NeuralFuse: Learning to Recover the Accuracy of Access-Limited Neural Network Inference in Low-Voltage Regimes

Hao-Lun Sun[1], **Lei Hsiung**[2], **Nandhini Chandramoorthy**[3], **Pin-Yu Chen**[3], **Tsung-Yi Ho**[4]

[1] National Tsing Hua University [2] Dartmouth College [3] IBM Research
[4] The Chinese University of Hong Kong
s109062594@m109.nthu.edu.tw
lei.hsiung.gr@dartmouth.edu
{pin-yu.chen, nandhini.chandramoorthy}@ibm.com
tyho@cse.cuhk.edu.hk

Abstract

Deep neural networks (DNNs) have become ubiquitous in machine learning, but their energy consumption remains problematically high. An effective strategy for reducing such consumption is supply-voltage reduction, but if done too aggressively, it can lead to accuracy degradation. This is due to random bit-flips in static random access memory (SRAM), where model parameters are stored. To address this challenge, we have developed NeuralFuse, a novel add-on module that handles the energy-accuracy tradeoff in low-voltage regimes by learning input transformations and using them to generate error-resistant data representations, thereby protecting DNN accuracy in both nominal and low-voltage scenarios. As well as being easy to implement, NeuralFuse can be readily applied to DNNs with limited access, such cloud-based APIs that are accessed remotely or non-configurable hardware. Our experimental results demonstrate that, at a 1% bit-error rate, NeuralFuse can reduce SRAM access energy by up to 24% while recovering accuracy by up to 57%. To the best of our knowledge, this is the first approach to addressing low-voltage-induced bit errors that requires no model retraining.

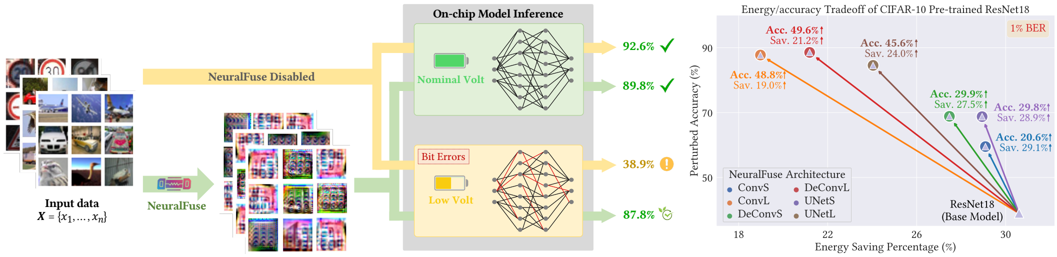
1 Introduction

Energy-efficient computing is of primary importance to the effective deployment of deep neural networks (DNNs), particularly in edge devices and in on-chip AI systems. Increasing DNN computation's energy efficiency and lowering its carbon footprint require iterative efforts from both chip designers and algorithm developers. Processors with specialized hardware accelerators for AI computing, capable of providing orders of magnitude better performance and energy efficiency for AI computation, are now ubiquitous. However, alongside reduced precision/quantization and architectural optimizations, endowing such systems with the capacity for low-voltage operation is a powerful lever for reducing their power consumption.

The computer engineering literature contains ample evidence of the effects of undervolting and low-voltage operation on accelerator memories that store weights and activations during computation. Aggressive scaling-down of static random access memory's (SRAM's) supply voltage to below the rated value saves power, thanks to the quadratic dependence of dynamic power on voltage. Crucially,

Project Page: https://trustsafeai-neuralfuse.static.hf.space
Code: https://github.com/IBM/NeuralFuse

38th Conference on Neural Information Processing Systems (NeurIPS 2024).



(a) The pipeline of the NeuralFuse framework at inference. (b) Energy/accuracy tradeoff example.

Figure 1: (a) At inference, NeuralFuse transforms input samples $\mathbf{x}$ into robust data representations. The *nominal* voltage allows models to work as expected, whereas at *low voltage*, one would encounter bit errors (e.g., 1%) that cause incorrect inferences. The percentages reflect the accuracy of a CIFAR-10 pre-trained ResNet18 with and without NeuralFuse in both those voltage cases. (b) On the same base model (ResNet18), we illustrate the energy/accuracy tradeoff of six NeuralFuse implementations. The x-axis represents the percentage reduction in dynamic-memory access energy at low-voltage settings (base model protected by NeuralFuse), as compared to the bit-error-free (nominal) voltage. The y-axis represents the perturbed accuracy (evaluated at low voltage) with a 1% bit-error rate.

however, it also leads to an exponential increase in bit failures. Memory bit flips cause errors in the stored weight and activation values [Chandramoorthy et al., 2019, Ganapathy et al., 2017], leading to catastrophic accuracy loss.

A recent wave of research has proposed numerous techniques for allowing low-voltage operation of DNN accelerators while preserving their accuracy. Most of these have been either hardware-based error-mitigation techniques or error-aware robust training of DNN models. On-chip error mitigation methods have significant performance and power overheads [Chandramoorthy et al., 2019, Reagen et al., 2016]. On the other hand, some have proposed to generate models that are robust to bit errors via a specific learning algorithm [Kim et al., 2018, Koppula et al., 2019, Stutz et al., 2021], thereby eliminating the need for on-chip error mitigation. However, error-aware robust training to find the optimal set of robust parameters for each model is time- and energy-intensive and may not be possible in all access-limited settings.

In this paper, therefore, we propose a novel model-agnostic approach: *NeuralFuse*. This proof-of-concept machine-learning module offers trainable input transformation parameterized by a relatively small DNN; and, by enhancing input's robustness, it mitigates bit errors caused by very low-voltage operation, thus serving the wider goal of more accurate inferencing. The pipeline of NeuralFuse is illustrated in Figure 1. To protect the deployed models from making wrong predictions under low-power conditions, NeuralFuse accepts scenarios under access-limited neural networks (e.g., non-configurable hardware or cloud-based APIs). Specifically, we consider two access-limited scenarios that are common in the real world: 1) *relaxed access*, in which 'black box' model details are unknown, but backpropagation through those models is possible; and 2) *restricted access*, in which the model details are unknown and backpropagation is disallowed. To enable it to deal with relaxed access, we trained NeuralFuse via backpropagation, and for restricted-access cases, we trained it on a white-box surrogate model. To the best of our knowledge, this is the first study that leverages a learning-based method to address random bit errors as a means of recovering accuracy in low-voltage and access-limited settings.

We summarize our **main contributions** as follows:

- We propose *NeuralFuse*, a novel learning-based input-transformation module aimed at enhancing the accuracy of DNNs that are subject to random bit errors caused by very low voltage operation. NeuralFuse is model-agnostic, i.e., operates on a plug-and-play basis at the data-input stage and does not require any re-training of deployed DNN models.
- We explore two practical limited-access scenarios for neural-network inference: relaxed access and restricted access. In the former setting, we use gradient-based methods for module training. In the latter one, we use a white-box surrogate model for training, which is highly transferable to other types of DNN architecture.
- We report the results of an extensive program of experiments with various combinations of DNN models (ResNet18, ResNet50, VGG11, VGG16, and VGG19), datasets (CIFAR-10, CIFAR-100,

GTSRB, and ImageNet-10), and NeuralFuse implementations of different architectures and sizes. These show that NeuralFuse can consistently increase the perturbed accuracy (accuracy evaluated under random bit errors in weights) by up to 57%, while simultaneously saving up to 24% of the energy normally required for SRAM access, based on our realistic characterization of bit-cell failures for a given memory array in a low-voltage regime inducing a $0.5\%/1\%$ bit-error rate.

- We demonstrate NeuralFuse's transferability (i.e., adaptability to unseen base models), versatility (i.e., ability to recover low-precision quantization loss), and competitiveness (i.e., state-of-the-art performance) in various scenarios, establishing it as a promising proof-of-concept for energy-efficient, resilient DNN inference.

2 Related Work and Background

Software-based Energy-saving Strategies. Various recent studies have proposed software-based methods of reducing computing's energy consumption. For instance, quantization techniques have been reported to reduce the precision required for storing model weights, and thus to decrease total memory storage [Gong et al., 2014, Rastegari et al., 2016, Wu et al., 2016]. On the other hand, Yang et al. [2017] - who proposed energy-aware pruning on each layer and fine-tuning of weights to maximize final accuracy - suggested several ways to reduce DNNs' energy consumption. For example, they devised the ECC framework, which compresses DNN models to meet a given energy constraint [Yang et al., 2019a], and a method of compressing such models via joint pruning and quantization [Yang et al., 2020]. It is also feasible, during DNN training, to treat energy constraints as an optimization problem and thereby reduce energy consumption while maximizing training accuracy [Yang et al., 2019b]. However, unlike ours, all these methods imply changing either model architectures or model weights.

Hardware-based Energy-saving Strategies. Prior studies have also explored ways of improving energy efficiency via specially designed hardware. Several of them have focused on the undervolting of DNN accelerators and proposed methods to maintain accuracy in the presence of bit errors. For instance, Reagen et al. [2016] proposed an SRAM fault-mitigation technique that rounds faulty weights to zero to avoid degradation of prediction accuracy. Srinivasan et al. [2016] recommended storing sensitive MSBs (most significant bits) in robust SRAM cells to preserve accuracy. Chandramoorthy et al. [2019] proposed dynamic supply-voltage boosting to improve the resilience of memory-access operations; and the learning-based approach proposed by Stutz et al. [2021] aims to find models that are robust to bit errors. The latter paper discusses several techniques for improving such robustness, notably quantization, weight-clipping, random bit-error training, and adversarial bit-error training. Its authors concluded from their experiments that a combination of quantization, weight-clipping, and adversarial bit-error training will yield excellent performance. However, they also admitted that the relevant training process was sensitive to hyperparameter settings, and hence, it might come with a challenging training procedure.

However, we suggest that all the methods mentioned above are difficult to implement and/or unsuitable for use in real-world access-limited settings. For example, the weights of DNN models packed on embedded systems may not be configurable or updatable, making model retraining (e.g., Stutz et al. [2021]) non-viable in that scenario. Moreover, DNN training is already a tedious and time-consuming task, so adding error-aware training to it may further increase its complexity and, in particular, make hyperparameter searches more challenging. Özdenizci and Legenstein [2022] also reported that error-aware training was ineffective for large DNNs with millions of bits. NeuralFuse obviates the need for model retraining via an add-on trainable input-transformation function parameterized by a relatively small secondary DNN.

SRAM Bit Errors in DNNs. Low voltage-induced memory bit-cell failures can cause bit-flips from 0 to 1 and vice versa. In practice, SRAM bit errors increase exponentially when the supply voltage is scaled below V_{min}, i.e., the minimum voltage required to avoid them. This phenomenon has been studied extensively in the prior literature, including work by Chandramoorthy et al. [2019] and Ganapathy et al. [2017]. The specific increases in bit errors as voltage scales down, in the case of an SRAM array of 512×64 bits with a 14nm technology node, is illustrated in Figure 2. The corresponding dynamic energy per SRAM read access, measured at each voltage at a constant frequency, is shown on the right-hand side of the figure. In this example, accessing the SRAM at

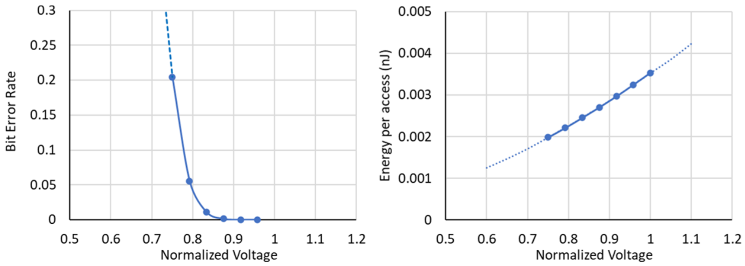


Figure 2: The bit-error rates (left) and dynamic energy per memory access versus voltage for static random access memory arrays (right) as reported by Chandramoorthy et al. [2019]. The x-axis shows voltages normalized with respect to the minimum bit error-free voltage (V_{min}).

$0.83V_{min}$ leads to a 1% bit-error rate, and at the same time, dynamic energy per access is reduced by approximately 30%. This can lead to DNNs making inaccurate inferences, particularly when bit-flips occur at the MSBs. However, improving robustness to bit errors can allow us to lower V_{min} and exploit the resulting energy savings.

It has been observed that bit-cell failures for a given memory array are randomly distributed and independent of each other. That is, the spatial distribution of bit-flips can be assumed to be random, as it generally differs from one array to another, within as well as between chips. Below, following Chandramoorthy et al. [2019], we model bit errors in a memory array of a given size by generating a random distribution of such errors with equal likelihood of 0-to-1 and 1-to-0 bit-flipping. More specifically, we assume that the model weights are quantized to 8-bit precision (i.e., from 32-bit floats to 8-bit integers), and generate perturbed models by injecting our randomly distributed bit errors into the two's complement representation of weights. For more implementation details, please refer to Section 4.1.

3 NeuralFuse: Framework and Algorithms

3.1 Error-Resistant Input Transformation

As illustrated in Figure 1, we propose a novel trainable input-transformation module, NeuralFuse, parametrized by a relatively small DNN, to mitigate the accuracy-energy tradeoff for model inference and thus overcome the drawback of performance degradation in low-voltage regimes. A specially designed loss function and training scheme are used to derive NeuralFuse and apply it to the input data such that the transformed inputs will become robust to low voltage-induced bit errors.

Consider the input $\mathbf{x}$ sampled from the data distribution $\mathcal{X}$ and a set of models $\mathcal{M}_p$ with $p\%$ random bit errors on weights (i.e., *perturbed* models). When it is not manifesting any bit errors (i.e., at normal-voltage settings), the perturbed model operates as a nominal deterministic one, denoted by M_0. NeuralFuse aims to ensure that a model $M_p \in \mathcal{M}_p$ can make correct inferences on the transformed inputs while also delivering consistent results in its M_0 state. To adapt to various data characteristics, NeuralFuse – designated as $\mathcal{F}$ in Eq. (1), below – is designed to be input-aware. This characteristic can be formally defined as

$$\mathcal{F}(\mathbf{x}) = \text{clip}_{[-1,1]}\big(\mathbf{x} + \mathcal{G}(\mathbf{x})\big), \tag{1}$$

where $\mathcal{G}(\mathbf{x})$ is a "generator" (i.e., an input-transformation function) that can generate a perturbation based on input $\mathbf{x}$. As transformed by NeuralFuse, i.e., as $\mathcal{F}(\mathbf{x})$, that input is passed to the deployed model (M_0 or M_p) for final inference. Without loss of generality, we assume the transformed input lies within a scaled input range $\mathcal{F}(\cdot) \in [-1, 1]^d$, where d is the (flattened) dimension of $\mathbf{x}$.

3.2 Training Objective and Optimizer

To train our generator $\mathcal{G}(\cdot)$, which ought to be able to ensure the correctness of both the perturbed model M_p and the clean model M_0, we parameterized it with a neural network and apply our training objective function

$$\arg\max_{\mathcal{W}_\mathcal{G}} \log \mathcal{P}_{M_0}(y|\mathcal{F}(\mathbf{x};\mathcal{W}_\mathcal{G})) + \lambda \cdot \mathbf{E}_{M_p \sim \mathcal{M}_p}[\log \mathcal{P}_{M_p}(y|\mathcal{F}(\mathbf{x};\mathcal{W}_\mathcal{G}))], \tag{2}$$

where $\mathcal{W}_\mathcal{G}$ is the set of trainable parameters for $\mathcal{G}$; y is the ground-truth label of $\mathbf{x}$; $\mathcal{P}_M$ denotes the likelihood of y as computed by a model M being given a transformed input $\mathcal{F}(\mathbf{x}; \mathcal{W}_\mathcal{G})$; $\mathcal{M}_p$ is the distribution of the perturbed models inherited from the clean model M_0, under a $p\%$ random bit-error rate; and λ is a hyperparameter that balances the importance of the nominal and perturbed models.

The training objective function can be readily converted to a loss function ($\mathcal{L}$) that evaluates cross-entropy between the ground-truth label y and the prediction $\mathcal{P}_M(y|\mathcal{F}(\mathbf{x}; \mathcal{W}_\mathcal{G})$. That is, the total loss function can be calculated as

$$\mathcal{L}_{\text{Total}} = \mathcal{L}_{M_0} + \lambda \cdot \mathcal{L}_{\mathcal{M}_p}. \tag{3}$$

In particular, optimizing the loss function requires evaluation of the impact of the loss term $\mathcal{L}_{\mathcal{M}_p}$ on randomly perturbed models. Our training process is inspired by expectation over transformation (EOT) attacks [Athalye et al., 2018], which aim to produce robust adversarial examples that are simultaneously adversarial over the entire transformation distribution. Based on that idea, we propose a new optimizer for solving Eq. (3), which we call expectation over perturbed models (EOPM). EOPM-trained generators can generate error-resistant input transformations and mitigate inherent bit errors. However, it would be computationally impossible to enumerate all possible perturbed models with random bit errors, and the number of realizations for perturbed models is constrained by the memory size of the GPUs used for training. In practice, therefore, we only use N perturbed models per iteration to calculate empirical average loss, i.e.,

$$\mathcal{L}_{\mathcal{M}_p} \approx \frac{\mathcal{L}_{M_{p_1}} + \cdots + \mathcal{L}_{M_{p_N}}}{N}, \tag{4}$$

where N is the number of perturbed models $\{M_{p_1}, \cdots, M_{p_N}\}$ that are simulated to calculate the loss caused by random bit errors. Therefore, the gradient used to update the generator can be calculated as follows:

$$\frac{\partial \mathcal{L}_{\text{Total}}}{\partial \mathcal{W}_\mathcal{G}} = \frac{\partial \mathcal{L}_{M_0}}{\partial \mathcal{W}_\mathcal{G}} + \frac{\lambda}{N} \left(\frac{\partial \mathcal{L}_{M_{p1}}}{\partial \mathcal{W}_\mathcal{G}} + \cdots + \frac{\partial \mathcal{L}_{M_{pN}}}{\partial \mathcal{W}_\mathcal{G}} \right). \tag{5}$$

Through our implementation, we found that stable performance could be delivered when $N = 10$, and that there was little to be gained by using a larger value. The results of our ablation study for different values of N can be found in Appendix E.

