SIPHER: Spike based Neuromorphic Computing for Secure Inference against Bit-Flip Attack

Anonymous authors
Paper under double-blind review

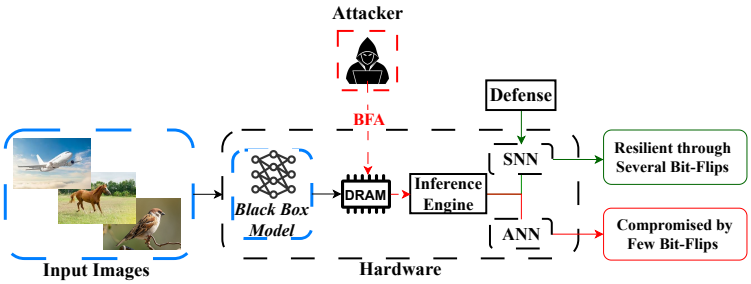
Abstract

Deep Artificial Neural Networks (ANNs) have been shown to be vulnerable to parameter attacks, such as the bit-flip attack, where intentional alterations of network weights can cause significant performance loss. Although extensive research has enhanced the efficacy of these attacks against standard ANN models, robust and efficient defense mechanisms remain underdeveloped. In this work, we propose the spike-based neuromorphic computing paradigm, referred to as SIPHER, as a potent defense strategy that exploits the inherent properties of Spiking Neural Networks (SNNs) to mitigate such attacks. SNNs have emerged as a biologically plausible and energy-efficient alternative to ANNs. However, their fault tolerance and robustness against parameter attacks have not yet been thoroughly investigated. We show that SNNs, on account of their temporal computing capability, effectively neutralize the state-of-the-art progressive bit search method for bit-flip attack, effectively rendering the attack equivalent to random bit-flips. Our results reveal that an 8-bit quantized ResNet-20 SNN requires **145**× more malicious bit-flips compared to ANNs to achieve similar accuracy degradation, with **250**× longer average attack time per bit-flip. The resilience of SNNs increases significantly with model size, with an 8-bit quantized VGG-16 SNN requiring **518**× more bit-flips than ANNs to inflict comparable degradation, thus outperforming state-of-the-art defenses against bit-flip attack. We validate SIPHER on different models and datasets, thereby demonstrating the robustness of the spike-based inference method.

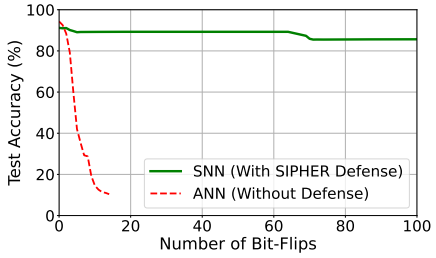
1 Introduction

Deep Artificial Neural Networks (ANNs) have achieved remarkable success in many applications, often attaining or even surpassing human-level performance in tasks such as image recognition (Krizhevsky et al., 2012), speech synthesis (van den Oord et al., 2016), and natural language processing (Touvron et al., 2023). Despite their impressive performance, ANNs have been shown to be vulnerable to adversarial attacks (Szegedy et al., 2014), in which deliberately created input perturbations cause the model to produce incorrect predictions. Although adversarial perturbations are typically imperceptible to the human eye, they can significantly degrade model performance, as demonstrated by previous research efforts (Szegedy et al., 2014; Goodfellow et al., 2015; Papernot et al., 2016a; Carlini & Wagner, 2017). Various defense approaches, such as adversarial training (Madry et al., 2018), ensemble techniques (Tramèr et al., 2018), and gradient regularization (Ross & Doshi-Velez, 2018; Papernot et al., 2016b; Gu & Rigazio, 2014; Miyato et al., 2017), have been developed to enhance ANN robustness against adversarial threats.

While considerable research has focused on protecting neural networks from input-based adversarial attacks, comparatively less attention has been given to attacks targeting model parameters through vulnerabilities in hardware. Such parameter manipulation attacks are particularly concerning because they can potentially render a model unusable without altering the input data, making them difficult to detect. Moreover, the emergence of ANN compression techniques, like pruning and quantization, has enabled the deployment of models on resource-limited platforms (Han et al., 2016) that often lack robust data integrity measures. This increases the susceptibility of models to fault injection attacks, including Row Hammer (Kim et al., 2014) and laser-based attacks (Selmke et al., 2015).



(a) Conceptual illustration of BFA with and without SIPHER.



(b) Test accuracy vs. the number of bit-flips for VGG-16 on the CIFAR-10 dataset.

Figure 1: **Motivation for our proposed defense**. (a) Illustration of the proposed SIPHER methodology, wherein transforming a model into an equivalent Spiking Neural Network (SNN) enhances its robustness against bit-flip attack on weight parameters. (b) Results for VGG-16 ANN (source model without defense) with 8-bit quantized weights show that flipping only 14 bits degrades accuracy from 94% to 10%. On the other hand, VGG-16 SNN (with SIPHER defense methodology) requires significantly more bit-flips to achieve similar degradation in accuracy. This demonstrates that SIPHER effectively prevents the model from becoming a random output classifier under bit-flip attacks (BFAs), thereby mitigating the impact of BFA and enhancing security during model deployment.

The recently demonstrated bit-flip attack, henceforth referred to as BFA, has emerged as one of the most potent parameter attacks that can compromise the security of ANNs, potentially surpassing other threats like adversarial or backdoor attacks (Gu et al., 2017; Coalson et al., 2024). Unlike adversarial attacks that manipulate input data to corrupt model predictions, BFA strikes the physical memory (for instance, DRAM) where the model parameters are stored and causes bit-flips in model weights through hardware fault injection techniques. Researchers in (Rakin et al., 2019) showed that flipping a single bit (the most significant bit of the exponent) in a random weight of a full-precision ANN, which uses IEEE floating-point representation, was sufficient to cause a trained ResNet-18 to malfunction. The catastrophic impact of BFA was subsequently demonstrated on quantized ANNs as well. For example, an 8-bit quantized ResNet-18 was successfully compromised by flipping only 13 bits out of 93 million, effectively reducing its top-1 accuracy on ImageNet from 69% to 0.1%, as shown in (Rakin et al., 2019). The stealthy nature of BFA makes such attacks difficult to detect and easy to evade traditional hardware defense mechanisms. For example, advanced Row Hammer techniques have been shown to circumvent existing memory protection solutions, including Error Correction Codes (ECC) (Cojocar et al., 2019) and Intel Software Guard Extensions (SGX) (Gruss et al., 2018). Moreover, recent practical implementations, such as DeepHammer (Yao et al., 2020), have demonstrated BFA in a real system by exploiting Row Hammer vulnerabilities.

Given the susceptibility of ANNs to adversarial and bit-flip attacks, it is imperative to investigate alternative neural computing paradigms with intrinsic fault tolerance capabilities to effectively resist such attacks. In this context, Spiking Neural Networks (SNNs) (Maass, 1997), inspired by the fault tolerance and computational efficiency of biological neurons, have emerged as a promising solution. SNNs, commonly regarded as the third generation of neural networks, process information through discrete binary spike events (Gerstner & Kistler, 2002). The sparse spike-based computation and communication capability of SNNs have been shown to yield improved energy efficiency in hardware realizations (Rathi & Roy, 2021; Davies et al., 2021). In addition to their computational efficiency, SNNs have been shown to provide enhanced robustness against adversarial perturbations (Sharmin et al., 2019; Liang et al., 2022). Prior works indicate that the temporal dynamics and sparse firing patterns of SNNs contribute to their resilience, particularly in black-box attack scenarios where the attacker lacks detailed insight into the underlying model architecture (Sharmin et al., 2019).

