SoftSNN: Low-Cost Fault Tolerance for Spiking Neural Network Accelerators under Soft Errors

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
rachmad.putra@tuwien.ac.at, {mh6117, muhammad.shafique}@nyu.edu

ABSTRACT

Specialized hardware accelerators have been designed and employed to maximize the performance efficiency of Spiking Neural Networks (SNNs). However, such accelerators are vulnerable to transient faults (i.e., soft errors), which occur due to high-energy particle strikes, and manifest as bit flips at the hardware layer. These errors can change the weight values and neuron operations in the compute engine of SNN accelerators, thereby leading to incorrect outputs and accuracy degradation. However, the impact of soft errors in the compute engine and the respective mitigation techniques have not been thoroughly studied yet for SNNs. A potential solution is employing redundant executions (re-execution) for ensuring correct outputs, but it leads to huge latency and energy overheads. Toward this, we propose SoftSNN, a novel methodology to mitigate soft errors in the weight registers (synapses) and neurons of SNN accelerators without re-execution, thereby maintaining the accuracy with low latency and energy overheads. Our SoftSNN methodology employs the following key steps: (1) analyzing the SNN characteristics under soft errors to identify faulty weights and neuron operations, which are required for recognizing faulty SNN behavior; (2) a Bound-and-Protect technique that leverages this analysis to improve the SNN fault tolerance by bounding the weight values and protecting the neurons from faulty operations; and (3) devising lightweight hardware enhancements for the neural hardware accelerator to efficiently support the proposed technique. The experimental results show that, for a 900-neuron network with even a high fault rate, our SoftSNN maintains the accuracy degradation below 3%, while reducing latency and energy by up to 3x and 2.3x respectively, as compared to the re-execution technique.

1 INTRODUCTION

SNNs have shown a great potential for obtaining high accuracy in classification tasks (e.g., digit classification, object recognition, etc.) with very low processing power/energy due to their sparse and event-based operations of spiking activity [14]. Moreover, SNNs can perform unsupervised learning using unlabeled data due to their bio-plausible learning mechanism like the spike-timing-dependent plasticity (STDP); see an SNN architecture that supports unsupervised learning in Fig. 1(a). This characteristic is needed in several real-world autonomous systems (e.g., robotics and UAVs) where collecting unlabeled data is easier and cheaper than labeled data [14].

To achieve these benefits while maximizing the performance efficiency of SNN processing, specialized hardware accelerators have been proposed in the literature [1, 3, 4, 6, 11]. However, such hardware accelerators are vulnerable to transient faults (i.e., soft errors) [2] due to high-energy particle strikes, which may come from cosmic rays or packaging materials. These errors manifest as bit flips at the hardware layer, and can propagate to the application layer, resulting in incorrect outputs [2], as shown in Fig. 1(b). Soft errors in the local weight memory/registers and the neurons can affect the functionality of the SNN compute engine, e.g., by corrupting the weight values and the behavior of neuron operations.

1 For conciseness, we use "SNN compute engine" or "compute engine" interchangeably to denote the compute engine of an SNN accelerator.
1.2 Case Study and Research Challenges

We study the impact of faulty weight registers on accuracy, while considering the SNN architecture with direct lateral inhibition and STDP learning as shown in Fig. 1(a) and the hardware architecture shown in Fig. 2. Due to the criticality of soft errors in the memory components that are built with relatively small-sized transistors, as compared to other parts of the circuit, we first perform fault injection experiments on the weight registers of different neurons with random distribution, while considering different fault maps and different fault rates. Details on the experimental setup are provided in Section 4. The experimental results are shown in Fig. 3, from which we derive the following key observations.

- Different combinations of fault maps and fault rates (which represent different possible soft error patterns in real-world conditions) lead to diverse accuracy profiles, even for the same SNN model and workload, indicating its unpredictable nature at design time; see (a) in Fig. 3(a).
- A potential solution is to employ redundant executions (i.e., re-execution) as it does not require hardware modification, but at the cost of huge latency and energy overheads; see Fig. 3(b).