3.3 Training Algorithm

Algorithm 1 in Appendix A summarizes NeuralFuse's training steps. Briefly, this involves splitting the training data $\mathcal{X}$ into B mini-batches for training the generator in each epoch. For each mini-batch, we first feed these data into $\mathcal{F}(\cdot)$ to obtain the transformed inputs. Also, we simulate N perturbed models using a $p\%$ random bit-error rate, denoted by $M_{p_1}, \cdots, M_{p_N}$, from $\mathcal{M}_p$. Then, the transformed inputs are fed into those N perturbed models as well as into the clean model $\mathcal{M}_0$, and their respective losses and gradients are calculated. Finally, NeuralFuse parameters $\mathcal{W}_\mathcal{G}$ are updated based on the gradient obtained by EOPM.

4 Experiments

4.1 Experiment Setups

Datasets. We evaluate NeuralFuse on four different datasets: CIFAR-10 [Krizhevsky and Hinton, 2009], CIFAR-100 [Krizhevsky and Hinton, 2009], the German Traffic Sign Recognition Benchmark (GTSRB) [Stallkamp et al., 2012], and ImageNet-10 [Deng et al., 2009]. CIFAR-10 consists of 10 classes, with 50,000 training images and 10,000 testing images in total. Similarly, CIFAR-100 consists of 100 classes, with 500 training images and 100 testing images in each. The GTSRB contains 43 classes with a total of 39,209 training images and 12,630 testing images. Similar to CIFAR-10 and CIFAR-100, we resize GTSRB into 32×32×3 in our experiment. For ImageNet-10, we chose the same ten categories as Huang et al. [2022], in which there are 13,000 training images and 500 test images cropped into 224×224×3. Due to space limitations, our CIFAR-100 results are presented in Appendices F and G.

Base Models. We selected several common architectures for our base models: ResNet18, ResNet50 [He et al., 2016], VGG11, VGG16, and VGG19 [Simonyan and Zisserman, 2015]. To replicate the deployment of models on chips, all our based models were given quantization-aware training that followed Stutz et al. [2021].

NeuralFuse Generators. The architecture of the NeuralFuse generator ($\mathcal{G}$)) is based on an encoder-decoder structure. We designed and compared three types of generators, namely convolution-based, deconvolution-based, and UNet-based. We also considered large(L)/small(S) network sizes for each type. Further details can be found below and in Appendix B.

- **Convolution-based (Conv).** Conv uses convolution with MaxPool layers for its encoder and *convolution with UpSample layers* for its decoder. This architecture has previously been shown to be efficient and effective at generating *input-aware* backdoor triggers [Nguyen and Tran, 2020].
- **Deconvolution-based (DeConv).** DeConv uses convolution with MaxPool layers for its encoder and *deconvolution layers* for its decoder. We expected this modification both to enhance its performance and to reduce its energy consumption.
- **UNet-based (UNet).** UNet uses convolution with MaxPool layers for its encoder, and deconvolution layers for its decoder. UNet is known for its robust performance in image segmentation [Ronneberger et al., 2015].

Energy-consumption Calculation. The energy consumption reported in Figure 1 is based on the product of the total number of SRAM memory accesses in a systolic array-based convolution neural network (CNN) accelerator and the dynamic energy per read access at a given voltage. Research by Chen et al. [2016] previously showed that energy consumption by SRAM buffers and arrays accounts for a high proportion of total system energy consumption. We assume that there are no bit errors on NeuralFuse, given its status as an add-on data preprocessing module whose functions could also be performed by a general-purpose core. In this work we assume it is implemented on the accelerator equipped with dynamic voltage scaling and therefore NeuralFuse computation is performed at nominal error-free voltage. We report a reduction in overall weight-memory energy consumption (i.e., NeuralFuse + Base Model under a $p\%$ bit-error rate) with respect to the unprotected base model in the regular-voltage mode (i.e., 0% bit-error rate and without NeuralFuse).

To quantify memory accesses, we used the SCALE-SIM simulator [Samajdar et al., 2020], and our chosen configuration simulated an output-stationary dataflow and a 32×32 systolic array with 256KB of weight memory. We collected data on the dynamic energy per read access of the SRAM both at V_{min} and at the voltage corresponding to a 1% bit-error rate ($V_{ber} \approx 0.83V_{min}$) from Cadence ADE Spectre simulations, both at the same clock frequency.

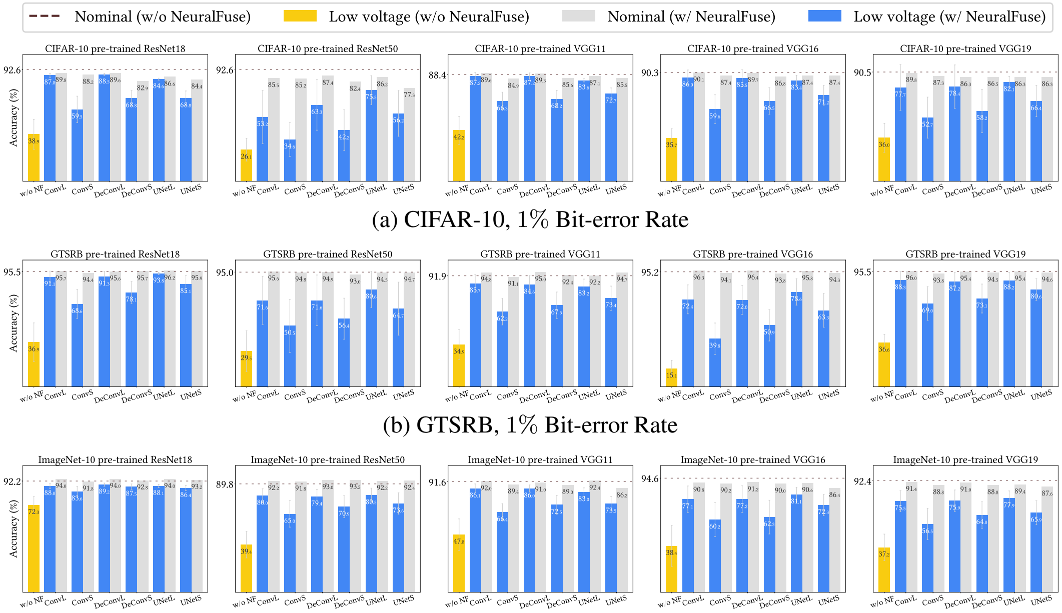
Relaxed and Restricted Access Settings. In the first of our experiments' two scenarios, relaxed access, the base-model information was not entirely transparent, but allowed us to obtain gradients from the black-box model through backpropagation. Therefore, this scenario allowed direct training of NeuralFuse with the base model using EOPM. In the restricted-access scenario, on the other hand, only the inference function was allowed for the base model, and we therefore trained NeuralFuse using a white-box surrogate base model and then transfering the generator to the access-restricted model.

Computing Resources. Our experiments were performed using eight Nvidia Tesla V100 GPUs and implemented with PyTorch. NeuralFuse was found to generally take 150 epochs to converge, and its training time was similar to that of the base model it incorporated. On both the CIFAR-10 and CIFAR-100 datasets, average training times were 17 hours (ResNet18), 50 hours (ResNet50), 9 hours (VGG11), 13 hours (VGG16), and 15 hours (VGG19). For GTSRB, the average training times were 9 hours (ResNet18), 27 hours (ResNet50), 5 hours (VGG11), 7 hours (VGG16), and 8 hours (VGG19); and for ImageNet-10, the average training times were 32 hours (ResNet18), 54 hours (ResNet50), 50 hours (VGG11), 90 hours (VGG16), and 102 hours (VGG19).

4.2 Performance Evaluation, Relaxed-access Scenario

Our experimental results pertaining to the relaxed-access scenario are shown in Figure 3. The bit-error rate (BER) due to low voltage was 1% in the cases of CIFAR-10 and GTSRB, and 0.5% for ImageNet-10. The BER of ImageNet-10 was lower than that of the other two because, being pre-trained, it has more parameters than either of them. For each experiment, we sampled and evaluated $N = 10$ perturbed models (independent from training), and below, we report the means and standard deviations of their respective accuracies. Below, clean accuracy (CA) refers to a model's accuracy measured at nominal voltage, and perturbed accuracy (PA) to its accuracy measured at low voltage.

In the cases of CIFAR-10 and the GTSRB, we observed that large generators like ConvL and UNetL recovered PA considerably, i.e., in the range of 41% to 63% on ResNet18, VGG11, VGG16,



(a) CIFAR-10, 1% Bit-error Rate

(b) GTSRB, 1% Bit-error Rate

(c) ImageNet-10, 0.5% Bit-error Rate

Figure 3: Relaxed-access scenario test accuracies (%) of various pre-trained models with and without NeuralFuse, compared at nominal voltage (0% bit-error rate) or low voltage (with specified bit-error rates). The results demonstrate that NeuralFuse consistently recovered perturbation accuracy.

and VGG19. ResNet50's recovery percentage was slightly worse than those of the other base models, but it nevertheless attained up to 51% recovery on the GTSRB. On the other hand, the recovery percentages achieved when we used small generators like DeConvS were worse than those of their larger counterparts. This could be explained by larger-sized networks' better ability to learn error-resistant generators (though perhaps at the cost of higher energy consumption). In the case of ImageNet-10, using larger generators also yielded better PA performance recovery, further demonstrating NeuralFuse's ability to work well with large input sizes and varied datasets.

4.3 Performance Evaluation, Restricted-access Scenario (Transferability)

The experimental results of our restricted-access scenario are shown in Table 1. We adopted ResNet18 and VGG19 as our white-box surrogate source models for training the generators under a 1.5% bit-error rate. We chose ConvL and UNetL as our generators because they performed best out of the six we tested (see Figure 3).

From Table 1, we can see that transferring from a larger BER such as 1.5% can endow a smaller one (e.g., 1%) with strong resilience. The table also makes it clear that using VGG19 as a surrogate model with UNet-based generators like UNetL can yield better recovery performance than other model/generator combinations. On the other hand, we observed in some cases that if we transferred between source and target models of the same type (but with different BERs for training and testing), performance results could exceed those we had obtained during the original relaxed-access scenario. For instance, when we transferred VGG19 with UNetL under a 1.5% BER to VGG19 or VGG11 under a 0.5% BER, the resulting accuracies were 85.86% (as against 84.99% for original VGG19) and 84.81% (as against 82.42% for original VGG11). We conjecture that generators trained on relatively large BERs can cover the error patterns of smaller BERs, and even help improve the latter's generalization. These findings indicate the considerable promise of access-limited base models in low-voltage settings to recover accuracy.

4.4 Energy/Accuracy Tradeoff

We report total dynamic energy consumption as the total number of SRAM-access events multiplied by the dynamic energy of a single such event. Specifically, we used SCALE-SIM to calculate total

Table 1: Restricted-access scenario: Transfer results on CIFAR-10 with 1.5% bit-error rate

SM	TM	BER	CA	PA	ConvL (1.5%)			UNetL (1.5%)		
					CA (NF)	PA (NF)	RP	CA (NF)	PA (NF)	RP
ResNet18	ResNet18	1%	92.6	38.9 ± 12.4	89.8	89.0 ± 0.5	50.1	85.8	85.2 ± 0.5	46.3
		0.5%		70.1 ± 11.6		89.6 ± 0.2	19.5		85.7 ± 0.2	15.6
	ResNet50	1%	92.6	26.1 ± 9.4	89.2	36.1 ± 18	10.0	84.4	38.9 ± 16	12.8
		0.5%		61.0 ± 10.3		74.1 ± 10	13.1		72.7 ± 4.6	11.7
	VGG11	1%	88.4	42.2 ± 11.6	86.3	59.2 ± 10	17.0	82.3	69.8 ± 7.5	27.6
		0.5%		63.6 ± 9.3		78.9 ± 4.9	15.3		77.0 ± 4.0	13.4
	VGG16	1%	90.3	35.7 ± 7.9	89.4	62.2 ± 18	26.5	84.7	68.9 ± 14	33.2
		0.5%		66.6 ± 8.1		83.4 ± 5.5	16.8		80.5 ± 5.9	13.9
	VGG19	1%	90.5	36.0 ± 12.0	89.8	49.9 ± 23	13.9	85.0	55.1 ± 17	19.1
		0.5%		64.2 ± 12.4		81.8 ± 8.5	17.6		78.5 ± 6.8	14.3
VGG19	ResNet18	1%	92.6	38.9 ± 12.4	88.9	62.6 ± 13	23.7	85.0	72.3 ± 11	33.4
		0.5%		70.1 ± 11.6		84.2 ± 7.2	14.1		82.1 ± 2.2	12.0
	ResNet50	1%	92.6	26.1 ± 9.4	88.8	37.9 ± 18	11.8	85.2	46.7 ± 17	20.6
		0.5%		61.0 ± 10.3		76.6 ± 7.8	15.6		78.3 ± 3.7	17.3
	VGG11	1%	88.4	42.2 ± 11.6	88.9	76.0 ± 6.1	33.8	85.5	81.9 ± 3.9	39.7
		0.5%		63.6 ± 9.3		85.9 ± 2.6	22.3		84.8 ± 0.5	21.2
	VGG16	1%	90.3	35.7 ± 7.9	89.0	76.5 ± 9.0	40.8	85.9	79.2 ± 7.5	43.5
		0.5%		66.6 ± 8.1		87.7 ± 0.7	21.1		84.7 ± 0.9	18.1
	VGG19	1%	90.5	36.0 ± 12.0	89.1	80.2 ± 12	44.2	86.3	84.3 ± 1.2	48.3
		0.5%		64.2 ± 12.4		88.8 ± 0.4	24.6		85.9 ± 0.3	21.7

Note. SM: source model, used for training generators; TM: target model, used for testing generators; BER: the bit-error rate of the target model; CA (%): clean accuracy; PA (%): perturbed accuracy; NF: NeuralFuse; and RP: total recovery percentage of PA (NF) vs. PA

Table 2: Energy saving (%) by NeuralFuse for 30 combinations of base models and generators.

	ConvL	ConvS	DeConvL	DeConvS	UNetL	UNetS
ResNet18	19.0	29.1	21.2	27.5	24.0	28.9
ResNet50	25.4	29.9	26.4	29.2	27.7	29.9
VGG11	6.6	27.5	11.2	24.1	17.1	27.2
VGG16	17.1	28.9	19.7	27.0	23.0	28.7
VGG19	20.3	29.7	22.3	27.8	24.8	29.1

weight-memory access (TWMA), specifics of which can be found in Appendix C's Table 6. In Table 2, below, we report the percentages of energy saved (ES) at voltages that yield a 1% bit-error rate for various base-model and generator combinations. The formula for computing ES is

$$\text{ES} = \frac{\text{Energy}_{NV} - \left(\text{Energy}_{LV} + \text{Energy}_{\text{NeuralFuse at } NV}\right)}{\text{Energy}_{NV}} \times 100\%, \tag{6}$$

where *NV* denotes nominal voltage regime, and *LV*, a low-voltage one.

Our results indicate that when ResNet18 is utilized as a base model, NeuralFuse can recover model accuracy by $20 - 49\%$ while reducing energy use by $19 - 29\%$. In Appendix C, we provide more results on the tradeoff between energy and accuracy of different NeuralFuse and base-model combinations. Overall, it would appear that using NeuralFuse can effectively restore model accuracy when SRAM encounters low-voltage-induced random bit errors.

Runtime and Latency. On the other hand, runtime and its corresponding energy consumption may also affect overall energy savings. For instance, previous research has shown that multiply-and-accumulate (MAC) operations account for more than 99% of all operations in state-of-the-art CNNs, dominating processing runtime and energy consumption alike Yang et al. [2017]. Therefore, we also report the results of our MAC-based energy-consumption estimation in Appendix C, and of our latency analysis in Appendix D. Here, it should also be noted that an additional latency overhead is an inevitable tradeoff for reducing energy consumption in our scenarios. Although neither runtime nor latency is a major focus of this paper, future researchers could usefully design a lighter-weight version of the NeuralFuse module, or apply model-compression techniques to it, to reduce these two factors.

4.5 Model Size and NeuralFuse Efficiency

To arrive at a full performance characterization of NeuralFuse, we analyzed the relationship between the final recovery achieved by each base model in combination with generators of varying parameter counts. For this purpose, we defined *efficiency ratio* as the recovery percentage in PA divided by NeuralFuse's parameter count. Table 3 compares the efficiency ratios of all NeuralFuse generators

Table 3: The efficiency ratio for all NeuralFuse generators.