In this work, we propose the spike-based neuromorphic computing methodology, referred to as **SIPHER**, for secure inference that effectively defends against parameter manipulation attacks such as the bit-flip attack

(BFA). We use the ANN-SNN conversion framework, which transforms a pre-trained ANN into an equivalent SNN for energy-efficient inference (Diehl et al., 2015; Rueckauer et al., 2017; Sengupta et al., 2019; Han et al., 2020; Deng & Gu, 2021). This is achieved by reusing the weights of the pre-trained ANN and replacing the ANN activation function, such as ReLU, with corresponding spiking activation, like the Integrate-and-Fire (IF) neuron. SNN inference is subsequently performed on the input over multiple timesteps. Notably, models implemented as ANNs or SNNs appear similar to an attacker in terms of weight parameters and architecture, making it difficult to differentiate and target them effectively, as illustrated in Figure 1a. Our results (discussed in Section 4) indicate that SNN inference exhibits significantly greater resilience than ANN inference when subjected to BFA. For instance, Figure 1b shows that while a VGG-16 ANN can be compromised with fewer than 20 bit-flips, its equivalent SNN requires many more bit-flips to attain a similar level of degradation. Overall, the key contributions of our work are:

- We propose SIPHER, a methodology that maps a source ANN into its equivalent SNN for secure inference against bit-flip attack on model parameters.
- We comprehensively validate the efficacy of SIPHER using different models and datasets, demonstrating its superiority over state-of-the-art defense mechanisms.
- We provide analytical insights (in Section 3.3) that show how the intrinsic temporal dynamics and the spike-based computing capability of SNNs contribute to their robustness, thereby supporting the experimental results reported in this work.

The remainder of the paper is organized as follows. Section 2 reviews the background and related work. Section 3 describes our proposed methodology. Section 4 reports our experiments and results while Section 5 provides a detailed analysis of the findings. Section 6 concludes the paper.

2 Background and Related Works

Overview of Bit-Flip Attack Methodology. The Bit-Flip Attack (BFA) is a simple and efficient method designed to compromise the performance of neural networks by flipping a small number of vulnerable bits in the weight parameters. In this work, we focus on quantized models, similar to those analyzed in (Rakin et al., 2019), because of their widespread use during inference. The quantized weights are stored in two's complement form and referred to as *quantized bit* tensors, denoted by $\{\mathbf{B}_l\}_{l=1}^{L}$, where l corresponds to each layer of a network. BFA targets the most vulnerable bits of the quantized weights, whose alteration causes the maximum increase in the loss function, $\mathcal{L}$. The main objective of the adversary is to maximize $\mathcal{L}$ while limiting the number of flipped bits to a minimum, which can be formally expressed as

$$\begin{aligned} \max_{\{\hat{\mathbf{B}}_l\}_{l=1}^{L}} \quad & \mathcal{L}\Big(f\big(x;\{\hat{\mathbf{B}}_l\}_{l=1}^{L}\big),\, t\Big), \\ \text{subject to} \quad & t = f\Big(x;\{\mathbf{B}_l\}_{l=1}^{L}\Big), \\ & \mathcal{H}\Big(\{\hat{\mathbf{B}}_l\}_{l=1}^{L},\, \{\mathbf{B}_l\}_{l=1}^{L}\Big) \in \{0,1,\ldots,N_b\}, \end{aligned} \tag{1}$$

where x is the input, $f(x;\cdot)$ denotes the network inference function, $\{\hat{\mathbf{B}}_l\}$ is perturbed weight (quantized bit) tensor in l^{th} layer, t is the output of the unperturbed network, and $\mathcal{H}(\cdot,\cdot)$ computes the Hamming distance between the original and the perturbed weights. The parameter N_b specifies the maximum number of bit-flips permitted. Progressive Bit Search (PBS) algorithm, which consists of two stages, was proposed in (Rakin et al., 2019) to efficiently determine the vulnerable bits. In the first stage, known as the *intra-layer search*, the attack computes the gradient of the loss with respect to this quantized bit tensor in each layer, namely $\nabla_{\hat{\mathbf{B}}_l}\mathcal{L}$. The absolute value of these gradients indicates the sensitivity of each bit. Accordingly, the top n_b bits with the largest gradient magnitudes per layer are chosen as vulnerable candidate bits $\hat{\mathbf{b}}_l$, which can be formulated as

$$\hat{\mathbf{b}}_l = \mathrm{Top}_{n_b}\Big(\Big|\nabla_{\hat{\mathbf{B}}_l}\mathcal{L}\Big|\Big). \tag{2}$$

During the following *inter-layer search* stage, the candidate bits from all layers are compared depending on their gradient magnitudes to identify the top n_b candidate bits across the network. The sensitive bits thus determined are flipped to produce the perturbed weight tensor, which results in the maximum increase in $\mathcal{L}$. The PBS algorithm is iteratively performed until either the model performance drops below a pre-specified threshold or the bit-flip budget N_b is fully expended, i.e., $\sum_{i=1}^{N_{\text{itr}}} n_{b_i} \geq N_b$, where N_{itr} denotes the maximum number of PBS iterations. Finally, the effectiveness of BFA is quantified by calculating the *Hamming distance* between the original and perturbed weights. We refer readers to (Rakin et al., 2019) for further details on the PBS method.

Existing Defenses against Bit-Flip Attacks on ANNs. Defense strategies against bit-flip attacks on ANNs can be categorized into weight reconstruction and quantization based methods. Weight reconstruction technique presented in (Li et al., 2020) recomputes the weights prior to inference to minimize the impact of perturbations caused by BFA. Researchers in (He et al., 2020) proposed piece-wise clustering of quantized weights as an effective defense strategy, which was validated in later studies (Bai et al., 2021; Rakin et al., 2019). Asymmetric fixed-point quantization combined with aggressive weight clipping during training was introduced in (Stutz et al., 2021) to enhance security against random bit errors. This method was further improved by per-layer weight clipping in (Stutz et al., 2022), which optimizes the quantization range for each layer to increase resilience against adversarial and random bit errors. Weight clipping-aware training (Chitsaz et al., 2023) has been shown to learn optimal layer-wise quantization ranges during training. Among quantization-based methods, extreme quantization techniques, such as those used in Binary Neural Networks (BNNs), have been shown to significantly improve robustness against BFA (He et al., 2020). However, BNNs incur a notable drop in baseline accuracy compared to higher precision models (Rastegari et al., 2016). To enhance the accuracy and robustness of BNNs, (Rakin et al., 2021) proposed augmenting model capacity during training alongside binarization. SIPHER, in contrast, does not require additional training overhead and provides superior resilience over BNNs (illustrated in Section 4.4).

Introduction to Spiking Neural Networks (SNNs). SNNs compute using discrete spikes instead of continuous activations, allowing for temporal domain processing. We use the Integrate-and-Fire (IF) spiking neuron, which maintains an internal state known as the membrane potential to accumulate the weighted sum of input spikes. The potential $\mathbf{v}_l(t^-)$ of the l-th layer is updated at every discrete time step before a spike is emitted as

$$\mathbf{v}_l(t^-) = \mathbf{v}_l(t-1) + \mathbf{W}_l\, \mathbf{s}_{l-1}(t), \tag{3}$$

where $\mathbf{W}_l$ is the weight matrix connecting layer $l-1$ to layer l and $\mathbf{s}_{l-1}(t)$ denotes the spike train from the preceding layer. A spike is generated when the pre-spike potential exceeds a threshold θ^l, which can be expressed as

$$\mathbf{s}_l(t) = H\Big(\mathbf{v}_l(t^-) - \theta^l\Big), \tag{4}$$

where $H(\cdot)$ represents the Heaviside step function. Once a spike is emitted, the potential is reset according to

$$v_l(t) = v_l(t^-) - s_l(t)\theta^l. \tag{5}$$

A pre-trained ANN can be converted to an SNN by replacing its activation function (e.g., ReLU) with an equivalent spiking activation (e.g., IF neuron). ANN-SNN conversion requires careful calibration of the layer-wise thresholds for near-lossless SNN inference (Cao et al., 2015; Diehl et al., 2015; Sengupta et al., 2019).

