Works in Progress in Embedded Computing (WiPiEC) Journal, vol. 11, no. 1, 2025

Reliability-Aware Hyperparameter Optimization for ANN-to-SNN Conversion

Saeed Sharifian[1,*], Mahdi Taheri[2,3], Vahid Rashtchi[1], Ali Azarpeyvand[1,3], Christian Herglotz[2], and Maksim Jenihhin[3]
[1] University of Zanjan, Zanjan, Iran
[2] Brandenburg Technical University, Cottbus, Germany
[3] Tallinn University of technology, Tallinn, Estonia
* sharifian.sa@znu.ac.ir

Abstract—**Spiking Neural Networks (SNNs) have emerged as an energy-efficient alternative to Artificial Neural Networks (ANNs), particularly for edge-computing and safety-critical applications. Unlike conventional ANNs, SNNs leverage sparse event-driven processing to reduce energy consumption while significantly maintaining high computational efficiency. This paper presents a framework designed to optimize the conversion of ANNs into equivalent SNNs while balancing accuracy, reliability, and energy efficiency. The proposed framework systematically explores SNN hyperparameters to identify configurations that achieve superior performance compared to their ANN counterparts. Experimental evaluations on MNIST and Fashion-MNIST datasets with different network topologies demonstrate that the optimized SNNs achieve comparable accuracy while offering in some cases 27.81× and 15.17× lower energy consumption and 1.92× and 1.84× less accuracy drop in the presence of faults, respectively, over the ANN baseline. The results highlight the applicability of SNNs in reliability-critical power-constrained environments.**

Index Terms—deep neural networks, spiking neural networks, reliability, edge applications, safety-critical applications

I. Introduction

Spiking Neural Networks (SNNs) are gaining traction due to their bio-inspired processing, event-driven computation, and energy efficiency. Unlike Artificial Neural Networks (ANNs), SNNs operate similarly to biological neurons, making them well-suited for low-power edge devices and neuromorphic computing [1], [2]. Their sparse and asynchronous nature enhances computational efficiency and scalability, making them ideal for applications in autonomous systems and safety-critical environments [3], [4]. However, the temporal dynamics of SNNs both at the neuron and network levels, along with the non-differentiability of spike functions, have made it difficult to train efficient SNNs [5]. Different studies with various approaches have attempted to adapt backpropagation-based supervised learning algorithms to SNNs [6]. To overcome these challenges and leverage the effectiveness of ANN training, many methods focus on converting well-trained ANNs into functionally equivalent SNNs. One key challenge in transitioning from ANNs to SNNs is ensuring structural consistency between the two architectures [7]. Many deep learning models are highly optimized for specific tasks, and modifying their topology during conversion can result in accuracy loss, increased training complexity, and inefficiencies in hardware deployment. For instance, in edge AI applications like real-time image recognition for autonomous vehicles, maintaining the original ANN topology ensures that pre-trained weights and feature extraction mechanisms remain effective while benefiting from SNNs' energy efficiency [8].

Beyond training, ensuring the reliability of SNNs is critical, especially in noisy or faulty hardware environments where robustness is essential [9]. Several frameworks have addressed specific aspects, such as memory fault tolerance (e.g., ReSpawn [10], rescueSNN [11]), Fault Injection (FI) and analysis (e.g., SpikingJET [12], SpikeFI [13]), or energy-efficient architecture search (e.g., AutoSNN [14]).

Despite these advancements, there remains a lack of unified approaches that convert ANN to SNN, jointly considering reliability, energy efficiency, and accuracy with hyperparameter tuning.

To fill these gaps, this paper proposes an automated framework for generating optimal, reliable, and low-energy consumption SNNs from ANNs. The proposed framework generates SNNs with topological similarity to the original ANN while searching for optimal SNN configurations within them that have balanced accuracy, reliability, and energy consumption. The proposed algorithm utilizes only the ANN architecture to generate new SNNs, distinguishing it from conventional ANN-to-SNN conversion methods that aim to transfer learned parameters for SNN training. SNN networks are learned with the surrogate gradient method. By using FI scenarios, our method ensures that the generated SNNs maintain or exceed the performance of their ANN counterparts while significantly reducing energy consumption.