Base Model	BER	NeuralFuse					
		ConvL	ConvS	DeConvL	DeConvS	UNetL	UNetS
ResNet18	1%	67.5	182	76.6	190.7	94.5	245.9
	0.5%	24.7	73.3	30.7	62.5	33.6	88.3
ResNet50	1%	37.4	75.1	57.4	102.7	102.3	248.4
	0.5%	35.2	108.7	40.4	92.5	47.4	124.6
VGG11	1%	62.3	212.9	69.5	165.8	92.0	251.7
	0.5%	32.3	96.3	35.8	77.2	38.9	100.7
VGG16	1%	69.6	211.2	76.9	196.5	98.8	292.9
	0.5%	30.3	98.1	33.0	75.3	40.6	113
VGG19	1%	57.6	147.5	65.5	141.6	95.4	250.8
	0.5%	33.0	91.0	37.5	70.2	43.1	106.4

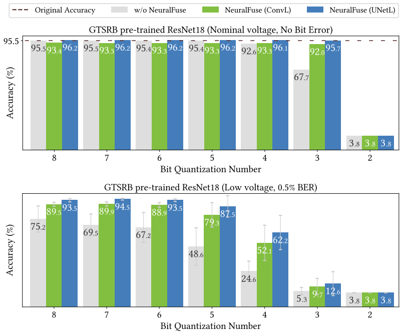


Figure 4: Reduced-precision accuracy

trained on CIFAR-10. Those results show that UNet-based generators had better efficiency per million parameters than either convolution-based or deconvolution-based ones.

4.6 NeuralFuse's Robustness to Reduced-precision Quantization

Lastly, we also explored NeuralFuse's robustness to low-precision quantization on model weights. Uniform quantization is the usual method for quantizing model weights [Gholami et al., 2022]. However, it is possible for it to cause an accuracy drop due to lack of precision. Given our aim of offering protection from bit errors, we hoped to understand whether NeuralFuse could also recover model-accuracy drops caused by this phenomenon. We therefore uniformly quantized the model weights to a lower bit precision and measured the resulting accuracy. Specifically, we applied symmetric uniform quantization to our base models with various numbers of bits to induce precision loss, and defined the quantized weight $\mathbf{W_q}$ (integer) as $\mathbf{W_q} = \lfloor \frac{\mathbf{W}}{\mathbf{s}} \rceil$, where $\mathbf{W}$ denotes the original model weight (full precision), $s = \frac{\max|\mathbf{W}|}{2^{b-1}-1}$ is the quantization scale parameter, and b is the precision (number of bits) used to quantize the models. Bit errors induced by low voltage operation as previously described, are also applied to low precision weights.

We used the GTSRB pre-trained ResNet18 as our example in an evaluation of two NeuralFuse generators, i.e., ConvL and UNetL trained with 0.5% BER, and varied precision b from 8 bits to 2 bits (integer). The results, shown in Figure 4, indicated that when $b > 3$ bits, NeuralFuse could effectively recover accuracy in both the low-voltage and low-precision scenarios. When $b = 3$, while NeuralFuse could still handle the bit-error-free model (Fig. 4 top), it exhibited a limited ability to recover the random bit-error case (Fig. 4 bottom). We find these results encouraging, insofar as NeuralFuse was only trained on random bit errors, yet demonstrated high accuracy in dealing with unseen bit-quantization errors. Further experimental results derived from other base models and datasets can be found in Appendix H.

4.7 Extended Analysis

Here, we would like to highlight some key findings from the additional results in the Appendices. In Appendix E, we compare NeuralFuse against a simple baseline of learning a universal input perturbation. We found that such baseline performed much worse than NeuralFuse at that task, validating the necessity of adopting input-aware transformation if the goal is to learn error-resistant data representations in low-voltage scenarios. In Appendix G, we report that ensemble training of white-box surrogate base models could further improve the transferability of NeuralFuse in restricted-access scenarios. Appendices K and L present visualization results of NeuralFuse's data embeddings and transformed inputs. In Appendix J, we show that NeuralFuse can further recover the accuracy of a base model trained with adversarial weight perturbation in a low-voltage setting.

5 Conclusion

In this paper, we have proposed NeuralFuse, the first non-intrusive post hoc module that protects model inference against bit errors induced by low voltage. NeuralFuse is particularly well suited to practical machine-deployment cases in which access to the base model is limited or relaxed. The design of NeuralFuse includes a novel loss function and a new optimizer, EOPM, that enable it to handle simulated randomness in perturbed models. Our comprehensive experimental results and analysis show that NeuralFuse can recover test accuracy by up to 57% while simultaneously enjoying an up to 24% reduction in memory-access energy requirements. Moreover, NeuralFuse demonstrates high transferability to access-constrained models and high versatility, e.g., robustness to low-precision quantization. In short, NeuralFuse is a notable advancement in mitigating neural-network inference's energy/accuracy tradeoff in low-voltage regimes, and points the way to greener future AI technology. Our future work will include extending this study to other neural-network architectures and modalities, such as transformer-based language models.

Limitations. We acknowledge the challenge of achieving significant power savings without accuracy loss and view NeuralFuse as a foundational, proof-of-concept step toward this goal. Future research could enhance this approach by optimizing the pre-processing module to adapt to specific error characteristics of low-voltage SRAM or by integrating lightweight hardware modifications to further improve the energy-accuracy trade-off.

Broader Impacts. We see no ethical or immediate negative societal consequence of our work, and it holds the potential for positive social impacts, from environmental benefits to improved access to technology and enhanced safety in critical applications.

Acknowledgments and Disclosure of Funding

We thank the anonymous reviewers for their insightful comments and valuable suggestions. The research described in this paper was conducted in the JC STEM Lab of Intelligent Design Automation, which is funded by The Hong Kong Jockey Club Charities Trust in support of Tsung-Yi Ho.

References

Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 284–293, Stockholm, Sweden, 10–15 Jul 2018. PMLR.

Babak Ehteshami Bejnordi, Tijmen Blankevoort, and Max Welling. Batch-shaping for learning conditional channel gated networks. In *International Conference on Learning Representations*, 2020.

Nandhini Chandramoorthy, Karthik Swaminathan, Martin Cochet, Arun Paidimarri, Schuyler Eldridge, Rajiv V. Joshi, Matthew M. Ziegler, Alper Buyuktosunoglu, and Pradip Bose. Resilient low voltage accelerators for high energy efficiency. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 147–158, 2019. doi: 10.1109/HPCA.2019.00034.

Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH computer architecture news*, 44(3):367–379, 2016.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Shrikanth Ganapathy, John Kalamatianos, Keith Kasprak, and Steven Raasch. On characterizing near-threshold sram failures in finfet technology. In *Proceedings of the 54th Annual Design Automation Conference 2017*, pages 1–6, 2017.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC, 2022.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Zhizhong Huang, Jie Chen, Junping Zhang, and Hongming Shan. Learning representation for clustering via prototype scattering and positive sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–16, 2022. doi: 10.1109/TPAMI.2022.3216454.

Sung Kim, Patrick Howe, Thierry Moreau, Armin Alaghi, Luis Ceze, and Visvesh Sathe. Matic: Learning around errors for efficient low-voltage neural network accelerators. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2018.

Skanda Koppula, Lois Orosa, A Giray Yağlıkçı, Roknoddin Azizi, Taha Shahroodi, Konstantinos Kanellopoulos, and Onur Mutlu. Eden: Enabling energy-efficient, high-performance deep neural network inference using approximate dram. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 166–181, 2019.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, Toronto, Ontario, 2009.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1765–1773, 2017.

Tuan Anh Nguyen and Anh Tran. Input-aware dynamic backdoor attack. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:3454–3464, 2020.

Ozan Özdenizci and Robert Legenstein. Improving robustness against stealthy weight bit-flip attacks by output code matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13388–13397, 2022.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision (ECCV)*, pages 525–542. Springer, 2016.

Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, José Miguel Hernández-Lobato, Gu-Yeon Wei, and David Brooks. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 267–278, 2016. doi: 10.1109/ISCA.2016.32.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. A systematic methodology for characterizing scalability of dnn accelerators using scale-sim. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 58–68, 2020.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

Vladislav Sovrasov. ptflops: a flops counting tool for neural networks in pytorch framework, 2018-2023. URL https://github.com/sovrasov/flops-counter.pytorch. Github Repository.

Gopalakrishnan Srinivasan, Parami Wijesinghe, Syed Shakib Sarwar, Akhilesh Jaiswal, and Kaushik Roy. Significance driven hybrid 8t-6t sram for energy-efficient synaptic storage in artificial neural networks. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 151–156. IEEE, 2016.

Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012.

David Stutz, Nandhini Chandramoorthy, Matthias Hein, and Bernt Schiele. Bit error robustness for energy-efficient dnn accelerators. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems*, volume 3, pages 569–598, 2021.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

Dongxian Wu, Shu-Tao Xia, and Yisen Wang. Adversarial weight perturbation helps robust generalization. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 2958–2969. Curran Associates, Inc., 2020.

Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4820–4828, 2016.

Haichuan Yang, Yuhao Zhu, and Ji Liu. Ecc: Platform-independent energy-constrained deep neural network compression via a bilinear regression model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11206–11215, 2019a.

Haichuan Yang, Yuhao Zhu, and Ji Liu. Energy-constrained compression for deep neural networks via weighted sparse projection and layer input masking. In *International Conference on Learning Representations (ICLR)*, 2019b.

Haichuan Yang, Shupeng Gui, Yuhao Zhu, and Ji Liu. Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization-based approach. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5687–5695, 2017.

Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. Theoretically principled trade-off between robustness and accuracy. In *International Conference on Machine Learning (ICML)*, pages 7472–7482. PMLR, 2019.

Appendix

The appendix provides more implementation details for our method, experimental results on more datasets and settings, ablation studies, and qualitative analysis. The appendices cover the following:

- **Implementation Details:** NeuralFuse Training Algorithm (Sec. A), NeuralFuse Generator (Sec. B), Energy/Accuracy Tradeoff (Sec. C), Latency Reports (Sec. D)
- **Experimental Results:** Ablation Studies (Sec. E), Relaxed Access (Sec. F), Restricted Access (Sec. G), Reduced-precision Quantization (Sec. H), Adversarial Training (Sec. I), Adversarial Weight Perturbation (Sec. J)
- **Qualitative Studies:** Data Embeddings Visualization (Sec. K), Transformed Inputs Visualization (Sec. L)

A Training Algorithm of NeuralFuse

Algorithm 1 Training steps for NeuralFuse

Input: Base model M_0; Generator $\mathcal{G}$; Training data samples $\mathcal{X}$; Distribution of the perturbed models $\mathcal{M}_p$; Number of perturbed models N; Total training iterations T
Output: Optimized parameters $\mathcal{W}_\mathcal{G}$ for the Generator $\mathcal{G}$

```
for $t = 0, ..., T-1$ do
    for all mini-batches $\{\mathbf{x}, y\}_{b=1}^{B} \sim \mathcal{X}$ do
        Create transformed inputs $\mathbf{x}_t = \mathcal{F}(\mathbf{x}) = \text{clip}_{[-1,1]}\big(\mathbf{x} + \mathcal{G}(\mathbf{x})\big)$.
        Sample $N$ perturbed models $\{M_{p_1}, ..., M_{p_N}\}$ from $\mathcal{M}_p$ under $p\%$ random bit errors.
        for all $M_{p_i} \sim \{M_{p_1}, ..., M_{p_N}\}$ do
            Calculate the loss $\mathcal{L}_{p_i}$ based on the output of the perturbed model $M_{p_i}$. Then calculate
            the gradients $g_{p_i}$ for $\mathcal{W}_\mathcal{G}$ based on $\mathcal{L}_{p_i}$.
        end for
        Calculate the loss $\mathcal{L}_0$ based on the output of the clean model $M_0$. Then calculate the
        gradients $g_0$ for $\mathcal{W}_\mathcal{G}$ based on $\mathcal{L}_0$.
        Calculate the final gradient $g_{final}$ using (5) based on $g_0$ and $g_{p_1}, ..., g_{p_N}$.
        Update $\mathcal{W}_\mathcal{G}$ using $g_{final}$.
    end for
end for
```

B Implementation Details of NeuralFuse Generator

We consider two main goals in designing the NeuralFuse Generator: 1) efficiency (so the overall energy overhead is decreased) and 2) robustness (so that it can generate robust patterns on the input image and overcome the random bit flipping in subsequent models). Accordingly, we choose to utilize an encode-decoder architecture in implementing the generator. The design of ConvL is inspired by Nguyen and Tran [2020], in which the authors utilize a similar architecture to design an input-aware trigger generator, and have demonstrated its efficiency and effectiveness. Furthermore, we attempted to enhance it by replacing the *Upsampling* layer with a *Deconvolution* layer, leading to the creation of DeConvL. The UNetL-based NeuralFuse draws inspiration from Ronneberger et al. [2015], known for its robust performance in image segmentation, and thus, we incorporated it as one of our architectures. Lastly, ConvS, DeConvS, and UNetS are *scaled-down* versions of the model designed to reduce computational costs and total parameters. The architectures of Convolutional-based and Deconvolutional-based are shown in Table 4, and the architecture of UNet-based generators is in Table 5. For the abbreviation used in the table, ConvBlock means the Convolution block, Conv means a single Convolution layer, DeConvBlock means the Deconvolution block, DeConv means a single Deconvolution layer, and BN means a Batch Normalization layer. We use learning rate $= 0.001$, $\lambda = 5$, and Adam optimizer. For CIFAR-10, GTSRB, and CIFAR-100, we set batch size $b = 25$ for each base model. For ImageNet-10, we set $b = 64$ for ResNet18, ResNet50 and VGG11, and $b = 32$ for both VGG16 and VGG19.

Table 4: Model architecture for both Convolution-based and Deconvolution-based generators. Each ConvBlock consists of a Convolution (kernel $= 3 \times 3$, padding $= 1$, stride $= 1$), a Batch Normalization, and a ReLU layer. Each DeConvBlock consists of a Deconvolution (kernel $= 4 \times 4$, padding $= 1$, stride $= 2$), a Batch Normalization, and a ReLU layer.

ConvL		ConvS		DeConvL		DeConvS	
Layers	#CHs	Layers	#CHs	Layers	#CHs	Layers	#CHs
(ConvBlock)×2, MaxPool	32	ConvBlock, Maxpool	32	(ConvBlock)×2, MaxPool	32	ConvBlock, Maxpool	32
(ConvBlock)×2, MaxPool	64	ConvBlock, Maxpool	64	(ConvBlock)×2, MaxPool	64	ConvBlock, Maxpool	64
(ConvBlock)×2, MaxPool	128	ConvBlock, Maxpool	64	(ConvBlock)×2, MaxPool,	128	ConvBlock, Maxpool	64
ConvBlock, UpSample, ConvBlock	128	ConvBlock, UpSample	64	ConvBlock	128	DeConvBlock	64
ConvBlock, UpSample, ConvBlock	64	ConvBlock, UpSample	32	DeConvBlock, ConvBlock	64	DeConvBlock	32
ConvBlock, UpSample, ConvBlock	32	ConvBlock, UpSample	3	DeConvBlock, ConvBlock	32	DeConv, BN, Tanh	3
Conv, BN, Tanh	32	Conv, BN, Tanh	3	Conv, BN, Tanh	3		

[Note] #CHs: *number of channels*.

Table 5: Model architecture for UNet-based generators. Each ConvBlock consists of a Convolution (kernel $= 3 \times 3$, padding $= 1$, stride $= 1$), a Batch Normalization, and a ReLU layer. Other layers, such as the Deconvolutional layer (kernel $= 2 \times 2$, padding $= 1$, stride $= 2$), are used in UNet-based models. For the final Convolution layer, the kernel size is set to 1.

UNetL		UNetS	
Layers	#Channels	Layers	#Channels
L1: (ConvBlock)×2	16	L1: (ConvBlock)×2	8
L2: Maxpool, (ConvBlock)×2	32	L2: Maxpool, (ConvBlock)×2	16
L3: Maxpool, (ConvBlock)×2	64	L3: Maxpool, (ConvBlock)×2	32
L4: Maxpool, (ConvBlock)×2	128	L4: Maxpool, (ConvBlock)×2	64
L5: DeConv	64	L5: DeConv	32
L6: Concat[L3, L5]	128	L6: Concat[L3, L5]	64
L7: (ConvBlock)×2	64	L7: (ConvBlock)×2	32
L8: DeConv	32	L8: DeConv	16
L9: Concat[L2, L8]	64	L9: Concat[L2, L8]	32
L10: (ConvBlock)×2	32	L10: (ConvBlock)×2	16
L11: DeConv	16	L11: DeConv	8
L12: Concat[L1, L11]	32	L12: Concat[L1, L11]	16
L13: (ConvBlock)×2	16	L13: (ConvBlock)×2	8
L14: Conv	3	L14: Conv	3

C NeuralFuse's Energy/Accuracy Tradeoff

SCALE-SIM. SCALE-SIM [Samajdar et al., 2020] is a systolic array-based CNN simulator that can calculate the number of memory accesses and the total time in execution cycles by giving the specific model architecture and accelerator architectural configuration as inputs. In this paper, we use SCALE-SIM to calculate the weights memory access of 5 based models (ResNet18, ResNet50, VGG11, VGG16, VGG19), and 6 generators (ConvL, ConvS, DeConvL, DeConvS, UNetL, UNetS). While SCALE-SIM supports both Convolutional and Linear layers, it does not yet support Deconvolution layers. Instead, we try to approximate the memory costs of Deconvolution layers by Convolution layers. We change the input and output from Deconvolution into the output and input of the Convolution layers. Besides, we also change the stride into 1 when we approximate it. We also add padding for the convolution layers while generating input files for SCALE-SIM. In this paper, we only consider the energy saving on weights accesses, so we only take the value "SRAM Filter Reads" from the output of SCALE-SIM as the *total weights memory accesses* (T.W.M.A.) for further energy calculation.

