapping is referred to as the FATM2. The proposed FATM is performed through the following mechanisms.

- 1) We employ the FAM technique (FAM1 or FAM2) on the given SNN model, to minimize the negative impacts of faults on the weights. This results in the SNN model whose weights have been minimally affected by the faults (i.e., the FAM-improved SNN model).
- 2) Afterward, we perform training to the FAM-improved SNN model through the following steps.
 - **Step-1:** The faults are generated for different rates, based on the SNN fault tolerance analysis in Section III-B.
 - **Step-2:** The generated faults are injected into locations in DRAM and weight buffer, thereby causing the weight bits stored in these locations to flip.
 - **Step-3:** We train the SNN model while considering fault rates that do not cause accuracy drop (fault rates from region-**A**) in Fig. 6) and are close to region-**B**). It makes the model adapting to high fault rates safely, without causing accuracy to decrease, and with less training time, since smaller fault rates are not considered.
 - **Step-4:** Afterward, we carefully train the SNN model while considering fault rates that cause notable accuracy drop, i.e., fault rates from region-**B**), by incrementally increasing the fault rates of DRAM and weight buffer, after each training epoch.
 - **Step-5:** Training is terminated when the network faces accuracy saturation or degradation. The final SNN model is selected from the trained model that is saved in the previous training epoch.

The proposed FAM (FAM1 and FAM2) and FATM (FATM1 and FATM2) techniques are applicable for different memory technologies (like CMOS, RRAM, etc.) since they consider bit-level fault mitigation, which is suitable for bit-level data storage in each memory cell. Therefore, the possible extension to the ReSpawn framework is by considering the multi-level cell characteristics into its optimization process.

IV. EVALUATION METHODOLOGY

The experimental setup for evaluating ReSpawn framework is illustrated in Fig. 10. Following is the detailed information of the evaluation methodology, comprising the scenarios for experiments and comparisons.

For the network architecture, we use the single-layer fully-connected SNN, like the network in Fig. 1(a), with a different number of neurons (i.e., 100, 400, 900, 1600, 2500, and

²Fault-aware training (FAT) is a widely used technique for improving the fault-tolerance of neural networks, by incorporating the information of faults in the training process [23] [24] [34].

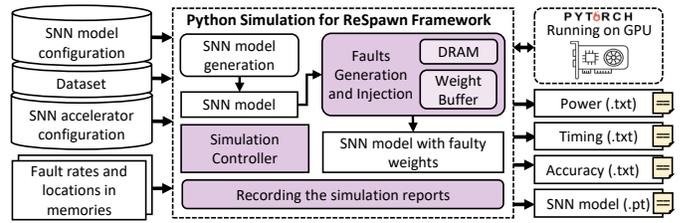


Fig. 10. Experimental setup and tool flow.

3600) to show the generality of the ReSpawn, which we refer them to as Net100, Net400, Net900, Net1600, Net2500, and Net3600, respectively. We consider this network as it provides robustness when performing different variants learning rules [42], thereby it is representative for evaluation. Meanwhile, for the comparison partners, we consider two designs: (1) the baseline SNN model without any fault-mitigation technique, and (2) the SNN model with the FAT technique. We compare these designs against our ReSpawn techniques (i.e., FAM1, FAM2, FATM1, and FATM2) on the MNIST dataset.

Fault Injection: We generate memory faults based on the fault modeling described in Section II-B. Afterward, we inject these faults into the locations in DRAM and weight buffer to represent the faulty memory cells, and the data bits in these cells are flipped. For ReSpawn, we employ mapping policies from Alg. 1 and Alg. 2, while for the baseline, we store the weights in subsequent addresses in a DRAM bank.

Accuracy Evaluation: To evaluate the accuracy of SNNs, we use Python-based simulations [43] that run on GPGPU, i.e., Nvidia RTX 2080 Ti, while considering an SNN accelerator architecture that follows the design of [9] with 8-bit precision of weights, a DDR3-1600 2Gb DRAM, and a 32KB weight buffer, as shown in Fig. 2(a). We use 8-bit precision as it has a sufficient range of values to represent the SNN weights [3].