Based on these observations, we highlight the following research challenges in devising solutions for the targeted problem.

- The mitigation technique should recognize any faulty components (weight registers and neurons operations) at run time, to cope with unpredictable run-time scenarios of different soft error profiles.
- The mitigation should not employ re-execution, because it requires huge latency and energy overheads.
- The mitigation should have minimal latency and energy overheads compared to that of the “SNN without mitigation”, thereby making it applicable for latency- and energy-constrained applications.

1.3 Our Novel Contributions

To address the above challenges, we propose SoftSNN, a novel methodology that enables reliable SNN processing on hardware accelerators under soft errors without re-execution. To the best of our knowledge, this work is the first effort that studies the impact of soft errors on SNN execution on the SNN accelerators, and develops a cost-effective mitigation technique. Our SoftSNN methodology employs the following key steps (see overview in Fig. 4).

- Analyzing the SNN fault tolerance under soft errors to understand the impact of faulty SNN components (i.e., synapses and neurons) on accuracy under different fault rates.
- Employing different Bound-and-Protect (BnP) techniques to bound the weight values within a safe range, and protect the neurons from performing faulty operations, based on the information from SNN fault tolerance analysis.
- Employing lightweight hardware support for the BnP techniques to efficiently identify when faulty weight values and neuron operations occur, then perform weight bounding and generate safe neuron behavior that does not cause significant accuracy degradation.

Key Results: We evaluate our SoftSNN methodology using a Python-based framework on a multi-GPU machine. We perform 3 epochs of unsupervised training (3x60K experiments) for each combination of SNN model and workload; and 10K experiments for each combination of SNN model, workload, and fault rate for inference. The experimental results show that, compared to the re-execution, the SoftSNN keeps the accuracy degradation below 3%, while reducing up to 3x and 2.3x of latency and energy respectively, for a 900-neuron SNN with a 0.1 fault rate and the MNIST workload.
routing (area), each synapse employs an adder to sum its weight value with an accumulated value from the previous synapses in the same column, hence a neuron only needs a single input from the connected synapses. In each neuron hardware, the membrane potential ($V_{\text{mem}}$) is increased by the weight ($wgh$) each time a spike arrives, otherwise, the $V_{\text{mem}}$ is decreased. If the $V_{\text{mem}}$ reaches the threshold potential ($V_{\text{th}}$), a spike is produced. Then, the $V_{\text{mem}}$ resets to $V_{\text{reset}}$, and the neuron does not generate spikes for some time, known as refractory period ($T_{\text{ref}}$). Hence, a LIF neuron model has four main operations: (1) $V_{\text{mem}}$ increase, (2) $V_{\text{mem}}$ leak, (3) $V_{\text{mem}}$ reset, and (4) spike generation, as shown in Fig. 2. The membrane potential dynamics are defined by operations (1)-(3).

Soft Error Generation and Distribution: Soft errors typically occur in random locations of a chip [2], leading to a certain fault map. Following are the steps for generating and distributing soft errors (see overview in Fig. 7).

- We consider each weight memory cell and neuron operation as the potential fault locations.
- We generate soft errors considering the given fault rate, and distribute them randomly across the potential fault locations.
- If a fault occurs in a memory cell, we flip the stored bit, which persists until it is overwritten by a new value. If an error occurs in a neuron operation, we randomly select the type of faulty operation, which persists until new parameters are set for the respective neuron.

Impact of faulty neurons: We inject soft errors on the neurons by randomly generating faulty neuron operations. The experimental results are shown in Figs. 3 and 9, from which we derive the following observations.

- Soft errors may increase or decrease weight values, and the increased ones have a more severe impact on accuracy since they trigger the neurons to generate spikes more frequently, thereby dominating classification.
- Increased weights may be recognized by employing the maximum weight value ($wgh_{\text{max}}$) of the pre-trained SNN without soft errors (clean SNN) as a threshold.

Impact of faulty synapses: We inject soft errors on the weight registers by randomly flipping the stored weight bits. The experimental results are shown in Figs. 3 and 9, from which we derive the following observations.