In Table 6, we report the total weight memory access (T.W.M.A.) using SCALE-SIM. We then showed the energy/accuracy tradeoff between all of the combinations of NeuralFuse and base models under a 1% of bit error rate in Figure 5.

Parameters and MACs Calculation. In addition to T.W.M.A., the number of parameters and MACs (multiply–accumulate operations) are also common metrics in measuring the energy consumption of machine learning models. Yang et al. [2017] have shown that the *energy consumption of computation*

Table 6: The total weights memory access calculated by SCALE-SIM.

Base Model	ResNet18	ResNet50	VGG11	VGG16	VGG19	-
T.W.M.A.	2,755,968	6,182,144	1,334,656	2,366,848	3,104,128	-
NeuralFuse	ConvL	ConvS	DeConvL	DeConvS	UNetL	UNetS
T.W.M.A.	320,256	41,508	259,264	86,208	180,894	45,711

[Note] T.W.M.A.: *total weight memory access*.

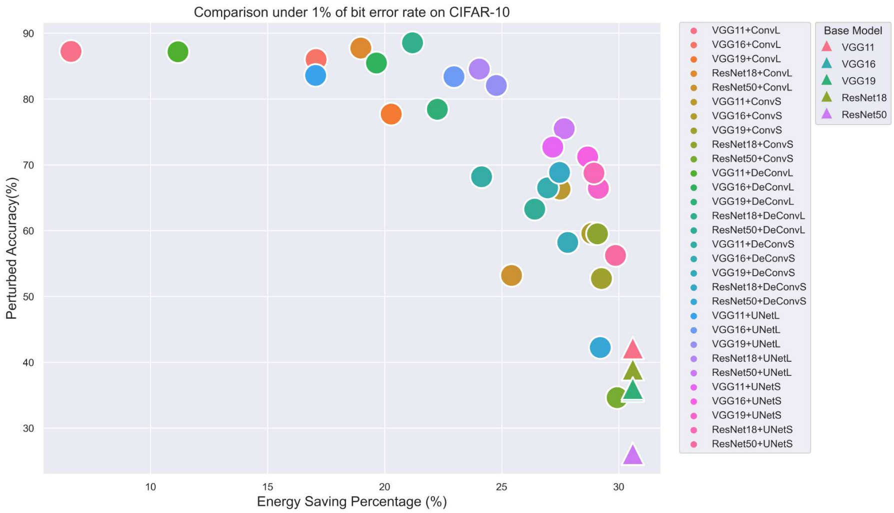


Figure 5: The energy/accuracy tradeoff of different NeuralFuse implementations with all CIFAR-10 pre-trained based models. The X-axis represents the percentage reduction in dynamic memory access energy at low-voltage settings (base model protected by NeuralFuse) compared to the bit-error-free (nominal) voltage; the Y-axis represents the perturbed accuracy (evaluated at low voltage) with a 1% bit error rate.

and *memory accesses* are both proportional to MACs, allowing us to take computation energy consumption into account.

Here, we use the open-source package `ptflops` [Sovrasov, 2018-2023] to calculate the parameters and MAC values of all the base models and the NeuralFuse generators, in the same units as Bejnordi et al. [2020] used. The results are shown in Table 7. Note that we modified the base model architectures for ImageNet-10, as it has larger input sizes. For example, we use a larger kernel size = 7 instead of 3 in the first Convolution layer in ResNet-based models to enhance the learning abilities. Therefore, the parameters of base models are different between different datasets. For NeuralFuse generators, we utilize the same architectures for implementation (including ImageNet-10). As a result, our proposed NeuralFuse generators are generally smaller than base models, either on total model parameters or MAC values.

Table 7: Parameter counts and MACs for all base models and generators in this paper.

		Base Model					
		ResNet18	ResNet50	VGG11	VGG16	VGG19	-
Parameter	CIFAR-10	11,173,962	23,520,842	9,231,114	14,728,266	20,040,522	-
	ImageNet-10	11,181,642	23,528,522	128,812,810	134,309,962	139,622,218	
MACs	CIFAR-10	557.14M	1.31G	153.5M	314.43M	399.47M	-
	ImageNet-10	1.82G	4.12G	7.64G	15.53G	19.69G	
		NeuralFuse					
		ConvL	ConvS	DeConvL	DeConvS	UNetL	UNetS
Parameter	CIFAR-10 ImageNet-10	723,273	113,187	647,785	156,777	482,771	121,195
MACs	CIFAR-10	80.5M	10.34M	64.69M	22.44M	41.41M	10.58M
	ImageNet-10	3.94G	506.78M	3.17G	1.1G	2.03G	518.47M

MACs-based Energy Consumption. We can then use the MAC values to further approximate the end-to-end energy consumption of the whole model. Assume that all values are stored on SRAM and that a MAC represents single memory access. The corresponding MACs-based energy saving percentage (MAC-ES, %) can be derived from Eq. 7 (c.f. Sec. 4.4), and results can be found in Table 8. We can observe that most combinations can save a large amount of energy, except that VGG11 with two larger NeuralFuse (ConvL and DeConvL) may increase the total energy. These results are consistent with the results reported in Table 2. In addition, we also showed the MACs-based energy/accuracy tradeoff between all of the combinations of NeuralFuse and base models under a 1% of bit error rate in Figure 6.

$$\text{MAC-ES} = \frac{\text{MACs}_{\text{base model}} \cdot \text{Energy}_{\text{nominal voltage}} - \left(\text{MACs}_{\text{base model}} \cdot \text{Energy}_{\text{low-voltage-regime}} + \text{MACs}_{\text{NeuralFuse}} \cdot \text{Energy}_{\text{NeuralFuse at nominal voltage}}\right)}{\text{MACs}_{\text{base model}} \cdot \text{Energy}_{\text{nominal voltage}}} \times 100\% \tag{7}$$

Table 8: The MACs-Based energy saving percentage (%) for different combinations of base models and NeuralFuse.

	ConvL	ConvS	DeConvL	DeConvS	UNetL	UNetS
ResNet18	16.2	28.7	19.0	26.6	23.2	28.7
ResNet50	24.5	29.8	25.7	28.9	27.4	29.8
VGG11	-21.8	23.9	-11.5	16.0	3.6	23.7
VGG16	5.0	27.3	10.0	23.5	17.4	27.2
VGG19	10.4	28.0	14.4	25.0	20.2	28.0

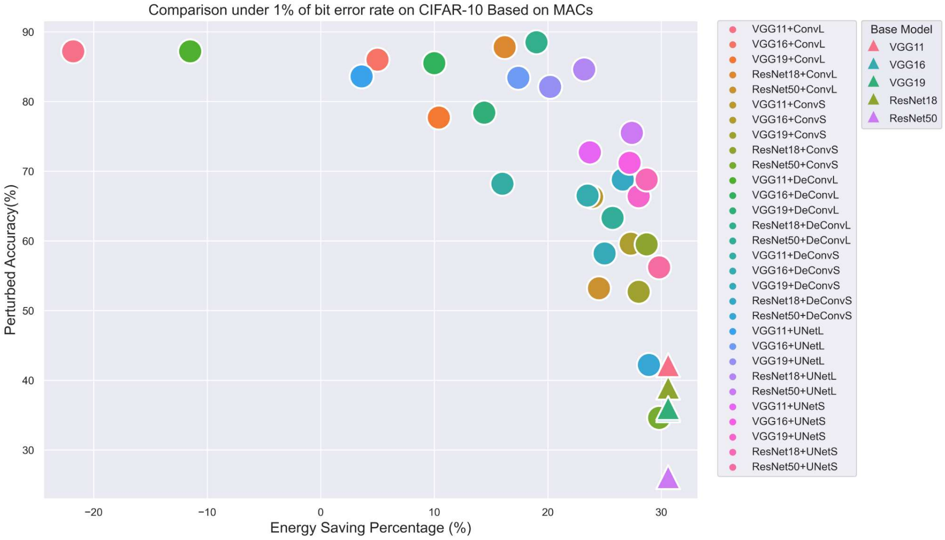


Figure 6: The Mac-based energy/accuracy tradeoff of different NeuralFuse implementations with all CIFAR-10 pre-trained based models. The X-axis represents the percentage reduction in dynamic memory access energy at low-voltage settings (base model protected by NeuralFuse), compared to the bit-error-free (nominal) voltage; the Y-axis represents the perturbed accuracy (evaluated at low voltage) with a 1% bit error rate.

Although using ConvL or DeConvL along with base model VGG11 for CIFAR-10 implies an increase in energy consumption, using other smaller-scale generators, we can still save the overall energy and recover the base model's accuracy. That said, developers can always choose smaller generators (with orders of magnitude fewer MAC operations than the original network) to restore model accuracy, further demonstrating the flexibility of choosing NeuralFuse generators of different sizes.

D Inference Latency of NeuralFuse

In Table 9, we report the latency (batch_size=1, CIFAR-10/ImageNet-10 testing dataset) of utilizing the different NeuralFuse generators with two different base models, ResNet18 and VGG19. While

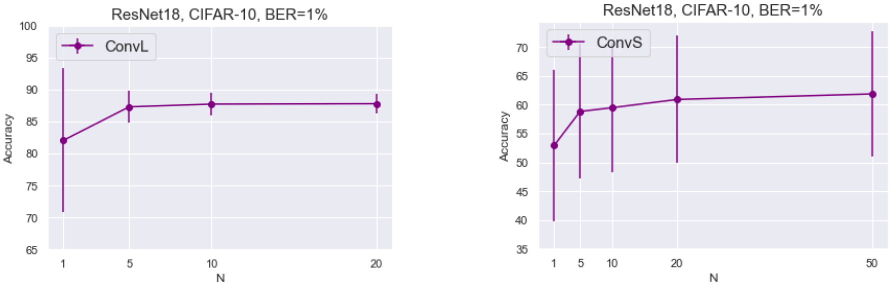
NeuralFuse contributes some additional latency, we consider this an unavoidable tradeoff necessary to achieve reduced energy consumption within our framework. Although the primary focus of this paper is not on latency, we acknowledge its importance. Future research could explore designing a more lightweight version of the NeuralFuse module or applying model compression techniques to minimize latency. Additionally, we recognize that running NeuralFuse on a general-purpose CPU could lead to different latency and energy consumption figures due to various influencing factors like CPU architecture and manufacturing processes.

Table 9: The Inference Latency of base model and base model with NeuralFuse.

	ResNet18 (CIFAR-10)	VGG19 (CIFAR-10)	ResNet18 (ImageNet-10)	VGG19 (ImageNet-10)
Base Model	5.84 ms	5.32 ms	6.21 ms	14.34 ms
+ ConvL	9.37 ms (+3.53)	8.96 ms (+3.64)	10.51 ms (+4.3)	17.66 ms (+3.32)
+ ConvS	7.86 ms (+2.02)	7.40 ms (+2.08)	8.28 ms (+2.07)	16.72 ms (+2.38)
+ DeConvL	9.18 ms (+3.34)	8.59 ms (+3.27)	10.07 ms (+3.86)	17.24 ms (+2.90)
+ DeConvS	7.49 ms (+1.65)	7.04 ms (+1.72)	7.79 ms (+1.58)	15.67 ms (+1.33)
+ UNetL	10.69 ms (+4.85)	10.06 ms (+4.74)	11.14 ms (+4.93)	18.54 ms (+4.20)
+ UNetS	10.63 ms (+4.79)	10.13 ms (+4.81)	11.36 ms (+5.15)	18.60 ms (+4.26)

E Ablation Studies

Study for N in EOPM. Here, we study the effect of N used in EOPM (Eq. 5). In Figure 7, we report the results for ConvL and ConvS on CIFAR-10 pre-trained ResNet18, under a 1% bit error rate (BER). The results demonstrate that if we apply larger N, the performance increases until convergence. Specifically, for ConvL (Figure 7a), larger N empirically has a smaller standard deviation; this means larger N gives better stability but at the cost of time-consuming training. In contrast, for the small generator ConvS (Figure 7b), we can find that the standard deviation is still large even trained by larger N; the reason might be that small generators are not as capable of learning representations as larger ones. Therefore, there exists a tradeoff between the *stability* of the generator performance and the total training time. In our implementation, choosing $N = 5$ or 10 is a good balance.



(a) Using ConvL

(b) Using ConvS

Figure 7: The experimental results on using different sizes of N for EOPM.

Tradeoff Between Clean Accuracy (CA) and Perturbed Accuracy (PA). We conducted an experiment to study the effect of different λ values, which balance the ratio of clean accuracy and perturbed accuracy. In Table 10, the experimental results showed that a smaller λ can preserve clean accuracy, but result in poor perturbed accuracy. On the contrary, larger λ can deliver higher perturbed accuracy, but with more clean accuracy drop. This phenomenon has also been observed in adversarial training [Zhang et al., 2019].

Comparison to Universal Input Perturbation (UIP). Moosavi-Dezfooli et al. [2017] has shown that there exists a universal adversarial perturbation to the input data such that the model will make wrong predictions on a majority of the perturbed images. In our NeuralFuse framework, the universal perturbation is a special case when we set $\mathcal{G}(\mathbf{x}) = \tanh(\mathbf{UIP})$ for any data sample $\mathbf{x}$. The transformed data sample then becomes $\mathbf{x}_t = \text{clip}_{[-1,1]}\big(\mathbf{x} + \tanh(\mathbf{UIP})\big)$, where $\mathbf{x}_t \in [-1, 1]^d$

Table 10: Experimental results based on λ value choosing. The results show that $\lambda = 5$ can balance the tradeoff between clean accuracy and perturbed accuracy.

Base Model	λ	CA	PA	ConvL		
				CA (NF)	PA (NF)	RP
ResNet18	10	92.6	38.9 ± 12.4	90.1	88.0 ± 1.7	49.1
	5			89.8	87.8 ± 1.7	48.8
	1			90.0	84.2 ± 3.8	45.3
	0.1			91.6	65.7 ± 9.3	26.8
	0.01			92.2	43.6 ± 13	4.7
VGG19	10	90.5	36.0 ± 12.0	89.6	77.9 ± 19	41.9
	5			89.8	77.7 ± 19	41.7
	1			89.9	73.1 ± 19	37.1
	0.1			89.1	51.2 ± 16	15.2
	0.01			90.2	36.8 ± 12	0.8

Note. CA (%): clean accuracy; PA (%): perturbed accuracy; NF: NeuralFuse; and RP: total recover percentage of PA (NF) vs. PA

and **UIP** is a trainable universal input perturbation that has the same size as the input data. The experimental results with the universal input perturbation are shown in Table 11. We observe that its performance is much worse than our proposed NeuralFuse. This result validates the necessity of adopting input-aware transformation for learning error-resistant data representations in low-voltage scenarios.

Table 11: Performance of the universal input perturbation (UIP) trained by EOPM on CIFAR-10 pre-trained ResNet18.

Base Model	BER	CA	PA	CA (**UIP**)	PA (**UIP**)	RP
ResNet18	1%	92.6	38.9 ± 12.4	91.8	37.9 ± 11	-1.0
	0.5%		70.1 ± 11.6	92.5	70.6 ± 11	0.5
ResNet50	1%	92.6	26.1 ± 9.4	80.7	21.0 ± 5.9	-5.1
	0.5%		61.0 ± 10.3	91.9	62.4 ± 12	1.4
VGG11	1%	88.4	42.2 ± 11.6	86.9	43.0 ± 11	0.8
	0.5%		63.6 ± 9.3	88.2	64.2 ± 8.8	0.6
VGG16	1%	90.3	35.7 ± 7.9	90.1	37.1 ± 8.5	1.4
	0.5%		66.6 ± 8.1	90.4	67.3 ± 8.1	0.7
VGG19	1%	90.5	36.0 ± 12.0	89.9	35.3 ± 12	-0.7
	0.5%		64.2 ± 12.4	90.1	64.4 ± 12	0.2

Note. BER: the bit-error rate of the base model; CA (%): clean accuracy; PA (%): perturbed accuracy; UIP: universal input transformation parameter; and RP: total recovery percentage of PA (UIP) vs. PA

F Additional Experimental Results on Relaxed Access Settings

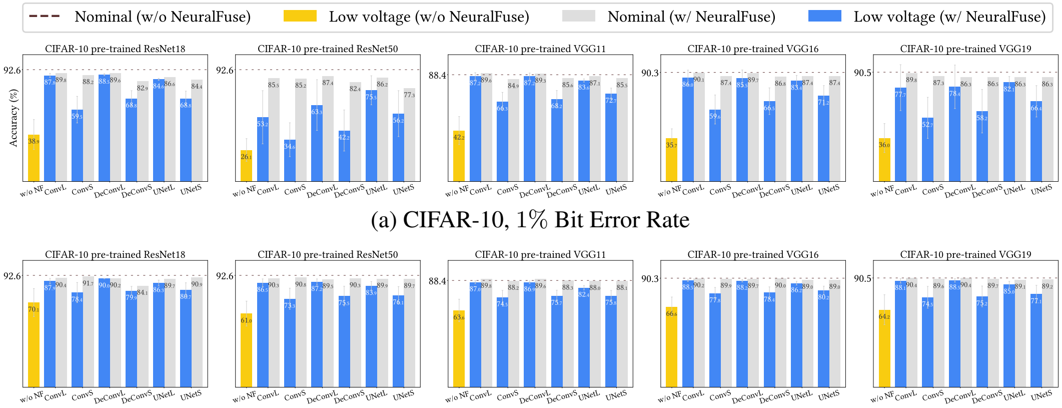
We conducted more experiments on *Relaxed Access* settings to show that our NeuralFuse can protect the models under different BER The results can be found in Sec. F.1 (CIFAR-10), Sec. F.2 (GTSRB), Sec. F.3 (ImageNet-10), and Sec. F.4 (CIFAR-100). For comparison, we also visualize the experimental results in the figures below each table.