Energy Evaluation: We consider the approach of [44] for estimating the SNN processing energy of an SNN model, i.e., by leveraging the information of processing power that is obtained through *nvidia-smi* utility, and its processing time. We perform the energy evaluation for different scenarios, i.e., the fault-mitigation techniques without retraining (i.e., FAM1 and FAM2) and with retraining (i.e., FAT, FATM1, and FATM2).

V. RESULTS AND DISCUSSIONS

A. Maintaining the Accuracy

Fig. 11 presents the experimental results on the accuracy of different fault-mitigation techniques, i.e., the baseline, the FAT, our FAM techniques (FAM1 and FAM2), and our FATM techniques (FATM1 and FATM2).

We observe that the baseline is susceptible to accuracy degradation when the SNN is run under the presence of faults, as these faults alter the weights and affect the output of the SNN model. The accuracy degradation is more evident in the scenarios where high fault rates are observed, as shown by label-**1**. The FAT technique improves the SNN fault tolerance compared to the baseline across all evaluation scenarios, as the FAT-improved SNN model has a better capability for adapting to the presence of faults, as shown by label-**2**. However,

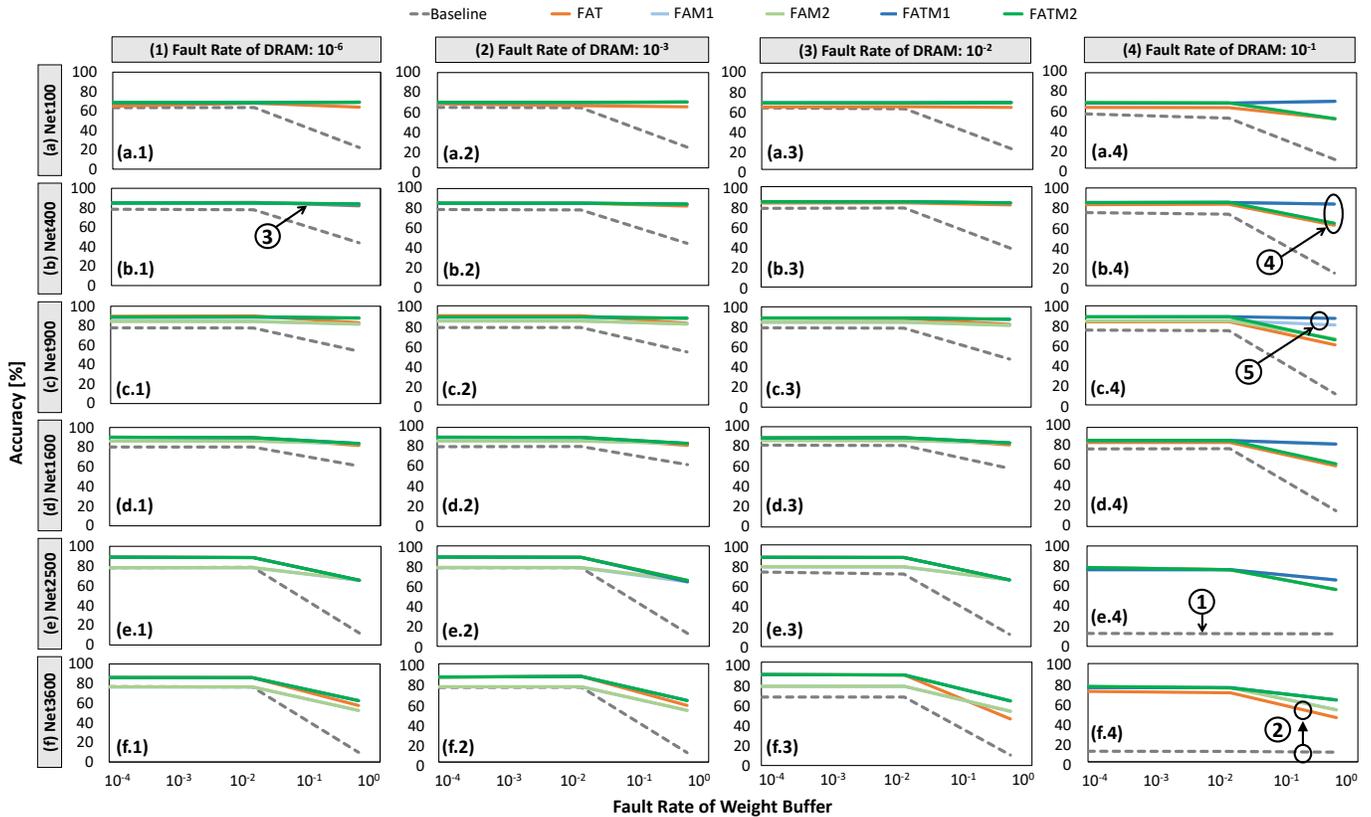


Fig. 11. Accuracy achieved by different techniques, i.e., the baseline, the fault-aware training (FAT), our fault-aware mapping (FAM1 and FAM2), and our fault-aware training-and-mapping (FATM1 and FATM2), across different sizes of network, i.e., (a) Net100, (b) Net400, (c) Net900, (d) Net1600, (e) Net2500, and (f) Net3600, as well as different fault rates in DRAM and weight buffer.

the FAT technique may offer limited accuracy improvements, since it does not substantially eliminate the negative impact of faults on the significant bits of weights, and its performance depends on the effectiveness of the training strategy. On the other hand, our FAM techniques (FAM1 and FAM2) can achieve comparable accuracy compared to the FAT without retraining, as shown by label-③. They improve the accuracy by up to 61%, 70%, 70%, 67%, 53%, and 43% for Net100, Net400, Net900, Net1600, Net2500, and Net3600 respectively, compared to the baseline. The reason is that, the main idea of our FAM1 and FAM2 is to eliminate the impact of faults on the