The key contributions of this paper are:

- A hyperparameter optimization-based technique to ensure a high-performance, high accuracy reliable SNN network
- An automated framework for optimized ANN-to-SNN conversion based on accuracy, reliability, and energy consumption
- Experimental validation on different datasets and network topologies demonstrating the energy, accuracy, and reliability trade-offs between ANNs and SNNs

Manuscript received May 20, 2025; revised July 31, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025.
Paper category: Regular
DOI: doi.org/10.64552/wipiec.v11i1.85

The proposed approach offers a practical and efficient pathway to leveraging SNNs in safety-critical power-constrained edge applications, making them viable alternatives to conventional ANN-based solutions.

The remainder of this paper is structured as follows: Section II presents the proposed methodology. Section III discusses experimental results and the comparison of the initial input ANN and the selected SNN. Finally, Section IV concludes the paper.

II. Proposed Methodology

The framework is developed using *PyTorch* for ANN implementation and *snnTorch* [15] for SNN implementation, both of which support GPU acceleration for training and inference. The *snnTorch* framework supports multiple spiking neuron models, with one of the most widely used being the Leaky Integrate-and-Fire (LIF) model [15] which was also used in this research. Equation (1) represents the discretized form of the LIF neuron's differential equation, which consists of three main components. The neuron's membrane potential is denoted as U. The input component is the product of the input vector X (a spike train of 0s and 1s) and the synaptic weights W. The decay term, governed by the decay factor β, causes the membrane potential to decrease at a rate of β per time step.

The neuron's threshold voltage is represented by θ, which ensures that when the membrane potential reaches a certain threshold, it resets to a predefined value, producing a spike at the output [15]. The spikes generated at the output are denoted as $S_{out}[t] \in \{0, 1\}$. As described in (2), when $S_{out} = 1$, θ is subtracted from the membrane potential; otherwise, no reset occurs. This mechanism, known as the subtraction reset or soft reset mechanism, is widely used in spiking neural networks [15], [16].

$$U[t] = \underbrace{\beta . U[t-1]}_{\text{decay}} + \underbrace{W . X[t]}_{\text{input}} - \underbrace{S_{out}[t-1] . \theta}_{\text{reset}} \tag{1}$$

$$S_{out}[t] = \begin{cases} 1, & if\ U[t] > \theta \\ 0, & otherwise. \end{cases} \tag{2}$$

The *snnTorch* framework also supports multiple spike encoding schemes, such as rate coding and temporal coding. In this study, rate coding is employed, which converts input intensity into a spike count [15].

Since all networks in this study are bias-free, the energy consumption of ANN models is computed using the equation $\sum w.x$ where w and x represent the weight and input data, respectively. The computational operations required in ANN neurons consist of Multiply-Accumulate (*MAC*) operations, which can be theoretically estimated. The total energy consumption is then determined using (3).

$$E_{ANN} = TotalOperations \times E_{MAC} \tag{3}$$

For spiking neurons, computations follow (1). However, in this study, the decay factor β is set to approximately 1, allowing us to disregard its effect for simplification. Additionally, the accumulation term U in (1) is ignored. Consequently, the computational operations in spiking neurons primarily involve the summation of weights, represented as $\sum w$, corresponding to Accumulation (*AC*) operations.

To determine the number of operations in spiking neurons, this study employs a state-of-the-art technique that accurately measures computational complexity by counting the average number of spikes fired across the entire network. This method, which accounts for dataset characteristics, spiking neuron hyperparameters, and encoding schemes, has been widely adopted in recent research [17], [18]. Specifically, after applying the full dataset to the network, the number of spikes generated in each layer is recorded, and the average spike count per layer is reported. The total energy consumption is then estimated by incorporating this spike count into (4) [17].

Table I shows the energy estimation resulting from the implementation of a 32-bit multiplier and adder at 45nm CMOS technology according to reference [19]. Therefore, the E_{MAC} and E_{AC} in (3), (4) can be calculated using this table.

$$E_{SNN} = SpikeCount \times E_{AC} \tag{4}$$

TABLE I. Energy estimation of AC and MAC operation in 45nm CMOS technology

45nm Technology	Energy (pJ)	
	INT	FP
ADD	0.1	0.9
MUL	3.1	3.7
ACC	0.1	0.9
MAC	3.2	4.6