F.1 CIFAR-10

Table 12: Testing accuracy (%) under 1% and 0.5% of random bit error rate on CIFAR-10.

Base Model	NF	CA	1% BER				0.5% BER			
			PA	CA (NF)	PA (NF)	RP	PA	CA (NF)	PA (NF)	RP
ResNet18	ConvL	92.6	38.9 ± 12.4	89.8	87.8 ± 1.7	48.8	70.1 ± 11.6	90.4	87.9 ± 2.2	17.8
	ConvS			88.2	59.5 ± 11	20.6		91.7	78.4 ± 8.3	8.3
	DeConvL			89.6	88.5 ± 0.8	49.6		90.2	90.0 ± 0.2	19.9
	DeConvS			82.9	68.8 ± 6.4	29.9		84.1	79.9 ± 3.6	9.8
	UNetL			86.6	84.6 ± 0.8	45.6		89.7	86.3 ± 2.4	16.2
	UNetS			84.4	68.8 ± 6.0	29.8		90.9	80.7 ± 5.8	10.7
ResNet50	ConvL	92.6	26.1 ± 9.4	85.5	53.2 ± 22	27.1	61.0 ± 10.3	90.3	86.5 ± 3.2	25.5
	ConvS			85.2	34.6 ± 14	8.5		90.8	73.3 ± 8.7	12.3
	DeConvL			87.4	63.3 ± 21	37.2		89.5	87.2 ± 2.5	26.2
	DeConvS			82.4	42.2 ± 17	16.1		90.3	75.5 ± 8.1	14.5
	UNetL			86.2	75.5 ± 12	49.4		89.9	83.9 ± 3.6	22.9
	UNetS			77.3	56.2 ± 19	30.1		89.7	76.1 ± 7.2	15.1
VGG11	ConvL	88.4	42.2 ± 11.6	89.6	87.2 ± 2.9	45.1	63.6 ± 9.3	89.8	87.0 ± 1.3	23.3
	ConvS			84.9	66.3 ± 7.5	24.1		88.2	74.5 ± 5.7	10.9
	DeConvL			89.3	87.2 ± 2.6	45.0		89.6	86.9 ± 1.1	23.2
	DeConvS			85.6	68.2 ± 7.1	26.0		88.3	75.7 ± 4.6	12.1
	UNetL			87.1	83.6 ± 1.3	41.4		88.0	82.4 ± 1.8	18.8
	UNetS			85.5	72.7 ± 4.6	30.5		88.1	75.8 ± 4.3	12.2
VGG16	ConvL	90.3	35.7 ± 7.9	90.1	86.0 ± 6.2	50.3	66.6 ± 8.1	90.2	88.5 ± 0.9	21.9
	ConvS			87.4	59.6 ± 12	23.9		89.9	77.8 ± 4.8	11.1
	DeConvL			89.7	85.5 ± 6.8	49.8		89.7	88.2 ± 1.0	21.4
	DeConvS			86.8	66.5 ± 11	30.8		90.0	78.4 ± 4.7	11.8
	UNetL			87.4	83.4 ± 4.4	47.7		89.0	86.2 ± 1.5	19.6
	UNetS			87.4	71.2 ± 8.2	35.5		89.0	80.2 ± 3.5	13.7
VGG19	ConvL	90.5	36.0 ± 12.0	89.8	77.7 ± 19	41.7	64.2 ± 12.4	90.4	88.1 ± 1.8	23.9
	ConvS			87.3	52.7 ± 17	16.7		89.6	74.5 ± 9.0	10.3
	DeConvL			86.3	78.4 ± 18	42.4		90.4	88.5 ± 1.4	24.3
	DeConvS			86.5	58.2 ± 18	22.2		89.7	75.2 ± 8.6	11.0
	UNetL			86.3	82.1 ± 4.8	46.0		89.1	85.0 ± 2.7	20.8
	UNetS			86.3	66.4 ± 13	30.4		89.2	77.1 ± 7.3	12.9

Note. CA (%): clean accuracy; PA (%): perturbed accuracy; NF: NeuralFuse; and RP: total recovery percentage of PA (NF) vs. PA



(a) CIFAR-10, 1% Bit Error Rate

(b) CIFAR-10, 0.5% Bit Error Rate

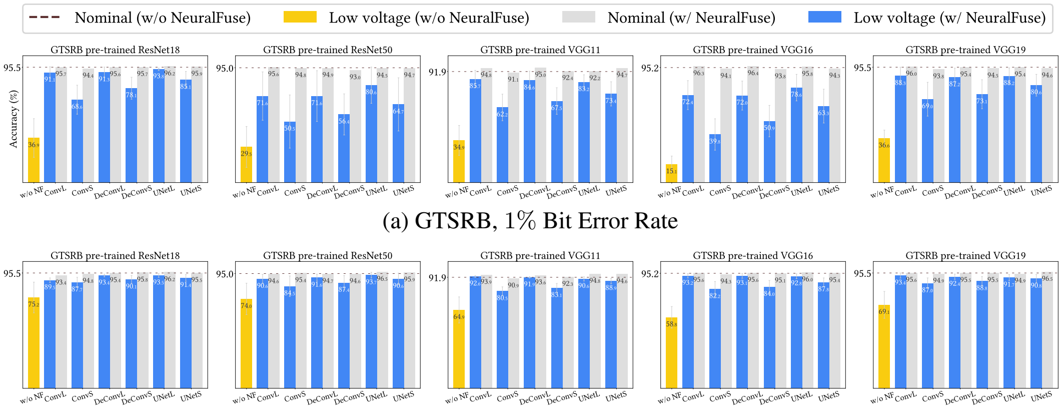
Figure 8: Experimental results on CIFAR-10

F.2 GTSRB

Table 13: Testing accuracy (%) under 1% and 0.5% of random bit error rate on GTSRB.

Base Model	NF	CA	**1% BER**				**0.5% BER**			
			PA	CA (NF)	PA (NF)	RP	PA	CA (NF)	PA (NF)	RP
ResNet18	ConvL	95.5	36.9 ± 16.0	95.7	91.1 ± 4.7	54.2	75.2 ± 12.7	93.4	89.5 ± 1.9	14.3
	ConvS			94.4	68.6 ± 12	31.7		94.8	87.7 ± 4.2	12.4
	DeConvL			95.6	91.3 ± 4.3	54.4		95.4	93.4 ± 1.1	18.1
	DeConvS			95.7	78.1 ± 9.1	41.2		95.8	90.1 ± 3.3	14.9
	UNetL			96.2	93.8 ± 1.0	56.9		96.2	93.5 ± 1.6	18.3
	UNetS			95.9	85.1 ± 6.9	48.2		95.5	91.4 ± 2.8	16.2
ResNet50	ConvL	95.0	29.5 ± 16.9	95.6	71.6 ± 20	42.1	74.0 ± 13.0	94.6	90.6 ± 3.7	16.6
	ConvS			94.8	50.5 ± 22	21.0		95.4	84.5 ± 8.5	10.5
	DeConvL			94.9	71.6 ± 21	42.0		94.7	91.6 ± 2.9	17.6
	DeConvS			93.0	56.4 ± 17	26.9		94.6	87.4 ± 5.9	13.5
	UNetL			94.5	80.6 ± 15	51.1		96.5	93.7 ± 2.3	19.7
	UNetS			94.7	64.7 ± 22	35.2		95.9	90.6 ± 4.8	16.7
VGG11	ConvL	91.9	34.9 ± 12.4	94.8	85.7 ± 7.2	50.9	64.9 ± 10.8	93.9	92.6 ± 0.7	27.7
	ConvS			91.1	62.2 ± 11	27.3		90.9	80.5 ± 3.5	15.7
	DeConvL			95.0	84.6 ± 7.6	49.7		93.6	91.9 ± 0.6	27.1
	DeConvS			92.4	67.5 ± 11	32.6		92.3	83.1 ± 3.7	18.2
	UNetL			92.2	83.2 ± 6.0	48.3		94.8	90.6 ± 1.7	25.7
	UNetS			94.7	73.4 ± 10	38.5		94.6	88.9 ± 2.2	24.1
VGG16	ConvL	95.2	15.1 ± 6.8	96.3	72.4 ± 12	57.3	58.8 ± 8.9	95.6	93.2 ± 1.8	34.4
	ConvS			94.1	39.8 ± 13	24.6		94.3	82.2 ± 6.2	23.4
	DeConvL			96.4	72.0 ± 12	56.9		95.6	93.1 ± 2.0	34.3
	DeConvS			93.8	50.9 ± 13	35.8		95.1	84.0 ± 5.3	25.2
	UNetL			95.8	78.6 ± 11	63.5		96.0	92.8 ± 2.0	34.0
	UNetS			94.3	63.3 ± 14	48.1		95.4	87.8 ± 3.6	29.0
VGG19	ConvL	95.5	36.6 ± 6.8	96.0	88.3 ± 7.2	51.7	69.1 ± 11.1	95.6	93.4 ± 2.1	24.2
	ConvS			93.8	69.0 ± 14	32.4		94.9	87.0 ± 4.4	17.8
	DeConvL			95.4	87.2 ± 7.5	50.6		95.5	92.4 ± 2.2	23.3
	DeConvS			94.5	73.1 ± 12	36.5		95.5	88.8 ± 3.7	19.7
	UNetL			95.4	88.2 ± 6.7	51.7		94.9	91.7 ± 2.5	22.6
	UNetS			94.6	80.6 ± 9.0	44.1		96.5	90.8 ± 3.4	21.6

Note. CA (%): clean accuracy; PA (%): perturbed accuracy; NF: NeuralFuse; and RP: total recovery percentage of PA (NF) vs. PA



(a) GTSRB, 1% Bit Error Rate

(b) GTSRB, 0.5% Bit Error Rate

Figure 9: Experimental results on GTSRB.

F.3 ImageNet-10

Table 14: Testing accuracy under 0.5% of random bit error rate on ImageNet-10.

Base Model	NF	CA	0.5% BER			
			PA	CA (NF)	PA (NF)	RP
ResNet18	ConvL	92.2	72.3 ± 7.0	94.0	88.0 ± 2.0	15.7
	ConvS			91.8	83.6 ± 4.1	11.3
	DeConvL			94.0	89.2 ± 1.3	16.9
	DeConvS			92.8	87.5 ± 2.3	15.2
	UNetL			94.0	88.1 ± 1.4	15.8
	UNetS			93.2	86.4 ± 2.2	14.1
ResNet50	ConvL	89.8	39.4 ± 11	92.2	80.0 ± 5.8	40.6
	ConvS			91.8	65.0 ± 11	25.6
	DeConvL			93.0	79.4 ± 5.9	40.0
	DeConvS			93.2	70.9 ± 9.1	31.5
	UNetL			92.2	80.5 ± 5.8	41.1
	UNetS			92.4	73.6 ± 8.9	34.2
VGG11	ConvL	91.6	47.8 ± 13	92.0	86.1 ± 3.7	38.3
	ConvS			89.4	66.4 ± 7.1	18.6
	DeConvL			91.0	86.0 ± 3.0	38.2
	DeConvS			89.0	72.5 ± 7.8	24.7
	UNetL			92.4	83.0 ± 3.5	35.2
	UNetS			86.2	73.5 ± 6.0	25.7
VGG16	ConvL	94.6	38.4 ± 17	90.8	77.1 ± 11	38.7
	ConvS			90.2	60.2 ± 14	21.8
	DeConvL			91.2	77.2 ± 11	38.8
	DeConvS			90.0	62.3 ± 14	23.9
	UNetL			90.6	81.1 ± 5.9	42.7
	UNetS			86.4	72.3 ± 8.8	33.9
VGG19	ConvL	92.4	37.2 ± 11	91.4	75.5 ± 8.8	38.3
	ConvS			88.8	56.5 ± 13	19.3
	DeConvL			91.0	75.9 ± 8.9	38.7
	DeConvS			88.8	64.0 ± 11	26.8
	UNetL			89.4	77.9 ± 6.1	40.7
	UNetS			87.6	65.9 ± 10	28.7

Note. CA (%): clean accuracy; PA (%): perturbed accuracy; NF: NeuralFuse; and RP: total recovery percentage of PA (NF) vs. PA

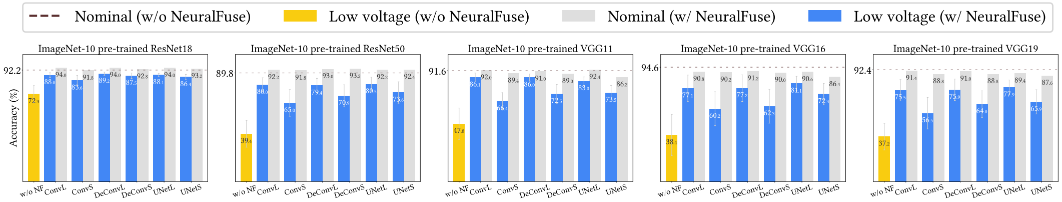


Figure 10: Experimental results on ImageNet-10, 0.5% Bit Error Rate.

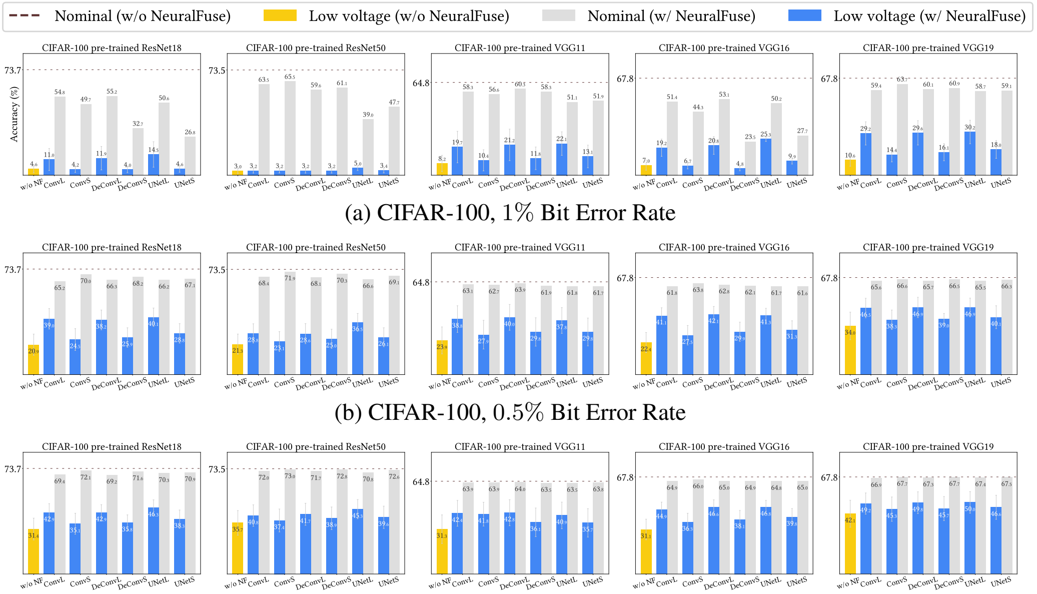
F.4 CIFAR-100

As mentioned in the previous section, larger generators like ConvL, DeConvL, and UNetL have better performance than small generators. For CIFAR-100, we find that the gains of utilizing NeuralFuse are less compared to the other datasets. We believe this is because CIFAR-100 is a more challenging dataset (more classes) for the generators to learn to protect the base models. Nevertheless, NeuralFuse can still function to restore some degraded accuracy; these results also demonstrate that our NeuralFuse is applicable to different datasets. In addition, although the recover percentage is less obvious on CIFAR-100 (the more difficult dataset), we can still conclude that our NeuralFuse is applicable to different datasets.

Table 15: Testing accuracy (%) under 1%, 0.5% and 0.35% of random bit error rate on CIFAR-100.