significant bits of weights through simple bit-shuffling, thereby maintaining the value of weights as close as possible to the weights that are trained in an ideal condition, i.e., environment without faults. These results show that *our FAM techniques (FAM1 and FAM2) are effective for fault-mitigation techniques in SNNs, especially in the case where the training dataset is not fully available*. Moreover, these techniques can enhance the yield, thereby decreasing the per-unit-cost of SNN chips.

We also observe that the FAM1 has better performance than the FAM2, as it consistently obtains high accuracy across all evaluation scenarios, while the performance of the FAM2 may still be affected by cases that have high fault rates, as shown by label-④. The reason is that, the FAM1 minimizes the impact of faults on the significant bits of weights from each memory. Meanwhile, the FAM2 minimizes the impact of faults on the significant bits of weights considering the integrated fault map

from multiple memories, which may be sub-optimal for each memory. Furthermore, we observe that our FATM techniques can further improve the SNN fault tolerance from the FAM techniques (shown in label-⑤), since the training is performed to the model whose weights are already minimally affected by the faults. The FATM techniques improve the accuracy by up to 61%, 70%, 76%, 67%, 53%, and 53% for Net100, Net400, Net900, Net1600, Net2500, and Net3600 respectively, compared to the baseline. These results show that, *our FATM techniques (FATM1 and FATM2) can further improve the SNN fault tolerance if the training dataset is fully available*.

B. Reducing the Energy Consumption

Fig. 12 presents the experimental results on the energy consumption of different fault-mitigation techniques, i.e., the baseline, the FAT, our FAM1, FAM2, FATM1 and FATM2. For the training-based solutions (FAT, FATM1, and FATM2), the energy consumption is evaluated when performing one training epoch over 60k samples from the full MNIST training set. For the solutions without training (FAM1 and FAM2), the energy consumption is evaluated when performing a test over the 10k samples from the full MNIST test set. *This evaluation scenario aims at showing how much energy the training-based solutions incur, compared to the solutions without training*.