Adversarial Robustness in SNNs. Previous studies have shown that SNNs trained using spike-based error backpropagation methods (Lee et al., 2016; Wu et al., 2018; Shrestha & Orchard, 2018; Neftci et al., 2019; Lee et al., 2020) exhibit increased robustness against adversarial attacks compared to ANNs in black-box attack conditions (Sharmin et al., 2019). Furthermore, SNNs trained with backpropagation were shown to possess greater adversarial robustness compared to SNNs transformed from ANNs in both white-box and black-box scenarios (Sharmin et al., 2019; 2020). Recent research has challenged this hypothesis by proposing an adversarially robust ANN-to-SNN conversion algorithm, which optimizes model weights and firing thresholds during a post-conversion fine-tuning phase (Ozdenizci & Legenstein, 2024).

3 SIPHER Methodology

In this section, we introduce SIPHER, a spike-based fault-tolerant inference methodology designed to enhance the robustness of quantized deep neural networks against bit-flip attacks on model parameters. We start by defining our threat model, followed by a description of ANN-SNN conversion algorithm used in this study. Finally, we provide analytical evidence supporting our hypothesis that the intrinsic spike-based temporal computing capability of SNNs contribute to increased resilience against parameter attacks.

3.1 Threat Model

We assume a white-box attacker who has knowledge of the underlying architecture and weights of the deployed neural network, as summarized in Table 1. However, the attacker lacks access to the inference method, specifically whether the model is executed as an ANN or SNN, as well as the hyperparameters and training data. The objective of the attacker is to degrade network performance by flipping as few bits as possible in the weights (Rakin et al., 2019). Our proposed defense, SIPHER, leverages this lack of information about the inference method to mitigate the impact of bit-flip attacks.

Table 1: Summary of the threat model.

Attacker's Knowledge	**Has Access**	**No Access**
Network architecture	✓	
Weight parameters	✓	
Hyperparameters		✓
Training data		✓
Inference method (ANN or SNN)		✓

3.2 ANN-SNN Conversion

We implement the following steps for converting an ANN to its equivalent SNN prior to inference.

- We train the source ANN alongside weight and activation quantization.
- We transfer the trained weight parameters from the ANN to the architecturally equivalent SNN and suitably initialize the neuronal firing thresholds.
- SNN inference is performed by feeding the input over a sufficient number of timesteps to minimize performance loss during conversion.

Weight Quantization. We quantize the weights in both the linear and convolutional layers of the ANN using a symmetric uniform quantizer comprising N_q bits. This reduces the precision of the weights, thereby mitigating the adverse impact of a single random bit-flip prevalent in full-precision models (Rakin et al., 2019). Given the floating-point weights W_l^{fp} at layer l, the quantization step size Δw_l is computed as

$$\Delta w_l = \frac{\max\left(\left|W_l^{\text{fp}}\right|\right)}{2^{N_q-1}-1}. \tag{6}$$

Quantization step size specifies the discrete levels available for mapping the continuous weight values. The quantized weights W^l are then determined by

$$W^l = \text{round}\left(\frac{W_l^{\text{fp}}}{\Delta w_l}\right) \times \Delta w_l. \tag{7}$$

We use the straight-through estimator (Bengio et al., 2013) during training to handle the non-differentiable rounding operation.

Quantization-Clip-Floor-Shift (QCFS) Activation. We use the QCFS activation (Bu et al., 2022), which is a quantized and truncated version of the standard ReLU activation, during ANN training. QCFS activation function has been shown to better approximate the IF neuronal dynamics, resulting in SNNs that provide accuracy comparable to the source ANNs. For pre-activation values $z^l = W^l a^{l-1}$ at layer l, the QCFS activation output a^l is obtained by

$$a^l = \lambda^l \cdot \text{clip}\left(\frac{1}{L}\left\lfloor \frac{z^l L}{\lambda^l} + \varphi \right\rfloor, 0, 1\right), \tag{8}$$

where λ^l is a trainable scaling parameter, L is the number of quantization levels (e.g., $L \in \{2, 4, 8, 16\}$), and φ is a shift term. We jointly train the network with quantized weights and QCFS activations using the SGD optimizer. By quantizing the activations into discrete levels, the QCFS function reduces the network's sensitivity to small perturbations in the pre-activation values z^l. A perturbation ΔW^l, in the weights, results in a change $\Delta z^l = \Delta W^l \cdot a^{l-1}$. For the quantized activation a^l to be affected by Δz^l, the perturbation must be large enough to cross a quantization boundary. The condition for a^l to change can be formulated as

$$|\Delta z^l| \geq \frac{\lambda^l}{2L}, \tag{9}$$

where $\frac{\lambda^l}{L}$ is the activation quantization step size and φ is set to $\frac{1}{2}$. This shows that minor weight perturbations, such as those from a single bit-flip, are much less likely to affect the activations when using QCFS. On the contrary, any change in z^l can proportionally affect a^l for the ReLU function.

3.3 SNN Inference and Robustness Analysis

The trained ANN is converted into an architecturally equivalent SNN, where the QCFS activation function is replaced by an Integrate-and-Fire (IF) neuron. The layerwise firing threshold θ^l of the SNN is programmed to the QCFS scaling factor (λ^l in Equation 8). The expectation of conversion error between the ANN and SNN activations was shown to approach zero if SNN inference is performed for the same number of timesteps as the number of QCFS quantization levels (L in Equation 8) used during ANN training (Bu et al., 2022). We observe that the conversion error is characterized by a finite variance despite having an expected value of zero. The variance of conversion error $\mathbf{Err}_i{}^l$, whose derivation is provided in Section A.1, is estimated as

$$\text{Var}\left(\mathbf{Err}_i^l\right) = \frac{(\theta^l{}^2)}{12}\left(\frac{1+T}{T^2}\right) + \frac{(\theta^l)^2}{12}\left(\frac{1+L}{L^2}\right), \tag{10}$$

where $\mathbf{Err}_i^l$ is conversion error for the i-th neuron and T is the number of inference timesteps. This variance quantifies the additional noise present during SNN inference, distinguishing it from ANN inference with QCFS. Further, the inherent characteristics of SNNs provide several mechanisms that enhance robustness against BFA, as detailed below.