SNNs have multiple hyperparameters affecting their performance, such as the spiking neuron model, time steps, neuron threshold voltage, and neuron reset type [1], [15]. These hyperparameters significantly influence SNN performance, impacting accuracy, spike rate, and energy consumption [1], [15]. A critical factor in SNN efficiency is the selection of an appropriate time step. Higher time steps improve accuracy but increase spike rate, leading to higher latency and energy consumption. Conversely, lower time steps reduce latency but may degrade accuracy. Similarly, adjusting the neuron threshold voltage modifies spiking behavior, influencing both the learning and inference phases. At the same time, these hyperparameters play an essential role in the reliability of the networks. The learning process in this research was conducted using the surrogate gradient method supported by *snnTorch*. Since a key objective of this research is to identify a suitable network for edge applications, adopting an integer number format is crucial compared to floating-point representation. To achieve this, quantization is applied to convert network parameters into an integer format. Notably, the proposed framework supports quantization with arbitrary precision; however, in this study, an 8-bit integer format was used. Quantization improves efficiency in hardware implementations such as FPGAs and ASICs by reducing memory size and computational complexity. To assess reliability, the FI method [20] is used, employing Bit Error Rate

(BER) analysis to simulate faults. This allows systematic evaluation of model robustness without requiring exhaustive FI into all bits, thus reducing computational overhead. To model transient faults, the bit-flip FI method is employed, applying different BER to network parameters (weights) to simulate cumulative faults. The results are presented in terms of accuracy drop as an indicator of network reliability.

Considering the described matters, Fig. 1 provides an overview of the proposed methodology. The workflow consists of three steps designed to find an optimal SNN topologically equivalent to an ANN, maintaining efficient accuracy, reliability, and energy consumption.

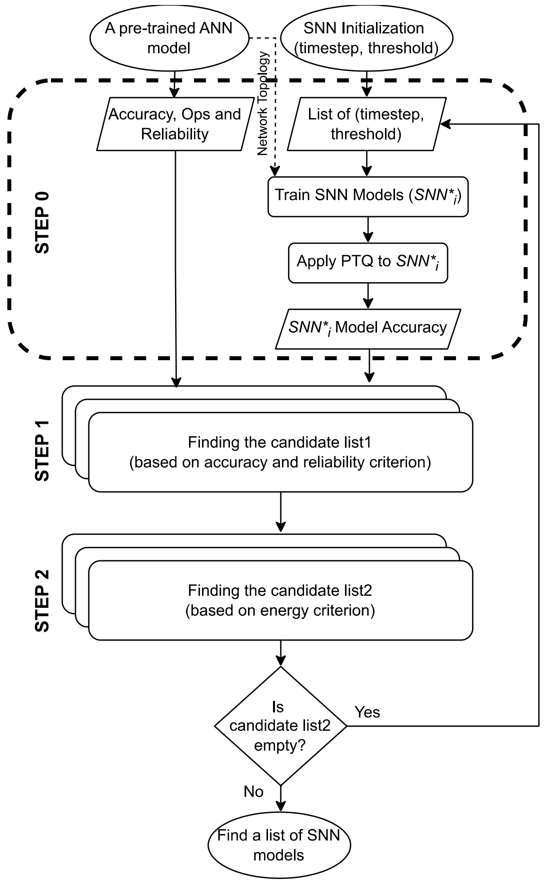

A pre-trained ANN model
SNN Initialization (timestep, threshold)
Network Topology
Accuracy, Ops and Reliability
List of (timestep, threshold)
Train SNN Models (SNN^*_i)
Apply PTQ to SNN^*_i
SNN^*_i Model Accuracy
STEP 0
STEP 1
Finding the candidate list1 (based on accuracy and reliability criterion)
STEP 2
Finding the candidate list2 (based on energy criterion)
Is candidate list2 empty?
Yes
No
Find a list of SNN models

Figure 1. The proposed methodology flowchart

At first, a pre-trained ANN and a set of hyperparameters defining its equivalent SNN are input into the framework.

In **STEP 0**, according to the pseudo-code proposed in Algorithm 1, the framework trains a set of SNN models, denoted as SNN_i^*, using the input hyperparameters. After training, post-training quantization (PTQ) is applied, allowing the user to specify bit-width precision. The accuracy check is performed at the end of this stage on SNN_i^*. On the other hand, the accuracy, reliability, and total operations (Ops) of the ANN are also measured for comparison.