We observe that, the FAT technique consumes high energy across different network sizes, since it requires the training process. Moreover, a larger-sized network incurs higher power

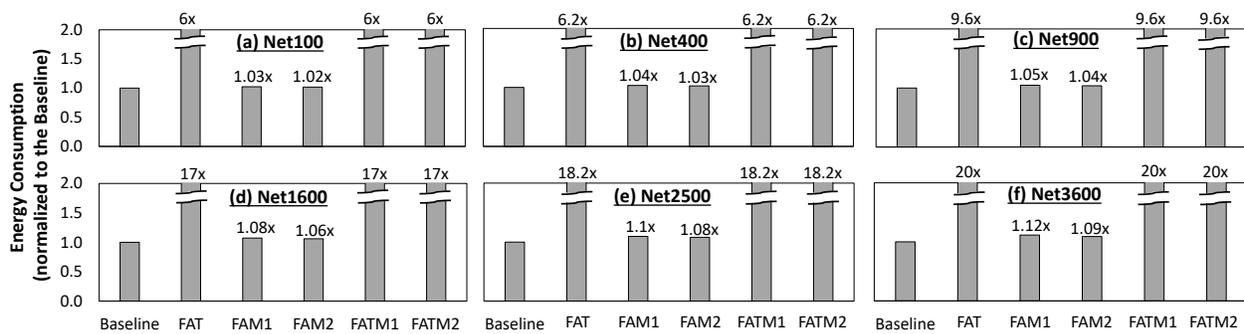


Fig. 12. Normalized energy consumption of different fault-mitigation techniques (i.e., the baseline, the FAT, our FAM1 and FAM2, as well as our FATM1 and FATM2) across different sizes of network: (a) Net100, (b) Net400, (c) Net900, (d) Net1600, (e) Net2500, and (f) Net3600.

and processing time, and thereby higher energy. If we consider one training epoch (i.e., running 60k samples from the full MNIST training set), the FAT incurs about 6x-20x energy for Net100-Net3600, compared to the baseline. This condition can be exacerbated by the fact that the energy consumption is increased if the FAT requires multiple training epochs. Here, the FATM1 and FATM2 techniques face the same issues due to the training-based approach. On the other hand, our FAM1 technique only incurs 1.03x-1.12x energy, and our FAM2 technique only incurs 1.02x-1.09x energy for Net100-Net3600 compared to the baseline, when running 10k samples from a full MNIST test set. Moreover, the energy efficiency of the FAM1 and the FAM2 can be better if the number of samples in the inference phase is higher. The reason is that, the mapping patterns in the FAM1 and the FAM2 need to be generated only once, before running the inference, therefore the energy overhead is negligible considering the huge number of samples to be processed in the inference phase. We also observe that the FAM2 incurs slightly less energy compared to the FAM1, as the FAM2 only considers one integrated fault map for its mapping operations while the FAM1 considers multiple fault maps for its mapping operations, thereby incurring fewer operations and processing energy. These results show that, *our FAM techniques (FAM1 and FAM2) have high potential as the energy-efficient fault-mitigation techniques for SNNs*, as they maintain high accuracy with minimum energy overhead.

VI. CONCLUSION

We propose ReSpawn framework for mitigating the faults in the off-chip and on-chip weight memories for SNN-based systems. Our ReSpawn employs SNN fault tolerance analysis, fault-aware mapping, and fault-aware mapping-and-training. The experimental results show that, ReSpawn with fault-aware mapping improves the accuracy by up to 70% for a 900-neuron network without retraining. Therefore, our work enhances the SNN fault tolerance with minimum energy overhead, thereby potentially improving the yield of SNN hardware chips.