Temporal Integration Attenuates Weight Perturbations. The ANN-SNN conversion method used in this work employs rate coding, where the spiking neuronal output is represented by the average firing rate over inference timesteps. The average firing rate $\phi^l(T)$ across T timesteps can be formulated as

$$\phi^l(T) = \frac{1}{T}\sum_{t=1}^{T} s_l(t), \tag{11}$$

where $s_l(t)$ is the spike output at time t. We now show that the temporal computing capability effectively attenuates the impact of weight perturbations induced by bit-flip attacks. A weight perturbation ΔW^l causes an incremental change $\Delta v_l(t)$ in membrane potential, which can be estimated as

$$\Delta v_l(t) = \Delta W^l \cdot s_{l-1}(t). \tag{12}$$

The perturbed potential triggers an update $\Delta s_l(t)$ in spike output only if it is high enough to cross the firing threshold. The cumulative change $\Delta\phi^l(T)$ in the average firing rate is then obtained by

$$\Delta\phi^l(T) = \frac{1}{T}\sum_{t=1}^{T} \Delta s_l(t). \tag{13}$$

Assuming that the weight perturbation influences the spike output only at a limited number of timesteps $N_p \ll T$, the change in average firing rate can be approximated as

$$\Delta\phi^l(T) \approx \frac{N_p}{T}. \tag{14}$$

Equation 14 shows that the effect of weight perturbation is attenuated by a factor of $1/T$ in SNNs. On the other hand, weight perturbation has a relatively larger impact on ANNs, where the change in activation Δa^l is given by

$$\Delta a^l = \Delta W^l \cdot a^{l-1}. \tag{15}$$

SNNs, on account of temporal integration, require larger or more frequent perturbations to experience the same level of degradation as ANNs.

Threshold-Driven Sparsity Filters Minor Perturbations. The spiking neuron incorporates a firing threshold θ^l, which acts as a filter against minor perturbations. A weight perturbation ΔW^l affects the neuron's output only if it causes the membrane potential $v_l(t)$ to exceed the threshold, which occurs when

$$|\Delta v_l(t)| = |\Delta W^l \cdot s_{l-1}(t)| \geq \theta^l - v_l(t). \tag{16}$$

The weight perturbation in quantized models, which use N_q bits for weight representation, is bounded by

$$|\Delta W^l| \leq \frac{w_{\max}}{2^{N_q-1}}, \tag{17}$$

where $w_{\max}$ is the maximum weight value. We hypothesize that weight quantization along with spiking sparsity makes it difficult for minor weight perturbations to satisfy Equation 16, thus enhancing the resilience of SNNs against BFA, as empirically validated in Section 4.3.

Reset Mechanism Mitigates Error Accumulation. The membrane potential of a spiking neuron is reset after it fires, preventing the accumulation of perturbation over time. The reset mechanism lowers the potential by the firing threshold, as shown in Equation 5. This ensures that even if a bit-flip successfully affects the membrane potential, the influence is limited to a specific time window rather than propagating across timesteps. The temporal isolation, in effect, requires larger perturbations to mount a potent attack.

4 Experimental Results

4.1 Experiment Setup

We perform experiments on three commonly used datasets, namely, CIFAR-10, CIFAR-100 (Krizhevsky et al., 2010), and Tiny-ImageNet. We train models with quantized weights at 6-bit and 8-bit precision, together with QCFS activation at quantization levels of $L = 4$ and $L = 8$. After training, we evaluate our defense mechanism against bit-flip attacks by performing SNN inference for different timesteps. For the BFA implementation, we use an attack sample size of 128 across all experiments. Additional details about the experimental setup are provided in Section A.2. In our work, we adapt BFA for SNNs using surrogate gradients, ensuring that the attack remains potent and that comparisons are fair. Surrogate gradient methods for backpropagating errors in SNNs have matured significantly, allowing accurate and robust gradient approximation through non-differentiable spiking functions (Deng et al., 2022; Zhang & Li, 2020).

4.2 Evaluation Metric

We validate the efficacy of SIPHER against BFA using several key metrics, which are described below. The *Clean Accuracy (CA)* represents the percentage of test samples correctly classified by the clean model. The *Post-Attack Accuracy (PA)* reflects the accuracy of the perturbed model after the attack. We quantify robustness by reporting the number of bit-flips required to either reduce the model accuracy to that of a random classifier (e.g., 10% for CIFAR-10 and 1% for CIFAR-100) or until a predefined maximum number

of attack iterations is reached (e.g., 500 for CIFAR-10 and 1000 for CIFAR-100). The *Average Attack Time (AAT)* (in seconds) specifies the average time taken per iteration to perform BFA. Finally, we introduce *Redundant Bit-Flips (RBF)* to measure the attack inefficiency, estimated as

$$\text{RBF} = \frac{N_{\text{flipped}} - \mathcal{H}}{N_{\text{flipped}}}, \tag{18}$$

where N_{flipped} is the total number of bit-flip operations performed and $\mathcal{H}$ is the hamming distance between the original and perturbed weights. An RBF of zero indicates no inefficiency (each bit was flipped only once), while higher values suggest that the same bits were flipped multiple times.

4.3 Resilience Evaluation

SIPHER on CIFAR-10. We conduct experiments on VGG-16, ResNet-18, ResNet-20, and ResNet-32 model architectures with 6-bit and 8-bit weight quantization. The results, summarized in Table 2, indicate that SIPHER substantially improves robustness compared to ANN inference across all models. For example, while the 8-bit quantized ResNet-20 ANN requires an average of only 9 bit-flips to severely degrade its performance, the equivalent SNN (using $T = 32$ timesteps) requires over **1,311 bit-flips**, which is more than **145**× the number needed by ANN to suffer a similar loss in accuracy. The AAT increased from 0.032 seconds to 8.01 seconds, a **250**× increase in duration, making the attack significantly more time-consuming.

The improvements are even more pronounced in some of the other models. For ResNet-18 SNN, the bit-flip attack reduced accuracy by only about **3%** (from 92.4% to 89.49%) with more than **1,500 bit-flips**. Similarly, VGG-16 SNN maintained accuracy above **25%** with **14,000 bit-flips**, representing a **518**× increase in the required number of bit-flips compared to ANN, which needed only 27 bit-flips on average to reduce its performance to that of a random classifier. Furthermore, we observed that SNN inference significantly outperforms ANN inference (with QCFS function) in terms of robustness. These findings demonstrate the superior efficacy of SIPHER against BFA by requiring more bit-flips and longer average attack times.

SIPHER on CIFAR-100. We analyzed the effectiveness of SIPHER on the CIFAR-100 dataset using an 8-bit quantized ResNet-18 and 6-bit quantized VGG-16, both of which are trained using QCFS activation with $L = 4$. Table 3 clearly indicates that both ResNet-18 SNN and VGG-16 SNN exhibit improved robustness against BFA by requiring an order of magnitude more bit-flips while still having a higher post-attack accuracy compared to ANN. For instance, the top-1 accuracy of ResNet-18 SNN decreased only slightly from 73.73% to 68.69% even after incurring **1,957 bit-flips**, demonstrating significant resilience. Overall, these results further reinforce the capability of SIPHER to provide enhanced resistance to bit-flip attacks.

SIPHER on Tiny-ImageNet. To further validate the robustness claims of SIPHER, we conducted experiments on the Tiny-ImageNet dataset using an 8-bit quantized VGG-16 architecture. The results, summarized in Table 4, demonstrate SIPHER's significant robustness advantage (**200**× increase in the number of bit-flips and **15**× longer AAT) over traditional ANN implementations against bit-flip attack.

4.4 Comparison with Existing Defenses

Several defenses have been proposed to protect neural networks against bit-flip attack, predominantly using standard neural network architectures (e.g., ANNs, BNNs). Table 5 compares the performance and robustness of these defenses with our proposed SIPHER approach using the 8-bit quantized ResNet-20 model inferred on the CIFAR-10 dataset. Our results indicate that SIPHER, which deploys ResNet-20 SNN during inference, significantly outperforms existing methods like piece-wise clustering (He et al., 2020), binary quantization (He et al., 2020), and weight reconstruction (Li et al., 2020). Our method is comparable in terms of robustness to RA-BNN (Rakin et al., 2021). We would like to point out that the efficacy of SIPHER defense increases with model size; for larger models, the required number of bit-flips can reach the order of $\mathbf{10^4}$, as illustrated in Section 4.3.