- Soft errors in the $V_{\text{mem}}$ increase operation make the neuron unable to increase $V_{\text{mem}}$, hence this neuron is unable to reach $V_{\text{th}}$ and does not produce any spikes.
- Soft errors in the $V_{\text{mem}}$ leak operation make the neuron unable to decrease $V_{\text{mem}}$.
- Soft errors in the $V_{\text{mem}}$ reset operation make the neuron unable to reset $V_{\text{mem}}$, hence this neuron continuously produces spikes.
- Soft errors in the ‘spike generation’ make the neuron unable to produce any spikes.
- These faulty operations persist until the neuron parameters are replaced with a new set of parameters.

3 SOFTSNN METHODOLOGY

The overview of SoftSNN methodology is shown in Fig. 8, and the detailed discussion of its key steps is provided in Section 3.1-3.3.

3.1 SNN Fault Tolerance Analysis

Our SoftSNN first performs SNN fault tolerance analysis by characterizing the behavior of a given SNN model under different soft error profiles for the underlying hardware, which provides beneficial information for devising lightweight soft error mitigation techniques. To do this, we perform experimental case studies for a 400-neuron network with MNIST as described in the following.

Impact of faulty synapses: We inject soft errors on the weight registers by randomly flipping the stored weight bits. The experimental results are shown in Figs. 3 and 9, from which we derive the following observations.

- Soft errors may increase or decrease weight values, and the increased ones have a more severe impact on accuracy since they trigger the neurons to generate spikes more frequently, thereby dominating classification.
- Increased weights may be recognized by employing the maximum weight value ($wgh_{\text{max}}$) of the pre-trained SNN without soft errors (clean SNN) as a threshold.

Impact of faulty neurons: We inject soft errors on the neurons by randomly generating faulty neuron operations. The experimental results are shown in Fig. 10(a), from which we obtain the following observations.

We use the MNIST dataset as its size enables fast exploration for SNN fault tolerance analysis. Moreover, each SNN input has the same time range and coding for its spike train representation, and the employed STDP learning limits the weights in a certain range of positive values, thereby making any workloads representative for the analysis.
Neuron leakage can achieve tolerable accuracy, as their faulty behavior does not make the neurons dominate classification, and the function for classifying the same input class may be substituted by other (non-faulty) neurons. Therefore, these faulty neurons can still be employed for SNN processing.

- Inference with faulty \( V_{\text{mem}} \) reset can decrease the accuracy significantly, as this faulty behavior makes the neurons' membrane potential stays greater or equal to the threshold potential \( (V_{\text{mem}} \geq V_{th}) \), thereby generating burst spikes and dominating classification. Therefore, these faulty neurons should not be employed for SNN processing.

### Impact of faulty synapses and neurons

We inject soft errors by randomly flipping bits on the weight registers and generate faulty neuron operations. The experimental results in Fig. 10(b) show that the faulty compute engine can severely decrease accuracy, thereby emphasizing the importance of soft error mitigation.

### 3.2 Our Bound-and-Protect (BnP) Techniques

To detect and mitigate soft errors at run time, our SoftSNN methodology employs the Bound-and-Protect (BnP) technique, which bounds the weight values within a safe range that does not make neurons hyper-active (i.e., weight bounding), and protects the neurons from performing faulty operations that can significantly decrease accuracy (i.e., neuron protection).

#### Weight Bounding

This mechanism clips the weight values that are greater or equal to the weight threshold \( (wgh \geq wgh_{th}) \), and replaces them with a pre-defined value \( (wgh_{def}) \), as stated in Eq. 1. Hence, each weight has the bounding value \( wgh_{th} \) that does not trigger neurons’ hyper-activity. To define \( wgh_{th} \), we leverage the SNN fault tolerance characteristics from Section 3.1. We consider the range of weight values from the pre-trained SNN without soft errors (clean SNN) as the safe range, and employ its maximum value as the weight threshold \( (wgh_{th} = wgh_{\max}) \), as shown in Fig. 9(a).