Algorithm 1: ANN-to-SNN Conversion Algorithm

```
Input  : A pre-trained ANN model with a certain topology, Dataset
Output: A set of the optimized SNN models or optionally just an
        optimized SNN model
STEP 0: Test ANN Accuracy, Ops, and Reliability, initialize the SNN
  architecture with a time step and threshold list;
Define $ann$ as the ANN architecture;
Define $snn$ as the SNN architecture;
Define $cur\_params$ as Current hyperparameters;
$\tau \leftarrow$ List of time steps;
$\theta \leftarrow$ List of thresholds;
$accuracy_{ANN} \leftarrow$ Accuracy($ann$);
$operations_{ANN} \leftarrow$ Calculate_ops($ann$);
$reliability_{ANN} \leftarrow$ Fi_test($ann$);
STEP 1: Find the candidate list1 of SNN network (accuracy &
  reliability criteria);
for $\tau_i \in \tau$ do
    for $\theta_i \in \theta$ do
        $cur\_params \leftarrow \tau_i, \theta_i$;
        $snn \leftarrow$ Train_model($SNN_i^*, \tau_i, \theta_i, bits$);
        $accuracy_{SNN} \leftarrow$ Accuracy($snn, \tau_i, \theta_i$);
        if $accuracy_{SNN} \geq accuracy_{ANN}$ then
            $reliability_{SNN} \leftarrow$ Fi_test($snn, \tau_i, \theta_i$);
            if $reliability_{SNN} \geq reliability_{ANN}$ then
                $candidate\_list1 \leftarrow candidate\_list1 + cur\_params$;
            end
        end
    end
end
STEP 2: Find a candidate list through $candidate\_list1$ (energy
  criterion);
for $snn\_items \in candidate\_list1$ do
    $cur\_params \leftarrow \tau_*, \theta_*$;
    $operations_{SNN} \leftarrow$ Calculate_ops($snn\_items, \tau_*, \theta_*$);
    if $operations_{SNN} < operations_{ANN}$ then
        $candidate\_list2 \leftarrow$
          $candidate\_list2 + cur\_params,\ operations_{SNN}$;
    end
end
(Optional Part): Select lowest-energy solution from $candidate\_list2$;
$best\_item \leftarrow$ Min($candidate\_list2$ Ops);
if $best\_item \neq 0$ then
    PASS; Found a set of optimized SNN or (Optional) an optimized
      SNN config;
end
else
    FAIL; Change given parameters (Selecting the list of new
      hyperparameters);
end
```

In **STEP 1**, the accuracy of SNN_i^* is compared to that of the original ANN. If accuracy is maintained or improved, the model undergoes reliability assessment. Only configurations meeting accuracy and then reliability thresholds are stored in *candidate_list1*.

In **STEP 2**, The final step evaluates energy consumption. Hyperparameters such as τ_*, θ_* are items from the previous list that are met, so they are used in this stage. Configurations with lower energy usage than the ANN are stored in *candidate_list2*. If optimal networks exist in *candidate_list2*, the framework returns a selection of viable SNN models. Optionally, the user can request the lowest-energy solution. If no configurations meet the criteria, the input parameters must be adjusted again. Thereby, the algorithm back to the start of STEP 0 according to Algorithm 1, and using an automatic or manual mechanism the list of hyperparameters must be expanded or be selected in other ranges.

The framework systematically searches for an optimal SNN while ensuring minimal performance degradation. If a network from STEP 2 is selected, it is guaranteed to outperform the ANN in terms of energy efficiency and reliability while maintaining accuracy.

By applying the order used in checking accuracy, reliability, and energy consumption, many weak cases are eliminated in a short period of time. According to the experiments performed, the accuracy check of an SNN, depending on the selected hyperparameters and with the topologies chosen in this study, is usually under 3 seconds. However, a reliability test may take several minutes to complete. In STEP 1, all cases with unacceptable accuracy are eliminated, and neither reliability nor energy efficiency tests are performed on them. Also, for SNNs, the energy consumption estimation in this algorithm is calculated simultaneously with their accuracy test.