Table 2: SIPHER robustness analysis on CIFAR-10 for various models. Vanilla models are baseline models with ReLU activation and weight quantization only. Models using QCFS activation are indicated with their quantization levels (L). The ANN/SNN column specifies the inference type (ANN or SNN), with timesteps (T) for SNNs. Clean Accuracy (CA) and Post-attack Accuracy (PA) denote top-1 accuracies. Average Attack Time (AAT) is specified in seconds. Bit-flips are measured for a maximum of 500 attack iterations.

Model & Quantization	ANN/SNN	CA (%)	PA (%)	AAT (sec)	Bit Flips
VGG-16 (8-bit)					
Vanilla	ANN	94.31	10.39	0.054	27
QCFS (L=4)	ANN	93.38	10.74	0.08	48
SIPHER: QCFS (L=4)	**SNN (T=16)**	**93.06**	**79.67**	**1.9**	**1250**
VGG-16 (6-bit)					
Vanilla	ANN	94.70	10.15	0.053	32
QCFS (L=8)	ANN	93.81	10.3	0.079	21
SIPHER: QCFS (L=8)	**SNN (T=16)**	**93.09**	**39.05**	**2.3**	**987**
ResNet-18 (8-bit)					
Vanilla	ANN	95.30	10.4	0.22	55
QCFS (L=4)	ANN	94.59	10.85	0.138	107
SIPHER: QCFS (L=4)	**SNN (T=4)**	**92.42**	**89.49**	**1.73**	**1505**
ResNet-18 (6-bit)					
Vanilla	ANN	95.51	10.2	0.22	53
QCFS (L=4)	ANN	94.57	10.95	0.40	111
SIPHER: QCFS (L=4)	**SNN (T=4)**	**92.73**	**89.40**	**1.50**	**1284**
ResNet-20 (8-bit)					
Vanilla	ANN	91.14	10.02	0.032	9
QCFS (L=8)	ANN	90.53	10.78	0.052	18
SIPHER: QCFS (L=8)	**SNN (T=32)**	**90.37**	**10.05**	**8.01**	**1311**
ResNet-20 (6-bit)					
Vanilla	ANN	91.58	10.1	0.031	13
QCFS (L=8)	ANN	90.68	10.43	0.052	17
SIPHER: QCFS (L=8)	**SNN (T=32)**	**90.34**	**10.01**	**3.94**	**328**
ResNet-32 (8-bit)					
Vanilla	ANN	92.66	10.93	0.067	14
QCFS (L=4)	ANN	91.59	10.17	0.118	22
SIPHER: QCFS (L=4)	**SNN (T=32)**	**91.29**	**14.88**	**9.56**	**647**
ResNet-32 (6-bit)					
Vanilla	ANN	92.55	10.2	0.066	12
QCFS (L=4)	ANN	91.33	10.78	0.116	21
SIPHER: QCFS (L=4)	**SNN (T=32)**	**92.73**	**12.73**	**10.29**	**645**

5 Analysis

In this section, we perform an in-depth analysis of bit-flips induced by BFA during SNN inference and compare them with those occurring in ANN inference.

Bit-Flip Behavior in SNN vs. ANN Inference. In ANN inference, the BFA typically requires flipping only a few bits to significantly degrade model accuracy, and the same bit is rarely flipped multiple times; consequently, the RBF metric is often zero. In each attack iteration, flipping a single bit is usually sufficient to increase the loss, as the top n_b gradients are non-zero. This enables PBS algorithm (Rakin et al., 2019) to efficiently identify and flip the vulnerable bit per attack iteration.

Table 3: SIPHER robustness analysis on CIFAR-100. Clean Accuracy (CA) and Post-attack Accuracy (PA) are presented as top-1/top-5 accuracies. Average Attack Time (AAT) is specified in seconds. Bit-flips are measured for a maximum of 1000 attack iterations.

Model & Quantization	ANN/SNN	CA (%)	PA (%)	AAT (sec)	Bit Flips
ResNet-18 (8-bit)					
Vanilla	ANN	76.26/94.02	1.9/6.47	0.084	69
QCFS (L=4)	ANN	77.19/94.11	1.87/29.5	0.174	144
SIPHER: QCFS (L=4)	**SNN (T=4)**	**73.73/92.53**	**68.69/91.74**	**1.123**	**1957**
VGG-16 (6-bit)					
Vanilla	ANN	75.44/92.62	1.22/6.49	0.051	12
QCFS (L=4)	ANN	74.13/93.72	1.72/11.40	0.074	43
SIPHER: QCFS (L=4)	**SNN (T=8)**	**72/92.78**	**12.03/30.17**	**1.111**	**1897**

Table 4: Results for 8-bit VGG-16 on Tiny-ImageNet dataset.

Model	CA (%)	PA (%)	AAT	Bit-flips
Vanilla	54.42	10.2	0.2	11
QCFS ($L = 4$)	49.78	10.05	0.305	82
SIPHER ($L = 4$, $T = 8$)	49.35	10.1	**3.1**	**2305**

Table 5: Comparison of SIPHER with the state-of-the-art defenses using 8-bit quantized ResNet-20 on the CIFAR-10 dataset.

Defenses	CA	PA	Order of bit-flips
Weight Reconstruction (Li et al., 2020)	90.83	10.00	10
Piece-wise Clustering (He et al., 2020)	90.02	10.07	10
Binary Weight (He et al., 2020)	88.36	10.13	10^2
RA-BNN (Rakin et al., 2021)	**90.18**	**10.00**	$\mathbf{10^3}$
SIPHER ($L = 8$, $T = 32$)	**90.37**	**10.05**	$\mathbf{10^3}$

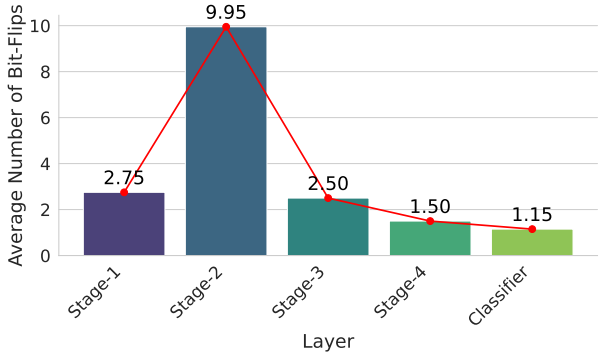
Conversely, in SNN inference, a larger number of bit-flips is often necessary to experience similar levels of accuracy degradation. Flipping a single bit per attack iteration may not increase the loss. Therefore, multiple bits may be flipped in each attack iteration. In addition, we observe that the magnitudes of the top n_b gradients can become zero, which causes PBS method to get stuck in local minima during the search for vulnerable bits. As the attack progresses, the same bits can be flipped multiple times, resulting in a **higher RBF** in SNN inference (as summarized in Table 6) and rendering the attack equivalent to random bit-flips.

Table 6: Analysis of RBF for VGG-16 with QCFS activation ($L = 4$) during SNN inference on CIFAR-10 and CIFAR-100 datasets, using 8-bit and 6-bit weight quantization, respectively.