\[
wgh_{th} = \begin{cases} 
\frac{wgh_{def}}{wgh} & \text{if } wgh \geq wgh_{th} \\
1 & \text{otherwise}
\end{cases}
\]  

(1)

#### Neuron Protection

This mechanism focuses on mitigating faulty \( V_{\text{mem}} \) reset’ operations, as suggested by analysis in Section 3.1. We detect the faulty \( V_{\text{mem}} \) reset’ operation in each neuron by monitoring the comparison output of \( V_{\text{mem}} \geq V_{th} \). If the output is ‘true’ for multiple clock cycles (e.g., \( \geq 2 \) clock cycles in this work), then it indicates that the \( V_{\text{mem}} \) reset’ operation does not work properly. To efficiently address this, we disable the spike generation to prevent the corresponding neuron from generating burst spikes.

We leverage these mechanisms for devising three variants of BnP techniques (i.e., BnP1, BnP2, and BnP3), which provide trade-offs in terms of accuracy, latency, and energy for soft error mitigation.

- **BnP1 Technique**: It replaces the weights that are greater or equal to \( wgh_{th} \) with zero. Therefore, the BnP1 can be stated as \( \text{Eq. 1 with } wgh_{def} = 0 \).

- **BnP2 Technique**: It replaces the weights that are greater or equal to \( wgh_{th} \) with the maximum weight value from clean SNN \( (wgh_{\max}) \). Therefore, the BnP2 can be stated as \( \text{Eq. 1 with } wgh_{def} = wgh_{\max} \).

- **BnP3 Technique**: It replaces the weights that are greater or equal to \( wgh_{th} \) with a highly probable value from the weight distribution of clean SNN \( (wgh_{hp}) \). Therefore, the BnP3 can be stated as \( \text{Eq. 1 with } wgh_{def} = wgh_{hp} \).

#### For All BnP Techniques

- We continuously monitor the neuron dynamics, and if the faulty \( V_{\text{mem}} \) reset’ operation occurs, we disable the respective spike generation.

### 3.3 Hardware Support for BnP Techniques

Performing the BnP techniques on the SNN accelerators at run time is challenging, as these accelerators typically have fixed dataflows. Hence, we propose lightweight self-healing hardware enhancements to support the deployment of our BnP techniques on the SNN accelerators without changing the dataflows, as described in the following.

#### Synapse Part: The Synapse enhancements aim at enabling the weight bounding, and they depend on the type of BnP technique.

- In case of the BnP1 Technique, we add (1) a radiation-hardened register for storing the weight threshold \( wgh_{th} \), which is used for all synapses in the compute engine, and (2) the hardened combinatorial logic units for performing a comparison and multiplexing in each synapse; see Fig. 11(a).

- In case of the BnP2 and BnP3 Techniques, we add (1) two radiation-hardened registers for storing the weight threshold \( wgh_{th} \) and the pre-defined weight value \( wgh_{def} \) respectively, which are used for all synapses in the compute engine, and (2) the
hardened combinational logic units for performing a comparison and multiplexing in each synapse; see Fig. 11(b).

Neuron Part: To recognize faulty \( V_{mem} \) reset’ operation, we monitor the comparison output of \( V_{mem} \geq V_{th} \). If the output is ‘true’ for \( \geq 2 \) clock cycles, then the ‘\( V_{mem} \) reset’ operation is faulty. To ensure that such faulty operations do not result in burst spikes, we add an AND logic and a multiplexer to leverage the current and upcoming outputs for determining if the neuron should generate a spike in the next cycle, as shown in Fig. 11(c).

Radiation Hardening: Since our hardware enhancements can also be affected by soft errors, we consider radiation hardened components for all the new hardware extensions to make them resistant to high-energy particle strikes. To do this, the hardening techniques that improve the fabrication process (e.g., re-sizing transistor and insulating substrates [7, 9]) are employed. We only need to harden the additional components, since they will provide correct values which can replace the corrupted bits in the subsequent circuits. Hence, the overhead of the hardening process is relatively low as compared to the full architecture of the SNN hardware, and will be discussed further in Section 5.2 (i.e., area overhead).

