NeuralFuse: Improving the Accuracy of Access-Limited Neural Network Inference in Low-Voltage Regimes

Hao-Lun Sun[1] **Lei Hsiung**[1,2] **Nandhini Chandramoorthy**[2] **Pin-Yu Chen**[2] **Tsung-Yi Ho**[1,3]
[1]National Tsing Hua University [2]IBM T. J. Watson Research Center
[3]The Chinese University of Hong Kong
s109062594@m109.nthu.edu.tw,
{lei.hsiung, nandhini.chandramoorthy, pin-yu.chen}@ibm.com,
tyho@cse.cuhk.edu.hk

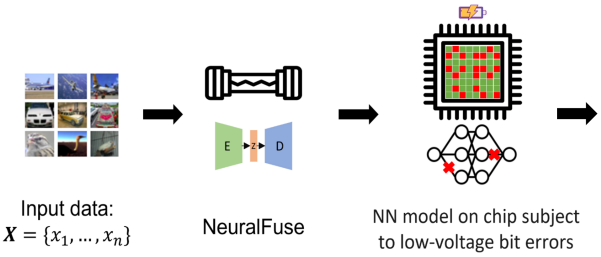
Abstract

Deep neural networks (DNNs) are state-of-the-art models adopted in many machine learning based systems and algorithms. However, a notable issue of DNNs is their considerable energy consumption for training and inference. At the hardware level, one current energy-saving solution at the inference phase is to reduce the voltage supplied to the DNN hardware accelerator. However, operating in the low-voltage regime would induce random bit errors saved in the memory and thereby degrade the model performance. To address this challenge, we propose **NeuralFuse**, a novel input transformation technique as an add-on module, to protect the model from severe accuracy drops in low-voltage regimes. With NeuralFuse, we can mitigate the tradeoff between energy and accuracy without retraining the model, and it can be readily applied to DNNs with limited access, such as DNNs on non-configurable hardware or remote access to cloud-based APIs. Compared with unprotected DNNs, our experimental results show that NeuralFuse can reduce memory access energy up to 24% and simultaneously improve the accuracy in low-voltage regimes up to an increase of 57%. To the best of our knowledge, this is the first model-agnostic approach (i.e., no model retraining) to mitigate the accuracy-energy tradeoff in low-voltage regimes.

1 Introduction

Energy-efficient computing is a primary consideration for the deployment of Deep Neural Networks (DNNs), particularly on edge devices and on-chip AI systems. Increasing energy efficiency and lowering the carbon footprint of DNN computation involves iterative efforts from both chip designers and algorithm developers. Processors with specialized hardware accelerators for AI computing are now ubiquitous, capable of providing orders of magnitude more performance and energy efficiency for AI computation. In addition to reduced precision/quantization and architectural optimizations, low voltage operation is a powerful knob that impacts power consumption. There is ample evidence in computer engineering literature that study the effects of undervolting and low-voltage operation on accelerator memories that store weights and activations during computation. Aggressively scaling down the SRAM (Static Random Access Memory) supply voltage below the rated value leads to an exponential increase in bit failures, but saves power on account of the quadratic dependence of dynamic power on voltage. Such memory bit flips in the stored weight and activation values can cause catastrophic accuracy loss. A recent spate of works advocates low voltage operation of DNN accelerators using numerous techniques to preserve accuracy ranging from hardware-based error mitigation techniques [1, 2] to error-aware robust training of DNN models [3–5]. On-chip error

36th Conference on Neural Information Processing Systems (NeurIPS 2022) Workshop on ML for Systems.



	Clean Accuracy	Perturbed Accuracy
Without NeuralFuse	92.6%	38.9%
With NeuralFuse	89.8%	87.8%

Figure 1: Pipeline of our proposed NeuralFuse framework. Given a data sample $\mathbf{x}$, NerualFuse transforms the input and then passes it to the access-limited DNN. Clean/Perturbed accuracy means the model is operating in regular/low-voltage mode. The latter mode incurs random bit errors (perturbations) to the deployed DNN model.

mitigation methods have significant performance and power overheads. The learning algorithms in [3–5] generate models which are more robust against bit errors, eliminating the need for on-chip error mitigation. However, error-aware training to find the optimal set of robust parameters for each model is time and energy-intensive, and may not be possible at all in access-limited settings.