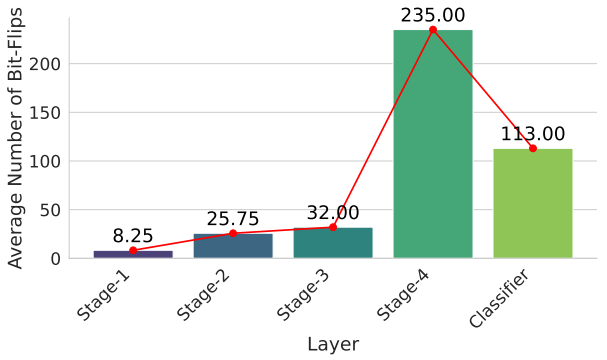
SNN (T)	Dataset	Bit Flips / Hamming Distance	RBF
$T = 16$	CIFAR-10	14,024 / 6,694	**0.52**
$T = 8$	CIFAR-100	1,897 / 1,227	**0.35**

Layer-Wise Sensitivity to BFA. We now analyze the layer-wise distribution of bit-flips required to degrade the ResNet-20 model. Our results, shown in Figure 2, indicate that most bit-flips are concentrated in the initial layers for ANN inference, suggesting that these layers produce higher gradient magnitudes and are therefore more susceptible to perturbations. In contrast, SNN inference exhibits a shift in sensitivity toward

the later layers. Specifically, Stage-4 layers are more susceptible due to their higher gradient magnitudes. Similarly, the classifier layer, which has been implemented as a linear layer without IF neurons similar to that used in ANN, contributes relatively more to the overall gradients.



(a) Average number of bit-flips per layer in ANN inference.



(b) Average number of bit-flips per layer in SNN inference at $T = 16$.

Figure 2: Layer-wise distribution of the average number of bit-flips required to compromise the ResNet-20 model on the CIFAR-10 dataset. Here, 'Stage-1' to 'Stage-4' correspond to successive residual blocks in the network, and 'Classifier' refers to the final fully-connected layer. The average is computed across different weight bit-widths (6-bit and 8-bit) and QCFS levels ($L = 4$ and $L = 8$).

The layer-wise distribution of bit-flips differs significantly between ANN and SNN inference, although the underlying weights remain largely identical in both settings. We conducted additional experiments by selectively freezing the sensitive SNN layers during BFA, specifically the Stage-4 block and the final classifier. With these layers excluded, the number of bit-flips required to achieve comparable degradation reduces considerably, as illustrated in Figure 3. This shows that the enhanced robustness observed during SNN inference largely originates in the later layers. When the attack is forced to rely solely on the initial layers, the robustness benefit of SIPHER diminishes.

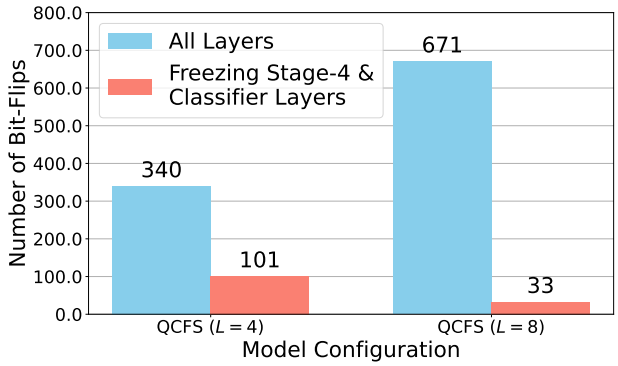


Figure 3: Impact of freezing sensitive layers on the number of bit-flips required to compromise the 6-bit quantized ResNet-20 model during SNN inference using $T = 16$ timesteps.

Impact of Timesteps on Attack Complexity. When mounting attacks on ANN model architectures, BFA has a computational cost proportional to the number of layers and the candidate bits examined (Rakin et al., 2019). However, for SNNs, this cost increases drastically because each attack iteration must process information over multiple timesteps. Although the weight parameters remain identical, calculating how bit-flips affect the network requires tracing through the entire temporal sequence during the forward pass. This effectively multiplies the computational complexity by the number of timesteps, creating a natural barrier that makes attacks on SNNs significantly more time-consuming than on ANNs.

Practical Realization of SIPHER. SIPHER requires that the inference engine (illustrated in Figure 1a) is capable of executing both ANNs and SNNs. SIPHER can be realized using general-purpose computing platforms such as CPUs and GPUs as well as custom hardware accelerators that support both ANN and SNN processing elements (Pei et al., 2019; Singh et al., 2020). SIPHER does not introduce any additional computational overhead during training compared to other defenses. However, it leads to higher inference latency due to processing over multiple time steps, albeit with higher energy efficiency over ANNs.

6 Conclusion

In this work, we introduced SIPHER, a methodology that transforms an ANN into an architecturally equivalent SNN, for enhancing security against bit-flip attacks on neural network weights. The improved robustness of SIPHER stems from the intrinsic spike-based sparse computing capability, temporal dynamics, and fault tolerance of spiking neurons. Our extensive experiments revealed that SNNs require up to three orders of magnitude more bit-flips to experience similar levels of performance degradation as their ANN counterparts. In addition, we presented analytical insights to quantify the distinctive attributes of SNN inference, supporting the experimental results presented in this study.

References

Jiawang Bai, Baoyuan Wu, Yong Zhang, Yiming Li, Zhifeng Li, and Shu-Tao Xia. Targeted attack against deep neural networks via flipping limited weight bits. In *ICLR*, 2021. 4

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 5

Tong Bu, Wei Fang, Jianhao Ding, PengLin Dai, Zhaofei Yu, and Tiejun Huang. Optimal ANN-SNN conversion for high-accuracy and ultra-low-latency spiking neural networks. In *ICLR*, 2022. 6, 16, 17

Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. In *IJCV*, pp. 54–66, 2015. 4

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy (SP)*, pp. 39–57, 2017. 1

Kamran Chitsaz, Goncalo Mordido, Jean-Pierre David, and François Leduc-Primeau. Training DNNs resilient to adversarial and random bit-flips by learning quantization ranges. *Transactions on Machine Learning Research*, 2023. 4

Zachary Coalson, Jeonghyun Woo, Shiyang Chen, Yu Sun, Lishan Yang, Prashant Nair, Bo Fang, and Sanghyun Hong. PrisonBreak: Jailbreaking large language models with fewer than twenty-five targeted bit-flips. *arXiv preprint arXiv:2412.07192*, 2024. 2

Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks. pp. 55–71. IEEE, 2019. 2

Mike Davies, Andreas Wild, Garrick Orchard, Yulia Sandamirskaya, Gabriel A Fonseca Guerra, Prasad Joshi, Philipp Plank, and Sumedh R Risbud. Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proceedings of the IEEE*, 109(5):911–934, 2021. 2

Shikuang Deng and Shi Gu. Optimal conversion of conventional artificial neural networks to spiking neural networks. *arXiv preprint arXiv:2103.00476*, 2021. 3

Shikuang Deng, Yuhang Li, Shanghang Zhang, and Shi Gu. Temporal efficient training of spiking neural network via gradient re-weighting. In *International Conference on Learning Representations*, 2022. 7

Terrance DeVries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 18

Peter U. Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2015. 3, 4

Wulfram Gerstner and Werner M Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002. 2

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015. 1

Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another flip in the wall of rowhammer defenses. In *IEEE Symposium on Security and Privacy (SP)*, pp. 245–261, 2018. 2

Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068*, 2014. 1

Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017. 2

Bing Han, Gopalakrishnan Srinivasan, and Kaushik Roy. Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13558–13567, 2020. 3

Song Han, Huizi Mao, and William J Dally. Deep Compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016. 1

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 18

Zhezhi He, Adnan Siraj Rakin, Jingtao Li, Chaitali Chakrabarti, and Deliang Fan. Defending and harnessing the bit-flip based adversarial weight attack. In *CVPR*, pp. 14095–14103, 2020. 4, 8, 10

Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *Proceedings of the 41st Annual International Symposium on Computer Architecture (ISCA)*, pp. 361–372. IEEE, 2014. 1

Alex Krizhevsky, Vinod Nair, and Geoffery E Hinton. CIFAR- 10 and 100 (canadian institute for advanced research). "http://www.cs.toronto.edu/~kriz/cifar.html", 2010. 7

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pp. 1097–1105, 2012. 1

Chankyu Lee, Syed Shakib Sarwar, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in Neuroscience*, 14, 2020. 4

Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience*, 10:508, 2016. 4

Jingtao Li, Adnan Siraj Rakin, Yan Xiong, Liangliang Chang, Zhezhi He, Deliang Fan, and Chaitali Chakrabarti. Defending bit-flip attack through dnn weight reconstruction. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020. 4, 8, 10

Ling Liang, Kaidi Xu, Xing Hu, Lei Deng, and Yuan Xie. Towards robust spiking neural network against adversarial perturbation. In *NIPS*, 2022. 2

Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997. 2

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018. 1

Takeru Miyato, Andrew M Dai, and Ian Goodfellow. Adversarial training methods for semi-supervised text classification. In *ICLR*, 2017. 1

Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019. 4

Ozan Ozdenizci and Robert Legenstein. Adversarially robust spiking neural networks through conversion. *Transactions on Machine Learning Research*, 2024. 4

Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 372–387, 2016a. 1

Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 582–597, 2016b. 1

Jing Pei, Lei Deng, Sen Song, Mingguo Zhao, Youhui Zhang, Shuang Wu, Guanrui Wang, Zhe Zou, Zhenzhi Wu, Wei He, et al. Towards artificial general intelligence with hybrid tianjic chip architecture. *Nature*, 572(7767):106–111, 2019. 12

Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-Flip Attack: Crushing neural network with progressive bit search. In *ICCV*, pp. 1211–1220, 2019. 2, 3, 4, 5, 9, 11

Adnan Siraj Rakin, Li Yang, Jingtao Li, Fan Yao, Chaitali Chakrabarti, Yu Cao, Jae sun Seo, and Deliang Fan. RA-BNN: Constructing robust accurate binary neural network to simultaneously defend adversarial bit-flip attack and improve accuracy. *arXiv preprint arXiv:2103.13813*, 2021. 4, 8, 10

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pp. 525–542. Springer, 2016. 4

Nitin Rathi and Kaushik Roy. DIET-SNN: A low-latency spiking neural network with direct input encoding and leakage and threshold optimization. *IEEE Transactions on Neural Networks and Learning Systems*, 34(6):3174–3182, 2021. 2

Andrew Slavin Ross and Finale Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018. 1

Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in neuroscience*, 11:682, 2017. 3

Bodo Selmke, Stefan Brummer, Johann Heyszl, and Georg Sigl. Precise laser fault injections into 90-nm and 45-nm sram-cells. In *Revised Selected Papers of the 14th International Conference on Smart Card Research and Advanced*, pp. 193–205. Springer-Verlag, 2015. 1

Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience*, 2019. 3, 4

Saima Sharmin, Priyadarshini Panda, Syed Shakib Sarwar, Chankyu Lee, Wachirawit Ponghiran, and Kaushik Roy. A comprehensive analysis on adversarial robustness of spiking neural networks. In *International Joint Conference on Neural Networks*, pp. 1–8, 2019. 2, 4

Saima Sharmin, Nitin Rathi, Priyadarshini Panda, and Kaushik Roy. Inherent adversarial robustness of deep spiking neural networks: Effects of discrete input encoding and non-linear activations. In *ECCV*, pp. 399–414. Springer-Verlag, 2020. 4

Sumit Bam Shrestha and Garrick Orchard. SLAYER: Spike layer error reassignment in time. In *NIPS*, pp. 1412–1421, 2018. 4

Sonali Singh, Anup Sarma, Nicholas Jao, Ashutosh Pattnaik, Sen Lu, Kezhou Yang, Abhronil Sengupta, Vijaykrishnan Narayanan, and Chita R Das. Nebula: A neuromorphic spin-based ultra-low power architecture for snns and anns. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 363–376. IEEE, 2020. 12

David Stutz, Nandhini Chandramoorthy, Matthias Hein, and Bernt Schiele. Bit error robustness for energy-efficient dnn accelerators. *arXiv preprint arXiv:2006.13977*, 2021. 4

David Stutz, Nandhini Chandramoorthy, Matthias Hein, and Bernt Schiele. Random and adversarial bit error robustness: Energy-efficient and secure dnn accelerators. *arXiv preprint arXiv:2104.08323*, 2022. 4

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2014. 1

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 1

Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. In *ICLR*, 2018. 1

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016. 1

Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience*, 12:331, 2018. 4

Fan Yao, Adnan Siraj Rakin, and Deliang Fan. DeepHammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 1463–1480, 2020. 2

Wenrui Zhang and Peng Li. Temporal spike sequence learning via backpropagation for deep spiking neural networks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020. 7

A Appendix

A.1 Proof of the Conversion Error Variance Between ANN Inference with QCFS and SNN Inference

In this section, we provide detailed proof of the variance of the conversion error between an Artificial Neural Network (ANN) with Quantized Clip-Floor-Shift (QCFS) activation functions and its equivalent Spiking Neural Network (SNN). While the expectation of the conversion error has been shown to be zero in the QCFS paper (Bu et al., 2022), we focus here on calculating the variance of the conversion error, which quantifies the additional noise present during SNN inference. Consider an ANN layer l with the QCFS activation function, output is given by:

$$\phi^l_{\text{ANN}} = \frac{\lambda^l}{L} \cdot \left\lfloor \frac{z^l L}{\lambda^l} + \varphi \right\rfloor, \tag{19}$$

where z^l is the pre-activation input to layer l, λ^l is a scaling parameter (trainable during training), L is the number of quantization levels in the activation function (e.g., $L \in \{2, 4, 8, 16\}$), and φ is the shift term, set to $\varphi = \frac{1}{2}$. In the equivalent SNN, the neuron's output over T time steps is given by:

$$\phi^l_{\text{SNN}} = \frac{\theta^l}{T} \cdot \left\lfloor \frac{z^l T + v^l(0)}{\theta^l} \right\rfloor, \tag{20}$$

where $\theta^l = \lambda^l$ is the firing threshold, $v^l(0) = \theta^l \varphi = \frac{\theta^l}{2}$ is the initial membrane potential (since $\varphi = \frac{1}{2}$), and T is the number of time steps during SNN inference. Our goal is to analyze the variance of the conversion error between the ANN and SNN activations, defined by:

$$\mathbf{Err}^l = \phi^l_{\text{SNN}} - \phi^l_{\text{ANN}}. \tag{21}$$

While the expected value $\mathrm{E}\left[\mathbf{Err}^l\right] = 0$ has been established in (Bu et al., 2022). Substituting the expressions from Equations equation 19 and equation 20 into Equation equation 21, we have:

$$\begin{aligned} \mathbf{Err}^l &= \frac{\theta^l}{T} \cdot \left\lfloor \frac{z^l T + v^l(0)}{\theta^l} \right\rfloor - \frac{\lambda^l}{L} \cdot \left\lfloor \frac{z^l L}{\lambda^l} + \varphi \right\rfloor \\ &= \theta^l \left(\frac{1}{T} \left\lfloor \frac{z^l T + \theta^l \varphi}{\theta^l} \right\rfloor - \frac{1}{L} \left\lfloor \frac{z^l L}{\theta^l} + \varphi \right\rfloor \right), \end{aligned} \tag{22}$$

since $\theta^l = \lambda^l$ and $v^l(0) = \theta^l \varphi$. As every element in the vector z^l behaves identically, we only need to consider a single element z^l_i. Defining $\mathbf{Err}^l_i$ as the conversion error for the i-th element:

$$\mathbf{Err}^l_i = \theta^l \left(\frac{1}{T} \left\lfloor \frac{z^l_i T + \theta^l \varphi}{\theta^l} \right\rfloor - \frac{1}{L} \left\lfloor \frac{z^l_i L}{\theta^l} + \varphi \right\rfloor \right). \tag{23}$$