III. Experimental Results

This section presents the results obtained from the proposed framework and its evaluated parameters. The evaluation considers multiple network topologies, ranging from shallow to deep architectures, as summarized in Table II. Fully connected SNNs are often chosen for experiments because of their simplicity and demonstrated effectiveness. Their ability to leverage the inherent sparsity and event-driven processing of spiking computation results in significant reductions in power consumption and computational load [5]. This makes them especially suitable for applications in edge scenarios such as health monitoring [21]. Key hyperparameters such as time steps and neuron threshold voltage, shown in Table III, are explored. To ensure comprehensive evaluation, a combination of the topologies in Table II and configurations in Table III is tested, allowing for the identification of the most energy-efficient and reliable SNN models.

The analysis is performed using two widely used classification datasets, MNIST and Fashion-MNIST, abbreviated as "M" and "F" in the tables, along with network topologies and configurations. Two forms of reliability assessment are conducted: model-wise and layer-wise. In the model-wise method, FI is applied to the entire network simultaneously, while in the layer-wise method, faults are selectively introduced into specific layers to evaluate their individual resilience.

TABLE II. Different network topologies used in this work

Name	Number of Neurons in layers	Number of Layers
TOP0	32-10	2
TOP1	64-32-10	3
TOP2	128-64-10	3
TOP3	128-64-64-32-10	5
TOP4	512-256-256-128-10	5

TABLE III. The total SNN configs used

Config	Timesteps	Threshold Voltage
C1	10	0.5
C2	10	1.5
C3	30	0.5
C4	30	1.5

As shown in Fig. 2, the first experiment compares a trained and quantized ANN with four SNN variants that share the same topology but differ in configuration. Initially, SNN models are trained with predefined hyperparameters, followed by quantization and comparison with their ANN counterparts. The results indicate that SNN models achieved accuracy levels comparable to their ANN counterparts.

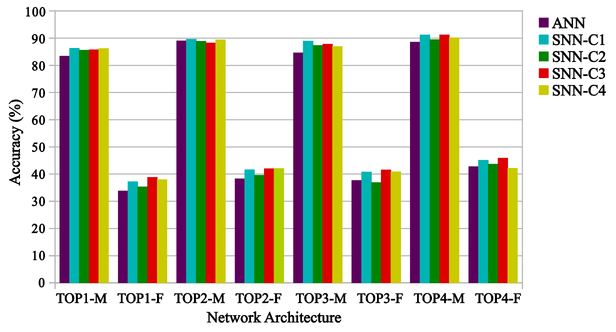
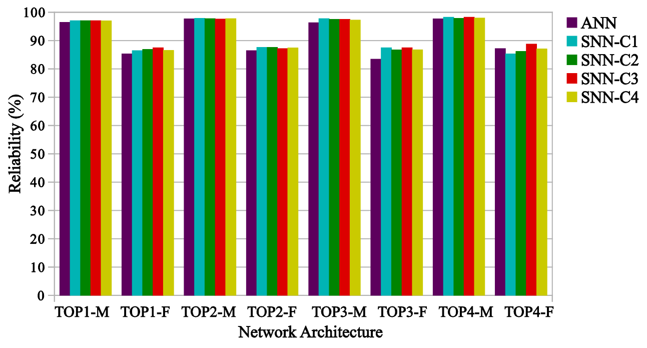


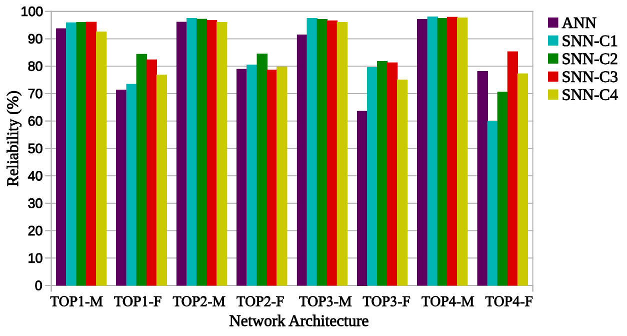
Figure 2. Comparison of accuracy in different architectures

The next study examines the impact of injecting faults into network parameters. For this purpose, four different topologies with four distinct configurations are evaluated, with each graph representing the results for a single BER. As shown in Fig. 3, the experiment covers four BER ranges. The analysis follows a model-wise approach, meaning faults are injected into all hyperparameters of a given model. In each experiment, an ANN is compared with four SNNs of the same topology but different configurations. By analyzing Fig. 3a to 3d, it is evident that networks with different hyperparameters exhibit varying levels of reliability. This underscores the importance of identifying the optimal configuration for an SNN with a given structure. Fig. 3d shows the results of heavy FI as BER equals 0.1, the network has started to lose its parameters, and fault resiliency is unreasonable in this situation.