VII. ACKNOWLEDGMENT

This work was partly supported by Intel Corporation through Gift funding for the project "Cost-Effective Dependability for Deep Neural Networks and Spiking Neural Networks", and by Indonesia Endowment Fund for Education (LPDP) through Graduate Scholarship Program.

REFERENCES

- [1] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in Neuroscience*, vol. 12, 2018.
- [2] A. Tavanaei *et al.*, "Deep learning in spiking neural networks," *Neural Networks*, vol. 111, pp. 47–63, 2019.
- [3] R. V. W. Putra and M. Shafique, "Fspinn: An optimization framework for memory-efficient and energy-efficient spiking neural networks," *IEEE TCAD*, vol. 39, no. 11, pp. 3601–3613, 2020.
- [4] R. V. W. Putra and M. Shafique, "Spikedyn: A framework for energy-efficient spiking neural networks with continual and unsupervised learning capabilities in dynamic environments," *arXiv preprint arXiv:2103.00424*, 2021.
- [5] F. Akopyan *et al.*, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE TCAD*, vol. 34, no. 10, pp. 1537–1557, 2015.
- [6] A. Roy *et al.*, "A programmable event-driven architecture for evaluating spiking neural networks," in *Proc. of ISLPED*, July 2017, pp. 1–6.
- [7] S. Sen *et al.*, "Approximate computing for spiking neural networks," in *Proc. of DATE*, March 2017, pp. 193–198.
- [8] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan 2018.
- [9] C. Frenkel *et al.*, "A 0.086-mm² 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm cmos," *IEEE TBCAS*, vol. 13, no. 1, pp. 145–158, Feb 2019.
- [10] C. Frenkel *et al.*, "Morphic: A 65-nm 738k-synapse/mm² quad-core binary-weight digital neuromorphic processor with stochastic spike-driven online learning," *IEEE TBCAS*, vol. 13, no. 5, 2019.
- [11] I. Koren and Z. Koren, "Defect tolerance in vlsi circuits: techniques and yield analysis," *Proc. of the IEEE*, vol. 86, no. 9, pp. 1819–1838, 1998.
- [12] K. K. Chang *et al.*, "Understanding reduced-voltage operation in modern dram devices: Experimental characterization, analysis, and mechanisms," *ACM POMACS*, vol. 1, no. 1, Jun. 2017.
- [13] S. Ganapathy *et al.*, "Mitigating the impact of faults in unreliable memories for error-resilient applications," in *Proc. of DAC*, 2015.
- [14] R. Vadlamani *et al.*, "Multicore soft error rate stabilization using adaptive dual modular redundancy," in *Proc. of DATE*, 2010, pp. 27–32.
- [15] R. E. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM J. Res. Dev.*, vol. 6, no. 2, 1962.
- [16] H. Y. Sze, "Circuit and method for rapid checking of error correction codes using cyclic redundancy check," Jul. 18 2000, uS Patent 6,092,231.
- [17] E.-I. Vatajelu *et al.*, "Special session: Reliability of hardware-implemented spiking neural networks (snn)," in *Proc. of VTS*, 2019.
- [18] S. A. El-Sayed *et al.*, "Spiking neuron hardware-level fault modeling," in *Proc. of IOLTS*, 2020, pp. 1–4.
- [19] T. Spyrou *et al.*, "Neuron fault tolerance in spiking neural networks," in *Proc. of DATE*, 2021.
- [20] V. Venceslai *et al.*, "Neuroattack: Undermining spiking neural networks security through externally triggered bit-flips," in *Proc. of IJCNN*, 2020, pp. 1–8.
- [21] C. D. Schuman *et al.*, "Resilience and robustness of spiking neural networks for neuromorphic systems," in *Proc. of IJCNN*, 2020, pp. 1–10.
- [22] M. Rastogi *et al.*, "On the self-repair role of astrocytes in stdp enabled unsupervised snns," *Frontiers in Neuroscience*, vol. 14, p. 1351, 2021.
- [23] S. Koppula *et al.*, "Eden: Enabling energy-efficient, high-performance deep neural network inference using approximate dram," in *Proc. of MICRO*, 2019, p. 166–181.