5 RESULTS AND DISCUSSION
5.1 Accuracy Comparisons
Fig. 13 presents the experimental results for the accuracy of different mitigation techniques across various test scenarios. The re-execution technique can achieve high accuracy as shown by \( \odot \), since it employs redundant executions to ensure consistent outputs, which indicates that the executions are minimally affected by soft errors. Meanwhile, our BnP techniques (BnP1, BnP2, and BnP3) achieve comparable accuracy to the re-execution, and significantly improve accuracy compared to the SNN without mitigation, i.e., by up to 80% and 47% for MNIST and Fashion MNIST, respectively; see \( \odot \). The reason is that, our techniques employ safe weight values and neuron operations to avoid faulty neural dynamics that significantly decrease accuracy. We observe that the BnP2 has slightly lower accuracy compared to the BnP1 and the BnP3, as it employs \( w_{def} \) whose values have low probability in the weight distribution of clean SNN; see \( \odot \). Hence, the generated neural dynamics do not closely match the neural dynamics of clean SNN. We also observe that the BnP1 and the BnP3 have comparable accuracy, as their \( w_{def} \) are relatively close to each other in the weight distribution of clean SNN; see \( \odot \). Hence, the neural dynamics of the BnP1 and the BnP3 are similar. However, the BnP3 has better applicability for diverse applications than the BnP1, since the \( w_{def} \) in the BnP3 can be updated for different weight distributions. These results show that our BnP techniques are effective for mitigating soft errors in the SNN compute engine at run time without re-execution.

5.2 Latency, Energy, and Area Overheads
Besides accuracy, we also evaluate the design overheads (i.e., latency, energy, and area) incurred by different mitigation techniques.

Latency: Experimental results for latency are shown in Fig. 14(a). We observe that the re-execution technique incurs ~3x latency compared to the SNN without mitigation, as it employs redundant executions for loading parameters on the compute engine and performing neural operations. Meanwhile, our BnP techniques only incur less than 1.06x latency compared to the SNN without mitigation, and reduce latency by up to 3x compared to the re-execution, due to our efficient hardware modifications that minimally affect the latency (i.e., a small number of registers and/or combinational logic units without noticeably affecting the critical path). In this manner, our BnP-enhanced compute engine preserves the existing processing dataflow, and enables reliable SNN executions for latency-constrained (real-time) applications.

Energy Consumption: Experimental results for energy consumption are shown in Fig. 14(b). We observe that, the re-execution technique incurs 3x energy overhead compared to the SNN without mitigation, due to its redundant executions. Meanwhile, our BnP techniques incur less than 1.0x energy consumption when compared to the SNN without mitigation, and reduce the energy consumption by up to 2.3x as compared to the re-execution. Note, compared to the original hardware executing SNN without mitigation, the slight increase in the energy consumption of our BnP-enhanced hardware is due to the additional hardware components...
to enable reliable SNN execution without incurring noticeable latency/performance overheads. Moreover, since the redundant executions are completely avoided, our techniques substantially optimize the energy consumption as compared to the re-execution based mitigation technique.

**Area:** Experimental results for area are shown in Fig. 14(c). We observe that our BnP-enhanced compute engine incurs tolerable area overhead (i.e., 14% for the BnP1, and 18% for the BnP2 and the BnP3) as compared to the compute engine without enhancements. These area overheads mainly come from additional components in synapses, as the synapse crossbar dominates the area of compute engine. Furthermore, the area overhead also represents the cost of the new radiation-hardened components to ensure reliable SNN execution. Note, we only need to harden the additional components for providing correct bits to the subsequent circuits, thereby correcting the corrupted bits and ensuring reliable executions in the respective circuits with low overhead.

In summary, all these results show that our BnP techniques effectively mitigate soft errors, while significantly reducing the latency and the energy of SNN executions as compared to the re-execution based mitigation technique.