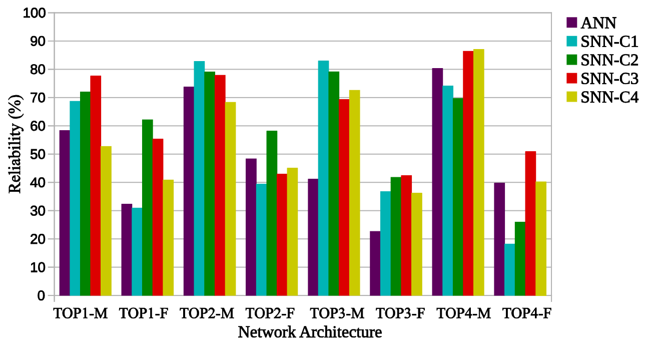
The layer-wise reliability analysis is presented in Fig. 4. Using the proposed framework, a test was conducted across all previously examined cases (various topologies and configurations). After determining the most reliable configuration for each topology, only the best-performing configuration was included in this layer-wise study. This analysis focuses on two topologies: a 3-layer and a 5-layer network. Faults were applied to all layers, and the ANN results



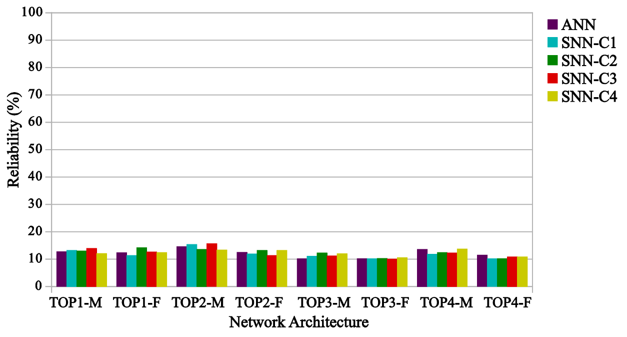
(a) BER = 0.0001



(b) BER = 0.001

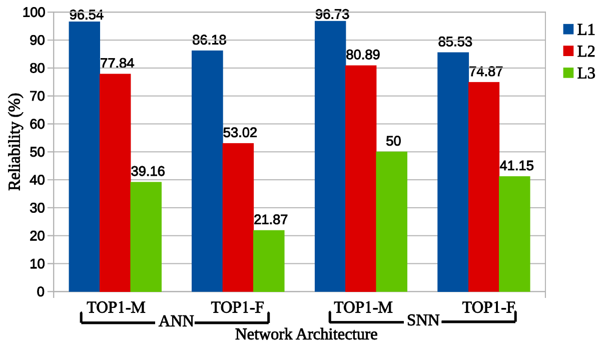


(c) BER = 0.01

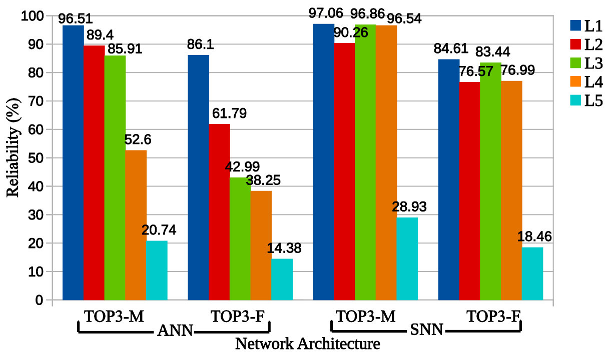


(d) BER = 0.1

Figure 3. Model-wise reliability analysis for some custom network topologies



(a) 3-Layer



(b) 5-Layer

Figure 4. Layer-wise reliability analysis for two 3-Layer and 5-Layer network topologies at BER=0.01

were compared with their corresponding SNNs. In this experiment, only the C2 configuration was analyzed at a BER of 0.01. The results show that SNN layers exhibit greater robustness to faults than their ANN counterparts. For instance, in Fig. 4b, the fourth layer (L4) of the TOP3 SNN achieves 96.54% reliability—1.84× higher than the equivalent ANN topology, which has a reliability of 52.6%.