In this paper, we propose a novel model-agnostic approach: *NeuralFuse*. NeuralFuse allows for mitigating bit errors caused by very low voltage operation, through a trainable input transformation parameterized by a relatively small DNN, to enhance the robustness of the original input and provide accurate inference. The pipeline of NeuralFuse is illustrated in Figure 1. NeuralFuse accepts the scenarios under access-limited neural networks (e.g., non-configurable hardware or remote access to cloud-based APIs) to protect the deployed models from making wrong predictions under low power. Specifically, we consider two settings: (a) *Relaxed Access*: the model details are unknown (e.g., packed as APIs) but backpropagating through the black-box models is possible. In this regard, we train NeuralFuse via backpropagation, also called the gradient-based method. (b) *Restricted Access*: models are unknown and backpropagation is disallowed. Therefore, we can only query the models to get the outputs. In this case, we consider training NeuralFuse on a white-box surrogate model and transferring it to the restricted access models. To the best of our knowledge, this is the first study to address random bit errors for improving accuracy in low-voltage and access-limited settings.

Our **main contributions** are as follows:

- We propose *NeuralFuse*, a novel input transformation framework to enhance the accuracy of DNNs subject to random bit errors caused by undervolting. NeuralFuse is model-agnostic because it is an add-on module and it does not require re-training the deployed DNN model.
- We consider two practical access-limited scenarios: Relaxed Access and Restricted Access. In the former setting, we use gradient-based methods to train the NeuralFuse module. In the latter setting, we use a white-box surrogate model to train NeuralFuse and show its high transferability to other types of DNN architectures.
- We conduct extensive experiments on various combinations of DNN models (ResNet18, ResNet50, VGG11, VGG16, and VGG19), datasets (CIFAR-10, CIFAR-100, and GTSRB), and NeuralFuse implementations with different architectures and sizes (ConvL, ConvS, DeConvL, DeConvS, UNetL, and UNetS). The results show that NeuralFuse can consistently increase the perturbed accuracy (accuracy under random bit errors in weights) by up to 57% while simultaneously save the overall SRAM memory access energy by up to 24% based on the realistic characterization of bit cell failures for a given memory array in a low-voltage regime inducing 1% of bit error rate.

2 NeuralFuse: Methodology and Algorithms

2.1 Error-Resistant Input Transformation

As illustrated in Figure 1, to overcome the drawback of performance degradation in low-voltage settings for access-limited DNNs, we propose a novel trainable input transformation function parametrized by a relatively small DNN, called *NeuralFuse*, to mitigate the trade-off in accuracy and

energy for model inference. The rationale is to use a designed loss function and training algorithm to train NeuralFuse, and then apply NeuralFuse to the input data such that the transformed inputs will become robust against random bit errors induced by undervolting.

Consider an input data sample $\mathbf{x}$ sampled from the data distribution $\mathcal{X}$ and a model M_p with random bit errors on its weights (which is called a perturbed model). When there are no bit errors (i.e., the normal-voltage settings), the perturbed model reduces a nominal deterministic model denoted by M_0. NeuralFuse aims to ensure the perturbed model M_p can make correct results on the transformed inputs as well as retain consistent results of M_0 in regular (normal-voltage) settings.

To adapt to different data characteristics, the input transformation function is designed to be input-aware, which is formally defined as:

$$\mathcal{F}(\mathbf{x}) = \mathbf{x} + \mathcal{G}(\mathbf{x})$$

where $\mathcal{G}(\mathbf{x})$ is a "generator" (i.e., an input transformation function) that can generate a perturbation based on the input $\mathbf{x}$. The transformed inputs $\mathcal{F}(\mathbf{x})$ by NeuralFuse will be taken as the input to a deployed model (either 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 dimension of $\mathbf{x}$.

2.2 Training Objective and Optimizer

To train the generator $\mathcal{G}(\cdot)$, which should ensure the correctness of both the perturbed model M_p and the clean model M_0, we design the following training objective function:

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

subject to $\mathcal{F}(\cdot) \in [-1, 1]^d$, 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 computed by a model M 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 $p\%$ of random bit error rate, and λ is the hyperparameter that we can tune to balance the importance between the clean model and the perturbed model.