6 CONCLUSION

We propose the SoftSNN methodology for mitigating soft errors in SNN accelerators without re-execution. Our SoftSNN analyzes the SNN characteristics under soft errors, performs weight bounding and neuron protection, and devises efficient hardware enhancements to enable the proposed technique. The results show that, our SoftSNN maintains high accuracy while reducing latency and energy, compared to the re-execution based mitigation technique, thereby enabling reliable SNN executions for real-time and energy-efficient applications.

ACKNOWLEDGMENTS

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). This work was also jointly supported by the NYUAD Center for Interacting Urban Networks (CITIES), funded by Tamkeen under the NYU Research Institute Award CG001 and Center for CyberSecurity (CCS), funded by Tamkeen under the NYU Research Institute Award G1104.

REFERENCES

[1] F. Akopyan et al. 2015. TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neuromorphic Chip. IEEE TCD 34, 10 (Oct 2015).
[2] R. Basmann. 2005. Soft Errors in Advanced Computer Systems. IEEE MDT 22, 3 (2005), 258–266.
[3] Q. Chen et al. 2021. A 67.5 pJ/Prediction Accelerator for Spiking Neural Networks in Image Segmentation. IEEE TCSI (2021).
[4] M. Davies et al. 2018. Leolu: A Neuromorphic Manycore Processor with On-Chip Learning. IEEE Micro 38, 1 (Jan 2018), 82–99.
[5] S. A. El-Sayed et al. 2020. Spiking Neuron Hardware-Level Fault Modeling. In IOLTS. 1–4.
[6] C. Frenkel et al. 2019. A 0.086-mm² 12.7-pJ/SOP 64k-Synapse 256-Neuron Online-Learning Digital Spiking Neuromorphic Processor in 28-nm CMOS. IEEE TBCAS 13, 1 (Feb 2019), 145–158.
[7] R. Garg et al. 2009. Circuit-Level Design Approaches for Radiation-Hard Digital Electronics. IEEE TVLSI 17, 6 (2009), 781–792.
[8] H. Hazan et al. 2018. BindsNET: A Machine Learning-Oriented Spiking Neural Networks Library in Python. FNNF 12 (2018), 89.
[9] Q. Huang and J. Jiang. 2019. An Overview of Radiation Effects on Electronic Devices under Severe Accident Conditions in NPPs, Rad-Hardened Design Techniques and Simulation Tools. Progress in Nuclear Energy 114 (2019), 105–120.
[10] R. E. Lyons and W. Vandervalk. 1962. The Use of Triple-Modular Redundancy to Improve Computer Reliability. IBM J. Res. Dev. 6, 4 (1962), 200–209.
[11] E. Painkras et al. 2013. SpNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation. IEEE JSSC 48, 8 (2013), 1943–1953.
[12] R. V. W. Putra et al. 2021. RespWN: Energy-Efficient Fault-Tolerance for Spiking Neural Networks considering Unreliable Memories. In JCAD. 1–9.
[13] R. V. W. Putra et al. 2021. SparkX: A Framework for Resilient and Energy-Efficient Spiking Neural Network Inference using Approximate DRAM. In DAC 379–384.
[14] R. V. W. Putra and M. Shahzad. 2020. FspNN: An Optimization Framework for Memory-Efficient and Energy-Efficient Spiking Neural Networks. IEEE TCD 39, 11 (2020), 3601–3613.
[15] M. Rastogi et al. 2021. On the Self-Repair Role of Astrocytes in STDP Enabled Unsupervised SNNs. FNNS 14 (2021), 1351.
[16] C. D. Schuman et al. 2020. Resilience and Robustness of Spiking Neural Networks for Neuromorphic Systems. In FCNN 1–10.
[17] T. Spyrou et al. 2021. Fault Tolerance in Spiking Neural Networks. In DATE 743–748.
[18] H. Y. Sze. 2000. Circuit and Method for Rapid Checking of Error Correction Codes using Cyclic Redundancy Check. US Patent 6,992,231.
[19] R. Vadlamani et al. 2010. Multicore soft error rate stabilization using adaptive dual modular redundancy. In DATE. 27–32.
[20] E. J. Vatapou et al. 2019. Special Session: Reliability of Hardware-Implemented Spiking Neural Networks (SNN). In VTS 1–8.