Base Model	NF	C.A.	1% BER				0.5% BER				0.35% BER			
			PA	CA (NF)	PA (NF)	RP	PA	CA (NF)	PA (NF)	RP	PA	CA (NF)	PA (NF)	RP
ResNet18	ConvL	73.7	4.6 ± 2.9	54.8	11.0 ± 7.7	6.4	20.9 ± 7.4	65.2	39.0 ± 7.1	18.1	31.4 ± 7.6	69.4	42.9 ± 6.2	11.4
	ConvS			49.7	4.2 ± 2.2	-0.4		70.0	24.5 ± 7.6	3.6		72.1	35.1 ± 7.3	3.7
	DeConvL			55.2	11.9 ± 8.2	7.3		66.3	38.2 ± 6.9	17.3		69.2	42.9 ± 5.5	11.4
	DeConvS			32.7	4.0 ± 2.2	-0.6		68.2	25.9 ± 6.8	5		71.6	35.8 ± 5.5	4.4
	UNetL			50.6	14.5 ± 8.9	10.0		66.2	40.1 ± 6.4	19.2		70.3	46.3 ± 5.5	14.9
	UNetS			26.8	4.6 ± 2.5	-0.0		67.1	28.8 ± 6.8	7.9		70.9	38.3 ± 6.4	6.9
ResNet50	ConvL	73.5	3.0 ± 1.8	63.5	3.2 ± 1.7	0.1	21.3 ± 7.0	68.4	28.8 ± 6.7	7.6	35.7 ± 8.6	72.0	40.8 ± 7.5	5.1
	ConvS			65.5	3.2 ± 1.6	0.1		71.9	23.1 ± 6.9	1.9		73.0	37.4 ± 8.0	1.7
	DeConvL			59.6	3.2 ± 1.7	0.2		68.1	28.6 ± 7.0	7.4		71.7	41.7 ± 7.7	6.1
	DeConvS			61.1	3.2 ± 1.7	0.1		70.3	25.0 ± 6.7	3.7		72.8	38.9 ± 7.9	3.3
	UNetL			39.0	5.0 ± 1.7	1.9		66.6	36.5 ± 6.2	15.3		70.8	45.3 ± 6.7	9.6
	UNetS			47.7	3.4 ± 1.8	0.3		69.1	26.1 ± 6.6	4.8		72.6	39.6 ± 7.8	3.9
VGG11	ConvL	64.8	8.2 ± 5.7	58.3	19.7 ± 11	11.5	23.9 ± 9.4	63.1	38.8 ± 9.3	15.0	31.3 ± 10	63.9	42.4 ± 9.0	11.1
	ConvS			56.6	10.4 ± 7.4	2.2		62.7	27.9 ± 10	4.0		63.9	41.8 ± 8.3	10.5
	DeConvL			60.3	21.2 ± 11	13.0		63.9	40.0 ± 9.0	16.2		64.0	42.8 ± 9.1	11.5
	DeConvS			58.3	11.8 ± 7.9	3.5		61.9	29.8 ± 9.9	5.9		63.5	36.1 ± 10	4.8
	UNetL			51.1	22.1 ± 8.2	13.9		61.8	37.8 ± 9.0	13.9		63.5	40.9 ± 9.3	9.6
	UNetS			51.9	13.1 ± 7.9	4.9		61.7	29.8 ± 9.7	6.0		63.8	35.7 ± 9.9	4.5
VGG16	ConvL	67.8	7.0 ± 3.5	51.4	19.2 ± 6.0	12.6	22.4 ± 7.0	61.8	41.1 ± 5.6	18.7	31.1 ± 7.2	64.9	44.9 ± 5.3	13.8
	ConvS			44.3	6.7 ± 2.3	0.1		63.8	27.5 ± 6.8	5.1		66.0	36.3 ± 6.1	5.1
	DeConvL			53.1	20.8 ± 6.2	14.2		62.8	42.1 ± 5.5	19.8		65.0	46.6 ± 5.2	15.5
	DeConvS			23.5	4.8 ± 1.7	-1.8		62.1	29.9 ± 6.7	7.5		64.9	38.1 ± 6.3	7.0
	UNetL			50.2	25.3 ± 1.7	18.7		61.7	41.3 ± 5.0	18.9		64.8	46.8 ± 4.6	15.7
	UNetS			27.7	9.9 ± 2.1	3.3		61.6	31.3 ± 6.3	8.9		65.0	39.8 ± 5.9	8.7
VGG19	ConvL	67.8	10.6 ± 4.3	59.4	29.2 ± 8.1	18.6	34.0 ± 9.6	65.6	46.5 ± 6.8	12.5	42.1 ± 9.4	66.9	49.2 ± 7.4	7.0
	ConvS			63.7	14.4 ± 5.1	3.8		66.6	38.3 ± 6.8	4.2		67.7	45.3 ± 8.5	3.2
	DeConvL			60.1	29.6 ± 8.5	19.0		65.7	46.9 ± 7.1	12.9		67.3	49.8 ± 7.6	7.6
	DeConvS			60.9	16.1 ± 6.0	5.6		66.5	39.0 ± 3.7	5.0		67.7	45.7 ± 8.4	3.6
	UNetL			58.7	30.2 ± 8.2	19.6		65.5	46.9 ± 6.5	12.9		67.4	50.0 ± 7.5	7.9
	UNetS			59.1	18.0 ± 6.2	7.4		66.3	40.1 ± 8.0	6.1		67.5	46.6 ± 8.4	4.5

Note. CA (%): clean accuracy; PA (%): perturbed accuracy; NF: NeuralFuse; and RP: total recovery percentage of PA (NF) vs. PA



(a) CIFAR-100, 1% Bit Error Rate

(b) CIFAR-100, 0.5% Bit Error Rate

(c) CIFAR-100, 0.35% Bit Error Rate

Figure 11: Experimental results on CIFAR-100.

G Additional Experimental Results on Restricted Access Settings (Transferability)

We conduct more experiments with *Restricted Access* settings to show that our NeuralFuse can be transferred to protect various black-box models. The experimental results are shown in Sec. G.1 (CIFAR-10), Sec. G.2 (GTSRB), and Sec. G.3 (CIFAR-100).

We find that using VGG19 as a white-box surrogate model has better *transferability* than ResNet18 for all datasets. In addition, we can observe that some NeuralFuse generators have *downward applicability* if base models have a similar architecture. In other words, if we try to transfer a generator trained on a large BER (e.g., 1%) to a model with a small BER (e.g., 0.5%), the performance will be better than that of a generator trained with the original BER (e.g., 0.5%). For example, in Table 16, we could find that if we use VGG19 as a source model to train the generator ConvL (1%), the generator could deliver better performance (in terms of PA (NF)) when applied to similar base models (e.g., VGG11, VGG16, or VGG19) under a 0.5% BER, compared to using itself as a source model (shown in Table 12). We conjecture that this is because the generators trained on a larger BER can also cover the error patterns of a smaller BER, and thus they have better generalizability across smaller B.E.Rs.

To further improve the transferability to cross-architecture target models, we also conduct an experiment in Sec. G.4 to show that using ensemble-based training can help the generator to achieve this feature.

G.1 CIFAR-10

The results of CIFAR-10 in which NeuralFuse is trained at 1% BER are shown in Table 16.

Table 16: Transfer results on CIFAR-10: NeuralFuse trained on SM with 1% BER

SM	TM	BER	CA	PA	ConvL (1%)			UNetL (1%)		
					CA (NF)	PA (NF)	RP	CA (NF)	PA (NF)	RP
ResNet18	ResNet18	0.5%	92.6	70.1 ± 11.6	89.8	89.5 ± 0.2	19.4	86.6	86.2 ± 0.3	16.1
	ResNet50	1%	92.6	26.1 ± 9.4	89.5	36.0 ± 19	9.9	85.2	38.8 ± 19	12.7
		0.5%		61.0 ± 10.3		75.1 ± 10	14.1		77.1 ± 5.0	16.1
	VGG11	1%	88.4	42.2 ± 11.6	88.4	62.5 ± 8.4	20.3	76.8	61.1 ± 8.5	18.9
		0.5%		63.6 ± 9.3		81.0 ± 4.6	17.4		73.7 ± 3.0	10.1
	VGG16	1%	90.3	35.7 ± 7.9	89.6	63.3 ± 18	27.6	85.2	59.9 ± 16	24.2
		0.5%		66.6 ± 8.1		85.0 ± 3.4	18.4		80.2 ± 4.5	13.6
	VGG19	1%	90.5	36.0 ± 12.0	89.6	50.7 ± 22	14.7	85.3	51.1 ± 16	15.1
		0.5%		64.2 ± 12.4		80.2 ± 8.7	16.0		76.5 ± 7.8	12.3
VGG19	ResNet18	1%	92.6	38.9 ± 12.4	89.8	61.0 ± 17	22.1	87.0	69.7 ± 11	30.8
		0.5%		70.1 ± 11.6		86.1 ± 6.9	16.0		84.2 ± 3.0	14.1
	ResNet50	1%	92.6	26.1 ± 9.4	89.9	34.0 ± 19	7.9	87.0	44.2 ± 17	18.1
		0.5%		61.0 ± 10.3		76.5 ± 10	15.5		80.7 ± 4.2	19.7
	VGG11	1%	88.4	42.2 ± 11.6	89.7	76.5 ± 7.0	34.3	87.1	79.9 ± 5.6	37.7
		0.5%		63.6 ± 9.3		88.0 ± 2.1	24.4		85.4 ± 0.8	21.8
	VGG16	1%	90.3	35.7 ± 7.9	89.6	75.5 ± 12	39.8	87.2	78.9 ± 7.8	43.2
		0.5%		66.6 ± 8.1		88.9 ± 0.6	22.3		86.2 ± 0.3	19.6
	VGG19	0.5%	90.5	64.2 ± 12.4	89.8	89.6 ± 8.7	25.4	87.4	86.8 ± 0.4	22.6

Note. SM: source model, used for training generators; TM: target model, used for testing generators; BER: the bit-error rate of the target model; CA (%): clean accuracy; PA (%): perturbed accuracy; NF: NeuralFuse; and RP: total recovery percentage of PA (NF) vs. PA

G.2 GTSRB

In Tables 17 and 18, we show the results on GTSRB in which NeuralFuse is trained at 1.5% and 1% BER, respectively.

Table 17: Transfer results on GTSRB: NeuralFuse trained on SM with 1.5% BER

SM	TM	BER	CA	PA	ConvL (1.5%)			UNetL (1.5%)		
					CA (NF)	PA (NF)	RP	CA (NF)	PA (NF)	RP
ResNet18	ResNet18	1%	95.5	36.9 ± 16.0	95.7	93.9 ± 1.9	57.0	94.9	94.4 ± 0.4	57.5
		0.5%		75.2 ± 12.7		95.7 ± 0.2	20.5		94.8 ± 0.2	19.6
	ResNet50	1%	95.0	29.5 ± 16.9	94.4	37.0 ± 22	7.5	94.4	47.1 ± 23	17.6
		0.5%		74.0 ± 13.0		77.5 ± 13	3.5		84.8 ± 9.5	10.8
	VGG11	1%	91.9	34.9 ± 12.4	92.8	45.2 ± 10	10.3	91.4	50.5 ± 13	15.6
		0.5%		64.9 ± 10.8		79.4 ± 5.8	14.5		83.9 ± 4.2	19.0
	VGG16	1%	95.2	15.1 ± 6.8	95.4	31.1 ± 13	15.9	94.6	36.8 ± 12	21.7
		0.5%		58.8 ± 8.9		84.5 ± 8.3	25.8		86.0 ± 8.6	27.2
	VGG19	1%	95.5	36.6 ± 6.8	95.0	56.4 ± 15	19.8	94.3	60.8 ± 15	24.2
		0.5%		69.1 ± 11.1		86.9 ± 3.4	17.8		87.7 ± 3.8	18.6
VGG19	ResNet18	1%	95.5	36.9 ± 16.0	88.4	50.3 ± 12	13.4	92.8	63.7 ± 16	26.8
		0.5%		75.2 ± 12.7		77.9 ± 7.4	2.7		87.5 ± 3.9	12.3
	ResNet50	1%	95.0	29.5 ± 16.9	87.5	29.7 ± 17	0.2	92.5	40.4 ± 21	10.9
		0.5%		74.0 ± 13.0		67.9 ± 17	-6.1		77.5 ± 15	3.5
	VGG11	1%	91.9	34.9 ± 12.4	89.7	47.1 ± 11	12.2	93.5	60.0 ± 12	25.1
		0.5%		64.9 ± 10.8		76.3 ± 5.1	11.4		86.0 ± 3.8	21.1
	VGG16	1%	95.2	15.1 ± 6.8	93.0	29.2 ± 15	14.1	93.0	38.5 ± 16	23.4
		0.5%		58.8 ± 8.9		75.7 ± 12	16.9		79.9 ± 8.3	21.1
	VGG19	1%	95.5	36.6 ± 6.8	95.1	87.4 ± 6.0	50.8	94.6	88.7 ± 5.0	52.1
		0.5%		69.1 ± 11.1		92.4 ± 2.4	23.3		92.4 ± 2.2	23.3

Note. SM: source model, used for training generators; TM: target model, used for testing generators; BER: the bit-error rate of the target model; CA (%): clean accuracy; PA (%): perturbed accuracy; NF: NeuralFuse; and RP: total recovery percentage of PA (NF) vs. PA

Table 18: Transfer results on GTSRB: NeuralFuse trained on SM with 1% BER

SM	TM	BER	CA	PA	ConvL (1%)			UNetL (1%)		
					CA (NF)	PA (NF)	RP	CA (NF)	PA (NF)	RP
ResNet18	ResNet18	0.5%	95.5	75.2 ± 12.7	95.7	95.3 ± 0.5	20.1	96.2	95.7 ± 0.3	20.5
	ResNet50	1%	95.0	29.5 ± 16.9	94.5	35.6 ± 21	6.1	95.6	42.6 ± 23	13.1
		0.5%		74.0 ± 13.0		78.8 ± 13	4.8		87.3 ± 9.0	13.3
	VGG11	1%	91.9	34.9 ± 12.4	93.1	45.8 ± 11	10.9	94.0	47.1 ± 14	12.2
		0.5%		64.9 ± 10.8		81.8 ± 5.0	16.9		84.2 ± 4.8	19.3
	VGG16	1%	95.2	15.1 ± 6.8	95.5	26.5 ± 12	11.4	95.5	32.4 ± 11	17.3
		0.5%		58.8 ± 8.9		82.2 ± 9.0	23.4		85.4 ± 6.7	26.6
	VGG19	1%	95.5	36.6 ± 6.8	94.9	53.2 ± 14	16.6	95.6	60.9 ± 15	24.3
		0.5%		69.1 ± 11.1		85.4 ± 4.5	16.3		87.5 ± 3.7	18.4
VGG19	ResNet18	1%	95.5	36.9 ± 16.0	93.7	53.1 ± 16	16.2	95.0	63.4 ± 18	26.5
		0.5%		75.2 ± 12.7		83.9 ± 7.6	8.7		89.7 ± 4.8	14.5
	ResNet50	1%	95.0	29.5 ± 16.9	92.8	30.6 ± 18	1.1	95.4	38.9 ± 22	9.4
		0.5%		74.0 ± 13.0		74.7 ± 18	0.7		81.5 ± 16	7.5
	VGG11	1%	91.9	34.9 ± 12.4	93.7	50.6 ± 11	15.7	95.1	58.9 ± 15	24.0
		0.5%		64.9 ± 10.8		82.3 ± 5.1	17.4		87.5 ± 3.7	22.6
	VGG16	1%	95.2	15.1 ± 6.8	95.2	27.8 ± 15	12.7	95.2	33.5 ± 14	18.4
		0.5%		58.8 ± 8.9		79.0 ± 12	20.2		81.8 ± 7.8	23.0
	VGG19	0.5%	95.5	69.1 ± 11.1	96.0	94.0 ± 2.2	24.9	95.4	93.9 ± 2.1	24.8

Note. SM: source model, used for training generators; TM: target model, used for testing generators; BER: the bit-error rate of the target model; CA (%): clean accuracy; PA (%): perturbed accuracy; NF: NeuralFuse; and RP: total recovery percentage of PA (NF) vs. PA

G.3 CIFAR-100

In Tables 19 and 20, we show results on CIFAR-100 with NeuralFuse trained at 1% and 0.5% BER, respectively.