The training objective function can be readily converted to a loss function (*loss*) that evaluates the cross-entropy between the groundtruth label y and the prediction $\mathcal{P}_M(y|\mathcal{F}(\mathbf{x}; \mathcal{W}_\mathcal{G})$. The total loss function becomes

$$Loss_{Total} = loss_{M_0} + \lambda \cdot loss_{\mathcal{M}_p}. \tag{1}$$

To optimize the loss function entailing the evaluation of the loss term $loss_{\mathcal{M}_p}$ on randomly perturbed models, our training process is inspired by the Expectation Over Transformation Attacks (EOT attacks) [6]. EOT attacks aim to find a robust adversarial example against multiple image transformations. Based on the idea, we propose a new optimizer for solving (1), which we call Expectation Over Perturbed Models (EOPM). EOPM-trained generators can generate error-resistant input transformations and help mitigate the inherent bit errors. However, it is impossible to enumerate all possible perturbed models with random bit errors, and the number of realizations for perturbed models is limited by the memory constraint of GPUs used for training. In practice, we only take N perturbed models for each iteration to calculate the empirical average loss. That is:

$$Loss_N = \frac{loss_{M_{p_1}} + ... + loss_{M_{p_N}}}{N},$$

where N is the number of simulated perturbed models $\{M_{p_1}, \ldots, M_{p_N}\}$ under random bit errors to calculate the loss. Based on the empirical loss, we can calculate the gradients for updating the generator as:

$$\frac{\partial Loss_{Total}}{\partial \mathcal{W}_\mathcal{G}} = \frac{\partial loss_{M_0}}{\partial \mathcal{W}_\mathcal{G}} + \frac{\lambda}{N} \cdot \left(\frac{\partial loss_{M_{p_1}}}{\partial \mathcal{W}_\mathcal{G}} + ... + \frac{\partial loss_{M_{p_N}}}{\partial \mathcal{W}_\mathcal{G}} \right).$$

In our implementation, we find that using $N = 10$ can already deliver stable performance, and there is little gain in using a larger value. Please refer to Appendix F for details.

3 Experiments

In this section, we reveal our experimental results trained by the EOPM algorithm for NeuralFuse under two scenarios (relaxed/restricted access). The details of the experimental settings are appeared in Appendix C. In addition, we also provide the visualization results and further analysis to better understand the properties of NeuralFuse in Appendix E and F, respectively.

3.1 Performance Evaluation on Relaxed Access

The experimental results of the Relaxed Access setting are shown in Table 1. We train and test NeuralFuse with ResNet18, ResNet50, VGG11, VGG16, and VGG19 as based models under the 1% bit error rate. For each experiment, we sample $N = 10$ perturbed models for evaluation and report the mean accuracy and standard deviation of the test accuracy, which are not used in the training process.

Table 1: Testing accuracy (%) under 1% of random bit error rate. Notations: B.M. (based model), C.A. (clean accuracy), P.A. (perturbed accuracy), N.F. (NeuralFuse), and IP (total improvement of P.A.+N.F. *v.s.* P.A.).