We can decompose the error by adding and subtracting z^l_i:

$$\begin{aligned} \mathbf{Err}^l_i &= \left(\frac{\theta^l}{T} \left\lfloor \frac{z^l_i T + \theta^l \varphi}{\theta^l} \right\rfloor - z^l_i \right) \\ &\quad + \left(z^l_i - \frac{\theta^l}{L} \left\lfloor \frac{z^l_i L}{\theta^l} + \varphi \right\rfloor \right) \qquad (24) \\ &= \mathbf{e}_{\text{SNN}} + \mathbf{e}_{\text{ANN}}, \qquad (25) \end{aligned}$$

where:

$$\mathbf{e}_{\text{SNN}} = \theta^l \left(\frac{1}{T} \left\lfloor \frac{z^l_i T + \theta^l \varphi}{\theta^l} \right\rfloor - z^l_i \right), \tag{26}$$

$$\mathbf{e}_{\text{ANN}} = \theta^l \left(z^l_i - \frac{1}{L} \left\lfloor \frac{z^l_i L}{\theta^l} + \varphi \right\rfloor \right). \tag{27}$$

Under the assumption that $\mathbf{e}_{\text{SNN}}$ and $\mathbf{e}_{\text{ANN}}$ are independent, we can write the variance of $\mathbf{Err}_i^l$ as:

$$\text{Var}\left(\mathbf{Err}_i^l\right) = \text{Var}\left(\mathbf{e}_{\text{SNN}}\right) + \text{Var}\left(\mathbf{e}_{\text{ANN}}\right). \tag{28}$$

Since the expectation $\text{E}\left[\mathbf{e}_{\text{SNN}}\right] = 0$ and $\text{E}\left[\mathbf{e}_{\text{ANN}}\right] = 0$ as shown in (Bu et al., 2022), we can compute the variances by calculating the second moments:

$$\text{Var}\left(\mathbf{Err}_i^l\right) = \text{E}\left[\mathbf{e^2}_{\text{SNN}}\right] + \text{E}\left[\mathbf{e^2}_{\text{ANN}}\right]. \tag{29}$$

Consider a scenario in which a random variable x defined on the interval $[0, \theta]$ is uniformly distributed within each subinterval $[m_t, m_{t+1}]$ for $t = 0, 1, \ldots, T$. In each of these subintervals, the corresponding probability density is denoted by p_t. The subinterval boundaries are defined such that $m_0 = 0$ and $m_{T+1} = \theta$, with the intermediate points given by

$$m_t = \frac{\left(t - \frac{1}{2}\right)\theta}{T} \quad \text{for } t = 1, 2, \ldots, T.$$

Under this setup, one can determine the second moment of x as follows:

$$\mathbb{E}_x\left(x - \frac{\theta}{T}\left[\frac{Tx}{\theta} + \frac{1}{2}\right]\right)^2 = \frac{\theta^2}{12}\left(1 + \frac{1}{T}\right)$$

Proof for the second moment can be shown as:

$$\begin{aligned}
&\mathbb{E}_x\left(x - \frac{\theta}{T}\left[\frac{Tx}{\theta} + \frac{1}{2}\right]\right)^2 \\
&= \int_0^{\theta/2T} p_0\left(x - \frac{\theta}{T}\left[\frac{Tx}{\theta} + \frac{1}{2}\right]\right)^2 dx \\
&+ \sum_{t=1}^{T-1} \int_{(2t-1)\theta/2T}^{(2t+1)\theta/2T} p_t\left(x - \frac{\theta}{T}\left[\frac{Tx}{\theta} + \frac{1}{2}\right]\right)^2 dx \\
&+ \int_{(2T-1)\theta/2T}^{\theta} p_T\left(x - \frac{\theta}{T}\left[\frac{Tx}{\theta} + \frac{1}{2}\right]\right)^2 dx
\end{aligned}$$

Therefore, the integrals become:

$$\begin{aligned}
&= p_0 \int_0^{\theta/2T} x^2 dx + \sum_{t=1}^{T-1} p_t \int_{(2t-1)\theta/2T}^{(2t+1)\theta/2T} \left(x - \frac{t\theta}{T}\right)^2 dx \\
&+ p_T \int_{(2T-1)\theta/2T}^{\theta} (x - \theta)^2 dx \\
&= \left(\frac{\theta}{2T}\right)^3 \left[\frac{p_0}{3} + \sum_{t=1}^{T-1} \frac{2p_t}{3} + \frac{p_T}{3}\right]
\end{aligned}$$

Since $p_0 = p_T = \frac{2T}{\theta}$ and $p_t = \frac{T}{\theta}$ by considering uniform distribution in each interval, we get:

$$\begin{aligned}
\mathbb{E}_x\left(x - \frac{\theta}{T}\left[\frac{Tx}{\theta} + \frac{1}{2}\right]\right)^2 &= \left(\frac{\theta}{2T}\right)^2 \cdot \left(\frac{T+1}{3}\right) \\
&= \frac{\theta^2}{12}\left(\frac{1+T}{T^2}\right)
\end{aligned} \tag{30}$$

Similarly, we can write variance of $\mathbf{Err}_i^l$ as:

$$\text{Var}\left(\mathbf{Err}_i^l\right) = \frac{(\theta^{l^2})}{12}\left(\frac{1+T}{T^2}\right) + \frac{(\theta^l)^2}{12}\left(\frac{1+L}{L^2}\right) \tag{31}$$

A.2 Experiment Details

In our experiments, we utilized three well-known image classification datasets: CIFAR-10, CIFAR-100, and Tiny ImageNet.

CIFAR-10. The CIFAR-10 dataset consists of 60,000 color images with dimensions of 32×32 pixels. These images are equally divided into 10 different classes. We used 50,000 images for training the models and 10,000 images for testing their performance.

CIFAR-100. Similar to CIFAR-10 in terms of image size and total number of images, the CIFAR-100 dataset contains 60,000 color images of 32×32 pixels. However, these images are categorized into 100 classes, each containing 600 images. The dataset is split into 50,000 training images and 10,000 test images.

Tiny ImageNet. The Tiny ImageNet dataset is a subset of the ImageNet dataset. It comprises 110,000 color images with dimensions of 64×64 pixels, organized into 200 different classes. Each class has 500 training images, 50 validation images, and 50 test images. We utilized the 100,000 training images and evaluated model performance on the 10,000 test images.

We utilized the **Cutout** data augmentation technique (DeVries & Taylor, 2017) alongside standard augmentation methods to enhance model performance and mitigate overfitting. Our architecture incorporates the residual learning framework from (He et al., 2016), enabling stable training of deeper networks. For optimization, we used momentum-based SGD (batch size: 128) with L2 regularization (weight decay: 1×10^{-4}). The learning rate was initialized at 0.01 and automatically reduced by a factor of 10 at the 150^{th} and 180^{th} epochs during the 200-epoch training cycle. All implementations were developed in PyTorch and executed on **NVIDIA GeForce RTX 4090** GPUs.