Table 19: Transfer results on CIFAR-100: NeuralFuse trained on SM with 1% BER

SM	TM	BER	CA	PA	ConvL (1%)			UNetL (1%)		
					CA (NF)	PA (NF)	RP	CA (NF)	PA (NF)	RP
ResNet18	ResNet18	0.5%	73.7	20.9 ± 7.4	54.8	35.8 ± 5.2	14.9	50.6	39.3 ± 2.8	18.4
		0.35%		31.4 ± 7.6		41.7 ± 3.7	10.3		43.3 ± 1.4	11.9
	ResNet50	1%	73.5	3.0 ± 1.8	44.9	2.2 ± 2.0	-0.8	41.5	2.4 ± 1.9	-0.6
		0.5%		21.3 ± 7.0		15.9 ± 8.2	-5.4		17.1 ± 7.1	-4.2
		0.35%		35.7 ± 8.6		23.7 ± 7.1	-12.0		26.2 ± 5.6	-9.5
	VGG11	1%	64.8	8.2 ± 5.7	41.2	9.8 ± 5.6	1.6	37.5	10.2 ± 5.1	2.0
		0.5%		23.9 ± 9.4		24.2 ± 5.9	0.3		24.5 ± 4.7	0.6
		0.35%		31.3 ± 10.0		29.0 ± 5.4	-2.3		28.2 ± 4.5	-3.1
	VGG16	1%	67.8	7.0 ± 3.5	44.0	7.9 ± 3.7	0.9	39.5	10.1 ± 4.5	3.1
		0.5%		22.4 ± 7.0		22.4 ± 7.6	0.0		26.3 ± 5.3	3.9
		0.35%		31.1 ± 7.2		28.1 ± 5.9	-3.0		30.6 ± 3.6	-0.5
	VGG19	1%	67.8	10.6 ± 4.3	44.2	13.5 ± 6.1	2.9	40.7	15.6 ± 6.2	5.0
		0.5%		34.0 ± 9.6		27.9 ± 4.8	-6.1		29.3 ± 4.6	-4.7
		0.35%		42.1 ± 9.4		33.2 ± 48	-8.9		32.8 ± 3.9	-9.3
VGG19	ResNet18	1%	73.7	4.6 ± 2.9	55.5	5.8 ± 3.7	1.2	57.3	6.8 ± 4.4	2.2
		0.5%		20.9 ± 7.4		24.6 ± 6.3	3.7		28.1 ± 5.9	7.2
		0.35%		31.4 ± 7.6		31.1 ± 5.0	-0.3		36.4 ± 4.5	5.0
	ResNet50	1%	73.5	3.0 ± 1.8	56.1	2.8 ± 2.1	-0.2	56.1	3.7 ± 2.4	0.7
		0.5%		21.3 ± 7.0		18.9 ± 8.6	-2.4		22.8 ± 8.5	1.5
		0.35%		35.7 ± 8.6		28.7 ± 8.2	-7.0		33.7 ± 7.0	-2.0
	VGG11	1%	64.8	8.2 ± 5.7	52.8	12.3 ± 8.4	4.1	53.9	15.4 ± 9.4	7.2
		0.5%		23.9 ± 9.4		30.0 ± 9.3	6.1		33.3 ± 7.2	9.4
		0.35%		31.3 ± 10.0		36.5 ± 7.7	5.2		38.8 ± 6.5	7.5
	VGG16	1%	67.8	7.0 ± 3.5	53.6	11.2 ± 4.4	4.2	55.2	13.6 ± 5.2	6.6
		0.5%		22.4 ± 7.0		32.4 ± 7.3	10.0		35.9 ± 6.2	13.5
		0.35%		31.1 ± 7.2		39.4 ± 6.3	8.3		42.4 ± 4.9	11.3
	VGG19	0.5%	67.8	34.0 ± 9.6	59.4	50.2 ± 3.1	16.2	58.7	49.1 ± 3.5	15.1
		0.35%		42.1 ± 9.4		53.1 ± 2.3	11.0		52.0 ± 3.1	9.9

Note. SM: source model, used for training generators; TM: target model, used for testing generators; BER: the bit-error rate of the target model; CA (%): clean accuracy; PA (%): perturbed accuracy; NF: NeuralFuse; and RP: total recovery percentage of PA (NF) vs. PA

Table 20: Transfer results on CIFAR-100: NeuralFuse trained on SM with 0.5% BER

SM	TM	BER	CA	PA	ConvL (0.5%)			UNetL (0.5%)		
					CA (NF)	PA (NF)	RP	CA (NF)	PA (NF)	RP
ResNet18	ResNet18	0.35%	73.7	31.4 ± 7.6	65.2	47.7 ± 4.9	16.3	66.2	49.2 ± 4.1	17.8
	ResNet50	0.5%	73.5	21.3 ± 7.0	62.5	24.0 ± 9.9	2.8	63.5	26.4 ± 9.1	5.1
		0.35%		35.7 ± 8.6		36.3 ± 8.9	0.6		39.4 ± 8.1	3.7
	VGG11	0.5%	64.8	23.9 ± 9.4	59.2	33.0 ± 9.8	9.2	61.1	34.2 ± 9.8	10.3
		0.35%		31.3 ± 10.0		40.4 ± 8.7	9.1		41.4 ± 9.0	10.1
	VGG16	0.5%	67.8	22.4 ± 7.0	59.5	34.7 ± 8.0	12.3	61.4	37.5 ± 6.8	15.2
		0.35%		31.1 ± 7.2		42.9 ± 6.0	11.8		45.3 ± 4.9	14.2
	VGG19	0.5%	67.8	34.0 ± 9.6	61.6	43.7 ± 6.2	9.6	62.0	45.0 ± 6.3	11.0
		0.35%		42.1 ± 9.4		49.0 ± 5.5	6.8		50.5 ± 5.3	8.3
VGG19	ResNet18	0.5%	73.7	20.9 ± 7.4	66.1	24.9 ± 6.7	4.0	67.8	27.7 ± 6.8	6.8
		0.35%		31.4 ± 7.6		34.4 ± 5.4	3.0		38.1 ± 5.6	6.7
	ResNet50	0.5%	73.5	21.3 ± 7.0	66.2	22.7 ± 7.8	1.4	66.7	25.4 ± 8.0	4.2
		0.35%		35.7 ± 8.6		35.5 ± 7.7	-0.2		38.8 ± 7.5	3.2
	VGG11	0.5%	64.8	23.9 ± 9.4	59.9	29.3 ± 10	5.4	61.0	31.2 ± 9.8	7.4
		0.35%		31.3 ± 10.0		36.6 ± 9.5	5.3		38.1 ± 9.0	6.8
	VGG16	0.5%	67.8	22.4 ± 7.0	62.5	30.8 ± 7.3	8.4	62.6	33.0 ± 7.3	10.7
		0.35%		31.1 ± 7.2		40.0 ± 6.5	8.9		42.5 ± 5.9	11.3
	VGG19	0.35%	67.8	42.1 ± 9.4	65.6	52.0 ± 6.2	9.8	65.5	52.6 ± 6.1	10.4

Note. SM: source model, used for training generators; TM: target model, used for testing generators; BER: the bit-error rate of the target model; CA (%): clean accuracy; PA (%): perturbed accuracy; NF: NeuralFuse; and RP: total recovery percentage of PA (NF) vs. PA

G.4 Generator Ensembling

To improve the transferability performance on cross-architecture cases (e.g., using ResNet-based models as surrogate models to train NeuralFuse and then transfer NeuralFuse to VGG-based target models), we try to adopt ensemble surrogate models to train our NeuralFuse. The experimental results are shown in Table 21. We use the same experimental settings mentioned in Table 1 but change one source model (e.g., ResNet18 or VGG19) into two (ResNet18 with VGG19) for training. The results show that the overall performance is better than the results shown in Table 1, which means ensemble-based training can easily solve the performance degradation on cross-architecture target models.

Table 21: Transfer results on CIFAR-10: NeuralFuse trained on two SM with 1.5% BER

SM	TM	BER	CA	PA	ConvL (1.5%)			UNetL (1.5%)		
					CA (NF)	PA (NF)	RP	CA (NF)	PA (NF)	RP
ResNet18 + VGG19	ResNet18	1%	92.6	38.9 ± 12.4	89.4	88.1 ± 1.0	49.2	86.3	85.4 ± 0.5	46.5
		0.5%		70.1 ± 11.6		89.2 ± 0.2	19.1		86.1 ± 0.2	16.0
	ResNet50	1%	92.6	26.1 ± 9.4	89.3	44.0 ± 22	17.9	86.1	50.9 ± 20	24.8
		0.5%		61.0 ± 10.3		80.3 ± 6.7	19.3		78.6 ± 3.9	17.6
	VGG11	1%	88.4	42.2 ± 11.6	89.1	77.0 ± 5.6	34.8	85.9	82.3 ± 4.1	40.1
		0.5%		63.6 ± 9.3		87.5 ± 1.6	23.9		85.0 ± 0.6	21.4
	VGG16	1%	90.3	35.7 ± 7.9	89.1	80.5 ± 8.6	44.8	85.7	81.4 ± 5.5	45.7
		0.5%		66.6 ± 8.1		88.2 ± 0.7	21.6		85.0 ± 0.7	18.4
	VGG19	1%	90.5	36.0 ± 12.0	89.2	75.1 ± 17	39.1	86.1	83.0 ± 3.4	47.0
		0.5%		64.2 ± 12.4		89.0 ± 0.2	24.8		85.9 ± 0.4	21.7

Note. SM: source model, used for training generators; TM: target model, used for testing generators; BER: the bit-error rate of the target model; CA (%): clean accuracy; PA (%): perturbed accuracy; NF: NeuralFuse; and RP: total recovery percentage of PA (NF) vs. PA

H NeuralFuse on Reduced-precision Quantization and Random Bit Errors

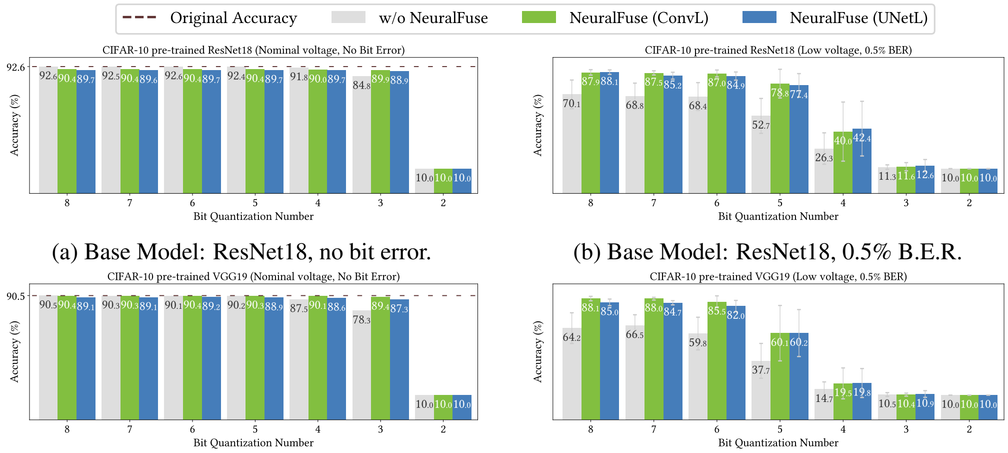
As mentioned in Sec. 4.6, we explore the robustness of NeuralFuse to low-precision quantization of model weights and consider the case of random bit errors. Here, we demonstrate that NeuralFuse can recover not only the accuracy drop due to reduced precision, but also the drop caused by low-voltage-induced bit errors (0.5% BER) under low precision. We selected two NeuralFuse generators (ConvL and UNetL) for our experiments, and these generators were trained with the corresponding base models (ResNet18 and VGG19) at 1% BER (CIFAR-10, GTSRB) and 0.5% BER (ImageNet-10). The experimental results are shown as follows: CIFAR-10 (Sec. H.1), GTSRB (Sec. H.2), and ImageNet-10 (Sec. H.3). Similarly, for ease of comparison, we visualize the experimental results in the figures below each table. Our results show that NeuralFuse can consistently perform well in low-precision regimes as well as recover the low-voltage-induced accuracy drop.

H.1 CIFAR-10

Table 22: Reduced-precision Quantization and with 0.5% BER on CIFAR-10 pre-trained models.

Base Model	#Bits	CA	PA	ConvL (1%)			UNetL (1%)		
				CA (NF)	PA (NF)	RP	CA (NF)	PA (NF)	RP
ResNet18	8	92.6	70.1 ± 11.6	89.8	89.5 ± 0.2	19.4	86.6	86.2 ± 0.3	16.1
	7	92.5	68.8 ± 10.4	89.8	89.5 ± 1.7	20.7	86.5	86.0 ± 0.5	17.2
	6	92.6	68.4 ± 11.2	89.7	89.5 ± 0.2	21.1	86.6	85.9 ± 0.3	17.5
	5	92.4	52.7 ± 14.1	89.7	90.0 ± 0.7	37.3	86.5	85.5 ± 0.8	32.8
	4	91.8	26.3 ± 12.7	89.8	58.7 ± 24.5	32.4	86.6	64.9 ± 22.5	38.6
	3	84.8	11.3 ± 1.8	89.8	12.8 ± 5.8	1.5	86.0	14.8 ± 10.0	3.5
	2	10.0	10.0 ± 0.0	10.0	10.0 ± 0.0	0.0	10.0	10.0 ± 0.0	0.0
VGG19	8	90.5	64.2 ± 12.4	89.8	89.6 ± 8.7	25.4	87.4	86.8 ± 0.4	22.6
	7	90.3	66.5 ± 8.5	89.8	89.6 ± 0.2	23.1	87.4	86.7 ± 0.3	20.2
	6	90.1	59.8 ± 13.2	89.9	89.4 ± 3.8	29.6	87.4	86.4 ± 0.7	26.6
	5	90.2	37.7 ± 14.1	89.8	78.0 ± 15.8	40.3	87.2	79.8 ± 0.8	42.1
	4	87.5	14.7 ± 6.0	89.8	27.8 ± 18.9	13.1	87.2	34.4 ± 20.5	19.7
	3	78.3	10.5 ± 1.5	89.7	10.9 ± 2.6	0.4	86.8	11.0 ± 2.9	0.5
	2	10.0	10.0 ± 0.0	10.0	10.0 ± 0.0	0.0	10.0	10.0 ± 0.0	0.0

Note. CA (%): clean accuracy; PA (%): perturbed accuracy; NF: NeuralFuse; and RP: total recovery percentage of PA (NF) vs. PA



(a) Base Model: ResNet18, no bit error.

(b) Base Model: ResNet18, 0.5% B.E.R.

(c) Base Model: VGG19, no bit error.

(d) Base Model: VGG19, 0.5% B.E.R.

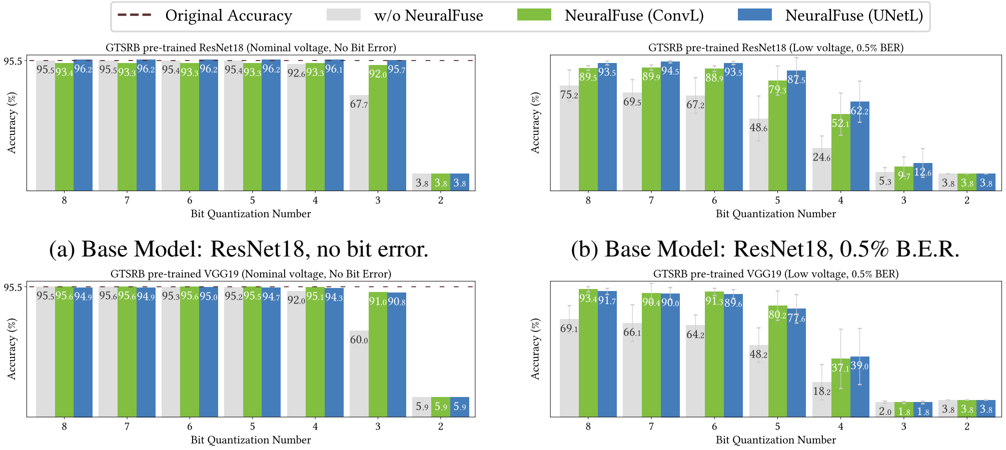
Figure 12: Results of Reduced-precision and bit errors (0.5%) on CIFAR-10 pre-trained base models.

H.2 GTSRB

Table 23: Reduced-precision Quantization and with 0.5% BER on GTSRB pre-trained models.

Base Model	#Bits	CA	PA	ConvL (1%)			UNetL (1%)		
				CA (NF)	PA (NF)	RP	CA (NF)	PA (NF)	RP
ResNet18	8	95.5	75.2 ± 12.7	95.7	95.3 ± 0.5	20.1	96.2	95.7 ± 0.3	20.5
	7	95.5	69.5 ± 10.6	95.7	95.3 ± 0.3	25.8	96.2	95.9 ± 0.3	26.4
	6	95.4	67.2 ± 14.4	95.7	95.2 ± 0.5	28.0	96.2	95.7 ± 0.5	28.5
	5	95.4	48.6 ± 18.2	95.8	92.6 ± 5.1	44.0	96.2	94.8 ± 2.5	46.2
	4	92.6	24.6 ± 9.8	95.9	75.6 ± 16.2	51.0	96.2	86.6 ± 9.5	62.0
	3	67.7	5.3 ± 3.5	95.4	18.4 ± 15.3	13.1	96.2	25.3 ± 22.5	20.0
	2	3.8	3.8 ± 0.0	4.1	3.8 ± 0.0	0.0	3.8	3.8 ± 0.0	0.0
VGG19	8	95.5	69.1 ± 11.1	96.0	94.0 ± 2.2	24.9	95.4	93.9 ± 2.1	24.8
	7	95.6	66.1 ± 14.8	96.0	92.2 ± 5.7	26.1	95.4	92.6 ± 3.7	26.5
	6	95.3	64.2 ± 8.4	96.0	92.2 ± 5.7	28.0	95.4	92.3 ± 2.3	28.1
	5	95.2	48.2 ± 14.0	96.0	92.2 ± 5.7	44.0	95.4	86.2 ± 8.4	38.0
	4	92.0	18.2 ± 14.3	93.0	92.2 ± 5.7	74.0	95.0	49.6 ± 22.8	31.4
	3	60.0	2.0 ± 0.9	87.3	92.2 ± 5.7	90.2	87.2	1.7 ± 0.9	-0.3
	2	5.9	3.8 ± 0.0	5.9	3.8 ± 0.0	0.0	5.9	3.8 ± 0.0	0.0

Note. CA (%): clean accuracy; PA (%): perturbed accuracy; NF: NeuralFuse; and RP: total recovery percentage of PA (NF) vs. PA



(a) Base Model: ResNet18, no bit error.

(b) Base Model: ResNet18, 0.5% B.E.R.

(c) Base Model: VGG19, no bit error.

(d) Base Model: VGG19, 0.5% B.E.R.

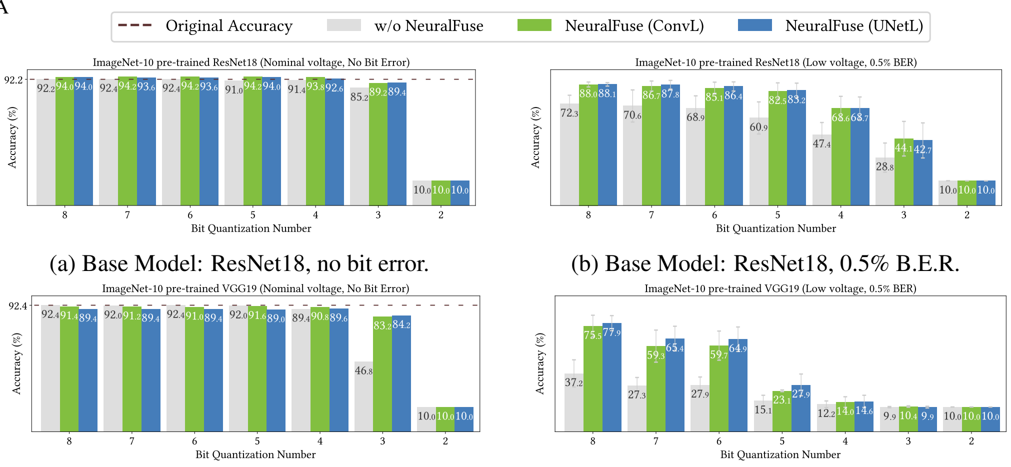
Figure 13: Results of Reduced-precision and bit errors (0.5%) on GTSRB pre-trained base models.