- [24] R. V. W. Putra *et al.*, “Sparkxd: A framework for resilient and energy-efficient spiking neural network inference using approximate dram,” *arXiv preprint arXiv:2103.00421*, 2021.
- [25] J. Gautrais and S. Thorpe, “Rate coding versus temporal order coding: a theoretical approach,” *Biosystems*, vol. 48, no. 1, pp. 57–65, 1998.
- [26] S. Thorpe and J. Gautrais, “Rank order coding,” in *Computational neuroscience*. Springer, 1998, pp. 113–118.
- [27] S. Park *et al.*, “Fast and efficient information transmission with burst spikes in deep spiking neural networks,” in *Proc. of DAC*, 2019, p. 53.
- [28] —, “T2fsnn: Deep spiking neural networks with time-to-first-spike coding,” in *Proc. of DAC*, 2020.
- [29] R. V. W. Putra and M. Shafique, “Q-spinn: A framework for quantizing spiking neural networks,” *arXiv preprint arXiv:2107.01807*, 2021.
- [30] C. Torres-Huitzil and B. Girau, “Fault and error tolerance in neural networks: A review,” *IEEE Access*, vol. 5, pp. 17 322–17 341, 2017.
- [31] M. A. Hanif *et al.*, “Robust machine learning systems: Reliability and security for deep neural networks,” in *Proc. of IOLTS*, 2018, pp. 257–260.
- [32] J. J. Zhang *et al.*, “Building robust machine learning systems: Current progress, research challenges, and opportunities,” in *Proc. of DAC*, 2019, pp. 1–4.
- [33] M. Shafique *et al.*, “Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead,” *IEEE Design & Test*, vol. 37, no. 2, pp. 30–57, 2020.
- [34] J. J. Zhang *et al.*, “Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator,” in *Proc. of VTS*, 2018, pp. 1–6.
- [35] M. A. Hanif and M. Shafique, “Salvagednn: salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping,” *Philosophical Trans. of the Royal Society A*, vol. 378, no. 2164, p. 20190164, 2020.
- [36] H. Hassan *et al.*, “Softmc: A flexible and practical open-source infrastructure for enabling experimental dram studies,” in *Proc. of HPCA*, 2017, pp. 241–252.
- [37] S. Krithivasan *et al.*, “Dynamic spike bundling for energy-efficient spiking neural networks,” in *Proc. of ISLPED*, July 2019, pp. 1–6.
- [38] S. Ghose *et al.*, “Demystifying complex workload-dram interactions: An experimental study,” in *Proc. of SIGMETRICS*, 2019, pp. 93–93.
- [39] R. V. W. Putra *et al.*, “Romanet: Fine-grained reuse-driven off-chip memory access management and data organization for deep neural network accelerators,” *IEEE TVLSI*, vol. 29, no. 4, pp. 702–715, 2021.
- [40] Y. Kim *et al.*, “A case for exploiting subarray-level parallelism (salp) in dram,” in *Proc. of ISCA*, June 2012.
- [41] R. V. W. Putra *et al.*, “Drmmap: A generic dram data mapping policy for energy-efficient processing of convolutional neural networks,” in *Proc. of DAC*, 2020, pp. 1–6.
- [42] P. Diehl and M. Cook, “Unsupervised learning of digit recognition using spike-timing-dependent plasticity,” *Frontiers in Computational Neuroscience*, vol. 9, p. 99, 2015.
- [43] H. Hazan *et al.*, “Bindsnet: A machine learning-oriented spiking neural networks library in python,” *Frontiers in Neuroinformatics*, 2018.
- [44] S. Han *et al.*, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.