Accuracy, reliability, and energy consumption trade-offs illustrated in Fig. 5. According to the values in Table I, the figure shows the energy consumption in the two equivalent ANN and SNN topologies. To better highlight differences in energy consumption, two topologies—2-layer and 5-layer networks—are examined, as detailed in Table II. The selected configurations—C2 and C3 for TOP0 and TOP4 respectively—represent the optimal SNN models identified by the proposed framework. As observed, for both topologies and datasets, the accuracy of SNN models remains comparable to their ANN counterparts, while their reliability surpasses that of equivalent ANN models. The figure shows the energy

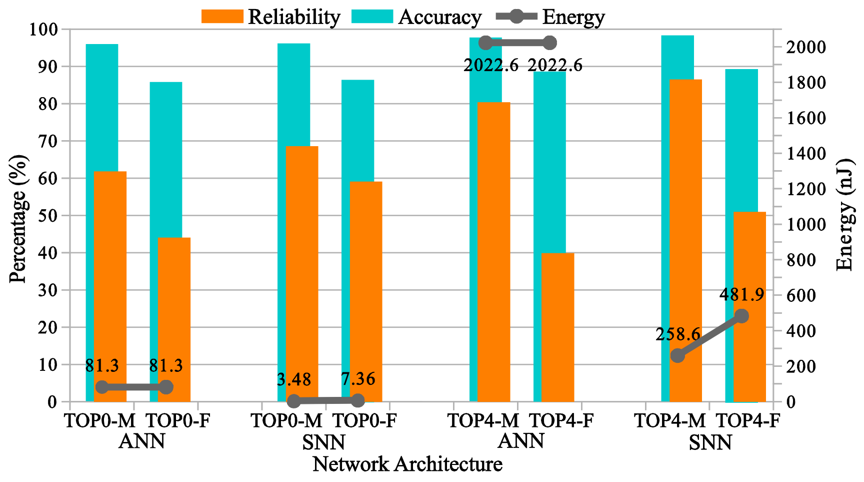


Figure 5. The comparison of accuracy, reliability, and energy for ANN and SNN topologies at BER=0.01

consumption difference between ANN and its equivalent SNN network, especially in a relatively large network. The energy consumption ratio of ANN to SNN in a 2-layer feedforward network (TOP0) is 23.36× for MNIST and 11.05× for Fashion-MNIST dataset. Also in a 5-layer feedforward network (TOP4) is 7.82× for MNIST and 4.2× for Fashion-MNIST dataset. The difference in energy consumption in two SNNs with different datasets is related to the difference in the spike rate of the encoded data of the two datasets, which naturally changes the computational operations and energy consumption.

Based on the data in Fig. 3c and TOP4, if a naïve conversion from ANN to SNN is performed and the proposed framework with three-lateral optimization is not used, the conversion result may end up in one of the configurations such as C1 or C2, which, as is clear from the results, although these configurations meet the accuracy and energy conditions, they deteriorate the reliability in the converted network up to 54.13%. In contrast, the network introduced by the proposed framework, although it meets the accuracy and energy conditions, has also improved its reliability in C3 configuration up to 28.03%.

In some other cases such as TOP3, the proposed framework gives a set of optimizes SNNs, based on Fig. 3c where SNNs showed up to 1.92× and 1.84× better reliability compared to ANNs and lower energy consumption reached up to 27.81× and 15.17× for the MNIST and Fashion-MNIST dataset when using the C2 configuration. Selecting candidate networks without considering reliability may yield better energy efficiency but often lacks fault resilience. Our framework addresses this by balancing all aspects to achieve an optimal trade-off, as reflected in the reported results. Expanding the SNN configuration space could further improve outcomes by offering more design choices.

IV. Conclusion

This paper presented a novel framework for optimizing the conversion of ANNs to SNNs while balancing accuracy, reliability, and energy efficiency. The proposed method systematically explores SNN hyperparameters to identify optimal configurations that maintain accuracy while significantly improving fault tolerance and reducing energy consumption.

Experimental evaluations on MNIST and Fashion-MNIST datasets demonstrated that the optimized SNN models achieved accuracy levels comparable to their ANN counterparts. Moreover, the proposed framework enhanced the reliability of SNNs, as reflected in FI studies, where SNNs showed up to 1.92× and 1.84× lower accuracy degradation under injected faults compared to ANNs in some cases. Additionally, layer-wise reliability assessments confirmed that certain SNN configurations exhibited significantly higher robustness in individual layers than their ANN equivalents.