H.3 ImageNet-10

Table 24: Reduced-precision Quantization and with 0.5% BER on ImageNet-10 pre-trained models.

Base Model	#Bits	CA	PA	ConvL (0.5%) CA (NF)	ConvL (0.5%) PA (NF)	ConvL (0.5%) RP	UNetL (0.5%) CA (NF)	UNetL (0.5%) PA (NF)	UNetL (0.5%) RP
ResNet18	8	92.2	72.3 ± 7.0	94.0	88.0 ± 2.0	15.7	94.0	88.1 ± 1.4	15.8
	7	92.4	70.6 ± 13.0	94.2	86.7 ± 4.1	16.1	93.6	87.8 ± 3.5	17.2
	6	92.4	68.9 ± 9.9	94.2	85.1 ± 4.8	16.2	93.6	86.4 ± 3.7	17.5
	5	91.0	60.9 ± 13.0	94.2	82.5 ± 6.8	21.6	94.0	83.2 ± 5.9	22.3
	4	91.4	47.4 ± 9.8	93.8	68.6 ± 9.8	21.2	92.6	68.7 ± 9.2	21.3
	3	85.2	28.8 ± 11.8	89.2	44.1 ± 14.0	15.3	89.4	42.7 ± 14.2	13.9
	2	10.0	10.0 ± 0.0	10.0	10.0 ± 0.0	0.0	10.0	10.0 ± 0.0	0.0
VGG19	8	92.4	37.2 ± 11.0	91.4	75.5 ± 8.8	38.3	89.4	77.9 ± 6.1	40.7
	7	92.0	27.3 ± 6.6	91.2	59.3 ± 13.0	32.0	89.4	65.4 ± 10.0	38.1
	6	92.4	27.9 ± 6.4	91.0	59.7 ± 11.8	31.8	89.4	64.9 ± 9.9	37.0
	5	92.0	15.1 ± 4.4	91.6	23.1 ± 0.7	8.0	89.0	27.9 ± 8.8	12.8
	4	89.4	12.2 ± 2.7	90.8	14.0 ± 4.3	1.8	89.6	14.6 ± 4.9	2.4
	3	46.8	9.9 ± 0.5	83.2	10.4 ± 0.6	0.5	84.2	9.9 ± 0.7	0.0
	2	10.0	10.0 ± 0.0	10.0	10.0 ± 0.0	0.0	10.0	10.0 ± 0.0	0.0

Note. CA (%): clean accuracy; PA (%): perturbed accuracy; NF: NeuralFuse; and RP: total recovery percentage of PA (NF) vs. PA



(a) Base Model: ResNet18, no bit error.

(b) Base Model: ResNet18, 0.5% B.E.R.

(c) Base Model: VGG19, no bit error.

(d) Base Model: VGG19, 0.5% B.E.R.

Figure 14: Results of Reduced-precision and bit errors (0.5%) on ImageNet-10 pre-trained base models.

I Additional Experiments on Adversarial Training

Adversarial training is a common strategy to derive a robust neural network against certain perturbations. By training the generator using adversarial training proposed in Stutz et al. [2021], we report its performance against low voltage-induced bit errors. We use ConvL as the generator and ResNet18 as the base model, trained on CIFAR-10. Furthermore, we explore different K flip bits as the perturbation on weights of the base model during adversarial training, and then for evaluation, the trained-generator will be applied against 1% of bit errors rate on the base model. The results are shown in Table 25. After careful tuning of hyperparameters, we find that we are not able to obtain satisfactory recovery when adopting adversarial training. Empirically, we argue that adversarial training may not be suitable for training generator-based methods.

J Additional Experiments on Robust Model Trained with Adversarial Weight Perturbation with NeuralFuse

Previously, Wu et al. [2020] proposed that one could obtain a more robust model via adversarial weight perturbation. To seek whether such models could also be robust to random bit errors, we

Table 25: Performance of the generator trained by adversarial training under K flip bits on ResNet18 with CIFAR-10. The results show that the generator trained by adversarial training cannot achieve high accuracy against bit errors under a 1% bit error rate.

K-bits	CA	PA	CA (NF)	PA (NF)	RP
100	92.6	38.9 ± 12.4	92.4	38.3 ± 12.1	-0.6
500			92.1	38.7 ± 12.5	-0.2
5,000			92.6	38.9 ± 12.5	0
20,000			60.1	23.0 ± 8.1	-16
100,000			71.1	23.6 ± 6.6	-16

Note. CA (%): clean accuracy; PA (%): perturbed accuracy; NF: NeuralFuse; and RP: total recovery percentage of PA (NF) vs. PA

conducted an experiment on CIFAR-10 with the proposed adversarially trained PreAct ResNet18. The experimental results are shown in Table 26. We find that the average perturbed accuracy is 23% and 63.2% for PreAct ResNet18 under 1% and 0.5% BER, respectively. This result is lower than 38.9% and 70.1% from ResNet18 in Table 12, indicating their poor generalization ability against random bit errors. Nevertheless, when equipped NeuralFuse on the perturbed model, we could still witness a significant recover percentage under both 1% and 0.5% BER This result further demonstrates that NeuralFuse could be adapted to various models (i.e., trained in different learning algorithms).

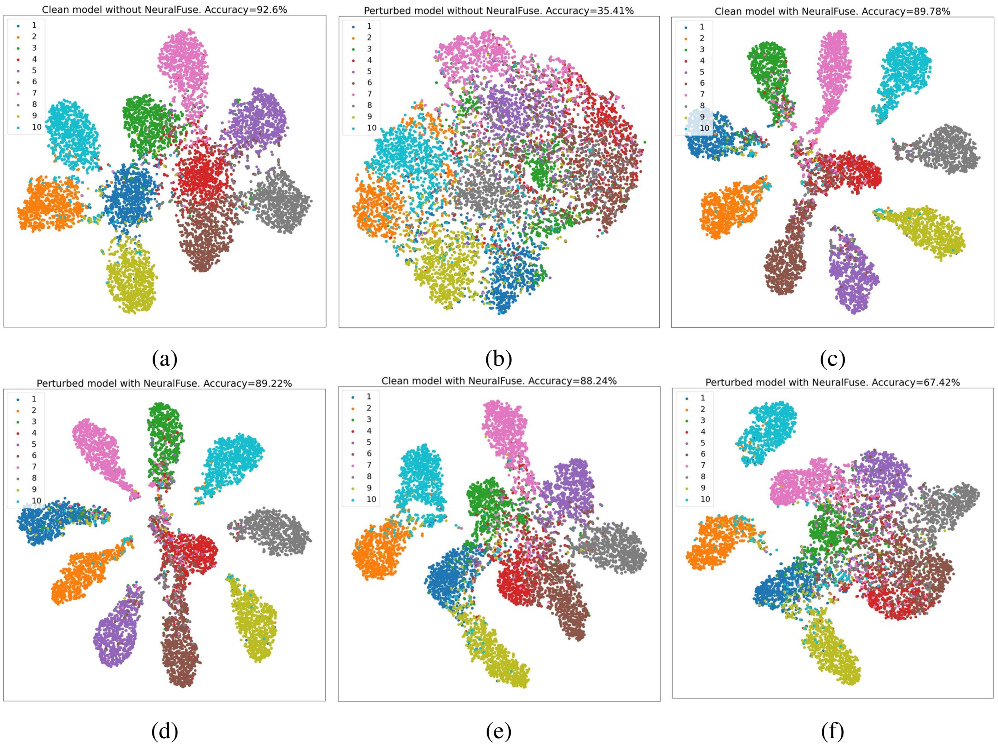
Table 26: Performance of NeuralFuse trained with rubust CIFAR-10 pre-trained PreAct ResNet18. The results show that NeuralFuse can be used together with a robust model and further improve perturbed accuracy under both 1% and 0.5% BER

Base Model	BER	NF	CA	PA	CA (NF)	PA (NF)	RP
PreAct ResNet18	1%	ConvL	89.7	23.0 ± 9.3	87.6	53.7 ± 26	30.7
		ConvS			83.1	34.6 ± 15	11.6
		DeConvL			87.7	55.4 ± 27	32.4
		DeConvS			82.9	32.4 ± 14	9.4
		UNetL			86.1	60.4 ± 28	37.4
		UNetS			80.4	51.9 ± 24	28.9
	0.5%	ConvL	89.7	63.2 ± 8.7	89.2	87.8 ± 1.1	24.6
		ConvS			89.2	74.0 ± 6.5	10.8
		DeConvL			89.0	87.4 ± 1.1	24.2
		DeConvS			89.9	74.4 ± 7.0	11.2
		UNetL			87.5	85.9 ± 0.8	22.7
		UNetS			88.2	80.4 ± 3.9	17.2

Note. BER: the bit-error rate of the base model; CA (%): clean accuracy; PA (%): perturbed accuracy; NF: NeuralFuse; and RP: total recovery percentage of PA (NF) vs. PA

K Data Embeddings Visualization

To further understand how our proposed NeuralFuse works, we visualize the output distribution from the final linear layer of the base models and project the results onto the 2D space using t-SNE [van der Maaten and Hinton, 2008]. Figure 15 shows the output distribution from ResNet18 (trained on CIFAR-10) under a 1% bit error rate. We chose two generators that have similar architecture: ConvL and ConvS, for this experiment. We can observe that: (a) The output distribution of the clean model without NeuralFuse can be grouped into 10 classes denoted by different colors. (b) The output distribution of the perturbed model under a 1% bit error rate without NeuralFuse shows mixed representations and therefore degraded accuracy. (c) The output distribution of the clean model with ConvL shows that applying NeuralFuse will not hurt the prediction of the clean model too much (i.e., it retains high accuracy in the regular voltage setting). (d) The output distribution of the perturbed model with ConvL shows high separability (and therefore high perturbed accuracy) as opposed to (b). (e)/(f) shows the output distribution of the clean/perturbed model with ConvS. For both (e) and (f), we can see nosier clustering when compared to (c) and (d), which means the degraded performance of ConvS compared to ConvL. The visualization validates that NeuralFuse can help retain good data representations under random bit errors and that larger generators in NeuralFuse have better performance than smaller ones.


Clean model without NeuralFuse. Accuracy=92.6%

Perturbed model without NeuralFuse. Accuracy=35.41%

Clean model with NeuralFuse. Accuracy=89.78%

(a)

(b)

(c)

Perturbed model with NeuralFuse. Accuracy=89.22%

Clean model with NeuralFuse. Accuracy=88.24%

Perturbed model with NeuralFuse. Accuracy=67.42%

(d)

(e)

(f)

Figure 15: t-SNE results for ResNet18 trained by CIFAR-10 under 1% of bit error rate. (a) Clean model. (b) Perturbed model. (c) Clean model with ConvL. (d) Perturbed model with ConvL. (e) Clean model with ConvS. (f) Perturbed model with ConvS.

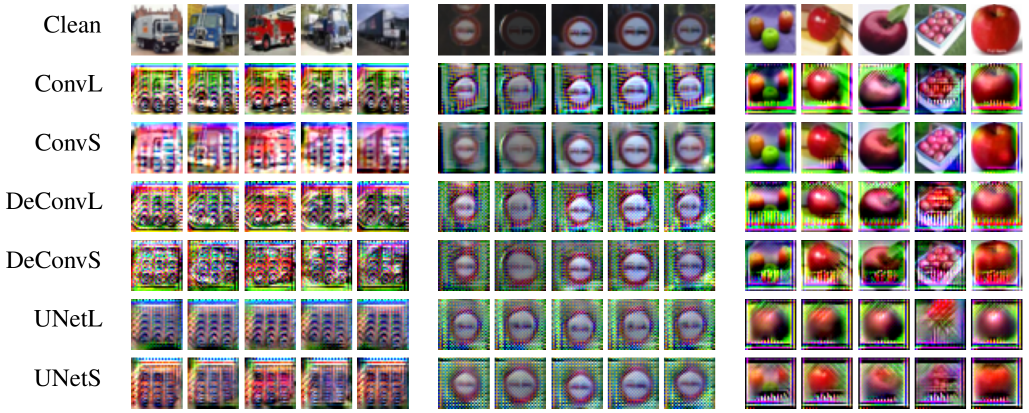
L Qualitative Analysis of Transformed Inputs

In this section, we conduct a qualitative study to visualize the images transformed by NeuralFuse and present some properties and observations of these images. We utilize six different architectures of NeuralFuse generators trained with ResNet18 under a 1% bit error rate.

Figure 16 (a) showcases several images from the *truck* class in CIFAR-10. Notably, images of the same class, when transformed by the same NeuralFuse, exhibit similar patterns, such as circles symbolizing the wheels of the trucks.

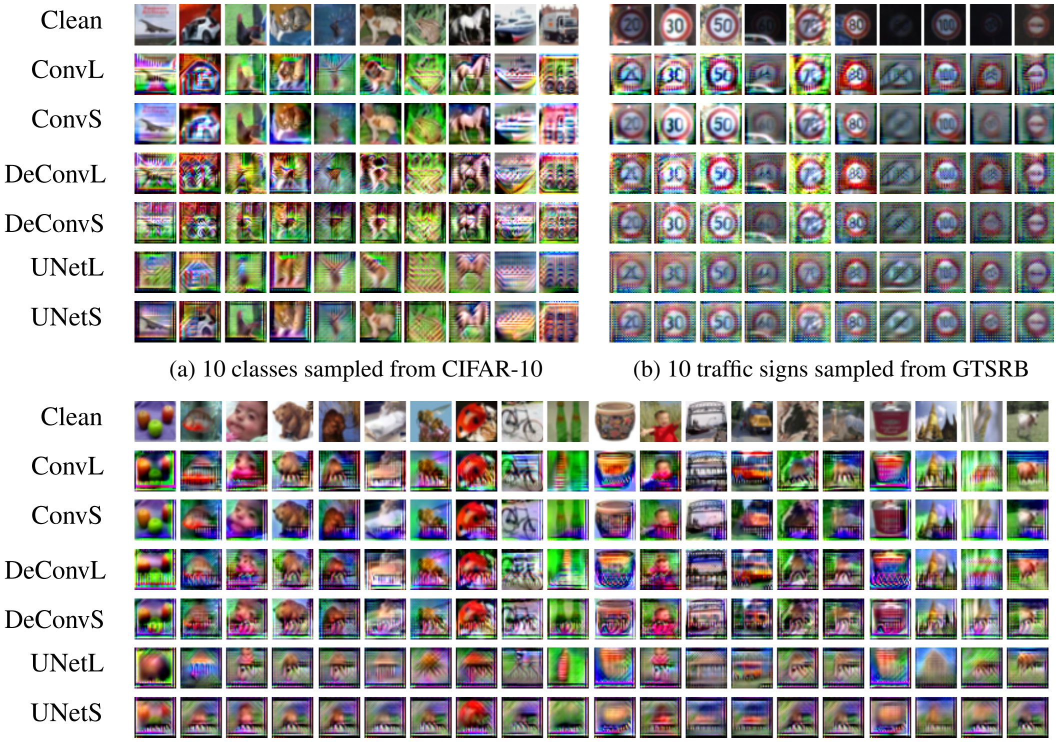
In Figures 16 (b) and 16 (c), we observe analogous phenomena in the GTSRB and CIFAR-100 datasets. Transformed images of the same class using the same generator consistently display patterns. On GTSRB, NeuralFuse-generated patterns highlight the sign's shape with a green background, even if the original images have a dark background and are under different lighting conditions. These results further underscore the efficacy and efficiency of NeuralFuse.

Figure 17 presents more images from different classes in (a) CIFAR-10, (b) GTSRB, and (c) CIFAR-100. The transformed images exhibit distinct patterns for each class, suggesting that NeuralFuse effectively transforms images into class-specific patterns, making associated features robust to random bit errors and easily recognizable by the base model in low-voltage settings.



(a) *Truck* class in CIFAR-10. (b) *No Passing* sign in GTSRB. (c) *Apple* class in CIFAR-100.

Figure 16: Visualization of transformed images from different NeuralFuse generators trained with ResNet18 at 1% bit error rate.



(a) 10 classes sampled from CIFAR-10 (b) 10 traffic signs sampled from GTSRB

(c) 20 classes sampled from CIFAR-100

Figure 17: Visualization of transformed images from different NeuralFuse generators trained by ResNet18 with 1% bit error rate.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We list our main contributions in the Abstract and Introduction.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discussed the limitation of our work on runtime/latency in Section 4.4.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: This paper does not include theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The implementation details are presented in Appendices A and B.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Our code can be found at `https://github.com/IBM/NeuralFuse`.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental details are presented in Section 4.1, Appendix and our code.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: For each experimental result, we report the mean and standard deviation and plot the error bars.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The computing resources are presented in Section 4.1.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work stated in this paper conform with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discussed broader impacts in Section 5.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: This paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have cited the original paper that produced the code package or dataset.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We described how to run our code in the `README.md`.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.