In terms of energy efficiency, the results showed that SNNs outperformed ANNs by substantial margins. The energy consumption ratio between ANN and SNN reached 27.81× for the MNIST dataset and 15.17× for the Fashion-MNIST dataset in some cases. These findings validate the effectiveness of the proposed approach in achieving energy-efficient and fault-tolerant SNN architectures, making them ideal candidates for edge computing and safety-critical applications.

Acknowledgment

This work was supported in part by the Estonian Research Council grant PUT PRG1467 "CRASHLESS“, EU Grant Project 101160182 “TAICHIP“ and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID "458578717".

References

[1] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[2] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.

[3] J. Ding, Z. Pan, Y. Liu, Z. Yu, and T. Huang, "Robust stable spiking neural networks," *arXiv preprint arXiv*:2405.20694, 2024.

[4] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *arXiv preprint arXiv*:1705.06963, 2017.

[5] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in neuroscience*, vol. 12, p. 409662, 2018.

[6] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in neuroscience*, vol. 10, p.508, 2016.

[7] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, pp. 54–66, 2015.

[8] C. Stöckl and W. Maass, "Optimized spiking neurons can classify images with high accuracy through temporal coding and synaptic weights trained with backpropagation," *Neural Networks*, vol. 143, pp. 100–111, 2021.

[9] T. Spyrou, S. A. El-Sayed, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Reliability analysis of a spiking neural network hardware accelerator," in *2022 Design, Automation &Test in Europe Conference & Exhibition (DATE)*, 2022, pp. 370–375.

[10] R. V. W. Putra, M. A. Hanif, and M. Shafique, "Respawn: Energy-efficient fault-tolerance for spiking neural networks considering unreliable memories," in *2021 IEEE/ACM International Conference OnComputer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[11] R. V. Putra, M. A. Hanif, and M. Shafique, "Rescuesnn: enabling reliable executions on spiking neural network accelerators under permanent faults," *Frontiers in Neuroscience*, vol. 17, p. 1159440, 2023.

[12] A. B. Göğebakan, E. Magliano, A. Carpegna, A. Ruospo, A. Savino, and S. Di Carlo, "Spikingjet: Enhancing fault injection for fully and convolutional spiking neural networks," in *2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2024, pp. 1–7.

[13] T. Spyrou, S. Hamdioui, and H.-G. Stratigopoulos, "Spikefi: A fault injection framework for spiking neural networks," *arXiv preprint arXiv*:2412.06795, 2024.

[14] B. Na, J. Mok, S. Park, D. Lee, H. Choe, and S. Yoon, "Autosnn:Towards energy-efficient spiking neural networks," in *International Conference on Machine Learning*. PMLR, 2022, pp. 16 253–16 269.

[15] J. K. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, "Training spiking neural networks using lessons from deep learning," *Proceedings of the IEEE*, vol. 111, no. 9, pp. 1016–1054, 2023.

[16] S. Venkatesh, R. Marinescu, and J. K. Eshraghian, "Squat: Stateful quantization-aware training in recurrent spiking neural networks," in *2024 Neuro Inspired Computational Elements Conference (NICE)*. IEEE, 2024, pp. 1–10.

[17] S. Barchid, J. Mennesson, J. Eshraghian, C. Djéraba, and M. Bennamoun, "Spiking neural networks for frame-based and event-based single object localization," *Neurocomputing*, vol. 559, p. 126805, 2023.

[18] T. Zhang, S. Xiang, W. Liu, Y. Han, X. Guo, and Y. Hao, "Hybrid spiking fully convolutional neural network for semantic segmentation," *Electronics*, vol. 12, no. 17, p. 3565, 2023.

[19] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*. IEEE, 2014, pp. 10–14.

[20] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "A systematic literature review on hardware reliability assessment methods for deep neural networks," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–39, 2024.

[21] L. Pang, J. Liu, J. Harkin, G. Martin, M. McElholm, A. Javed, and L. McDaid, "Case study—spiking neural network hardware system for structural health monitoring," *Sensors*, vol. 20, no. 18, p. 5126, 2020.