B.M.	N.F.	CIFAR-10					GTSRB					CIFAR-100				
		C.A.	P.A.	C.A.+N.F.	P.A.+N.F.	IP	C.A.	P.A.	C.A.+N.F.	P.A.+N.F.	IP	C.A.	P.A.	C.A.+N.F.	P.A.+N.F.	IP
ResNet18	ConvL	92.6	38.9 ± 12.4	89.8	87.8 ± 1.7	48.8	95.5	36.9 ± 16.0	95.7	91.1 ± 4.7	54.2	73.7	4.6 ± 2.9	54.8	11.0 ± 7.7	6.4
	ConvS			88.2	59.5 ± 11	20.6			94.4	68.6 ± 12	31.7			49.7	4.2 ± 2.2	-0.4
	DeConvL			89.6	88.5 ± 0.8	49.6			95.6	91.3 ± 4.3	54.4			55.2	11.9 ± 8.2	7.3
	DeConvS			82.9	68.8 ± 6.4	29.9			95.7	78.1 ± 9.1	41.2			32.7	4.0 ± 2.2	-0.6
	UNetL			86.6	84.6 ± 0.8	45.6			96.2	93.8 ± 1.0	56.9			50.6	14.5 ± 8.9	10.0
	UNetS			84.4	68.8 ± 6.0	29.8			95.9	85.1 ± 6.9	48.2			26.8	4.6 ± 2.5	-0.0
ResNet50	ConvL	92.6	26.1 ± 9.4	85.5	53.2 ± 22	27.1	95.0	29.5 ± 16.9	95.6	71.6 ± 20	42.1	73.5	3.0 ± 1.8	63.5	3.2 ± 1.7	0.1
	ConvS			85.2	34.6 ± 14	8.5			94.8	50.5 ± 22	21.0			65.5	3.2 ± 1.6	0.1
	DeConvL			87.4	63.3 ± 21	37.2			94.9	71.6 ± 21	42.0			59.6	3.2 ± 1.7	0.2
	DeConvS			82.4	42.2 ± 17	16.1			93.0	56.4 ± 17	26.9			61.1	3.2 ± 1.7	0.1
	UNetL			86.2	75.5 ± 12	49.4			94.5	80.6 ± 15	51.1			39.0	5.0 ± 1.7	1.9
	UNetS			77.3	56.2 ± 19	30.1			94.7	64.7 ± 22	35.2			47.7	3.4 ± 1.8	0.3
VGG11	ConvL	88.4	42.2 ± 11.6	89.6	87.2 ± 2.9	45.1	91.9	34.9 ± 12.4	94.8	85.7 ± 7.2	50.9	64.8	8.2 ± 5.7	58.3	19.7 ± 11	11.5
	ConvS			84.9	66.3 ± 7.5	24.1			91.1	62.2 ± 11	27.3			56.6	10.4 ± 7.4	2.2
	DeConvL			89.3	87.2 ± 2.6	45.0			95.0	84.6 ± 7.6	49.7			82.3	21.2 ± 11	13.0
	DeConvS			85.6	68.2 ± 7.1	26.0			92.4	67.5 ± 11	32.6			58.3	11.8 ± 7.9	3.5
	UNetL			87.1	83.6 ± 1.3	41.4			92.2	83.2 ± 6.0	48.3			51.1	22.1 ± 8.2	13.9
	UNetS			85.5	72.7 ± 4.6	30.5			94.7	73.4 ± 10	38.5			51.9	13.1 ± 7.9	4.9
VGG16	ConvL	90.3	35.7 ± 7.9	90.1	86.0 ± 6.2	50.3	95.2	15.1 ± 6.8	96.3	72.4 ± 12	57.3	67.8	7.0 ± 3.5	51.4	19.2 ± 6.0	12.6
	ConvS			87.4	59.6 ± 12	23.9			94.1	39.8 ± 13	24.6			44.3	6.7 ± 2.3	0.1
	DeConvL			89.7	85.5 ± 6.8	49.8			96.4	72.0 ± 12	56.9			53.1	20.8 ± 6.2	14.2
	DeConvS			86.8	66.5 ± 11	30.8			93.8	50.9 ± 13	35.8			23.5	4.8 ± 1.7	-1.8
	UNetL			87.4	83.4 ± 4.4	47.7			95.8	78.6 ± 11	63.5			50.2	25.3 ± 1.7	18.7
	UNetS			87.4	71.2 ± 8.2	35.5			94.3	63.3 ± 14	48.1			27.7	9.9 ± 2.1	3.3
VGG19	ConvL	90.5	36.0 ± 12	89.8	77.7 ± 19	41.7	95.5	36.6 ± 6.8	96.0	88.3 ± 7.2	51.7	67.8	10.6 ± 4.3	59.4	29.2 ± 8.1	18.6
	ConvS			87.3	52.7 ± 17	16.7			93.8	69.0 ± 14	32.4			63.7	14.4 ± 5.1	3.8
	DeConvL			86.3	78.4 ± 18	42.4			95.4	87.2 ± 7.5	50.6			60.1	29.6 ± 8.5	19.0
	DeConvS			86.5	58.2 ± 18	22.2			94.5	73.1 ± 12	36.5			60.9	16.1 ± 6.0	5.6
	UNetL			86.3	82.1 ± 4.8	46.0			95.4	88.2 ± 6.7	51.7			58.7	30.2 ± 8.2	19.6
	UNetS			86.3	66.4 ± 13	30.4			94.6	80.6 ± 9.0	44.1			59.1	18.0 ± 6.2	7.4

For CIFAR-10 and GTSRB, we observe that large generators like ConvL and UNetL can significantly improve the perturbed accuracy in the range of 41% to 63% on ResNet18, VGG11, VGG16, and VGG19. For ResNet50, the improvement is slightly worse than other base models, but it can attain up to 51% improvement on GTSRB. On the other hand, the improvements based on small generators like DeConvS are worse than larger generators. This can be explained by the better ability to learn error-resistant generators for larger-sized networks (though they may consume more energy). Moreover, for CIFAR-100, we can find that for both large and small generators, the gains are less compared to the other two datasets. We believe this is because CIFAR-100 is a more difficult dataset (more classes) for the generators to learn to protect the base models.

3.2 Transferability for Restricted Access

To test the performance of the Restricted Access scenario, we train NeuralFuse on a white-box surrogate model and transfer it to other black-box base models. The experimental results are shown in Table 2. We adopt ResNet18 and VGG19 as the white-box surrogate (source) models for training the generators under 1.5% of bit error rate (B.E.R.). For the generators, we choose ConvL and UNetL because they have outstanding performance in Table 1.

In Table 2, we can find that transferring from a larger B.E.R. (1.5%) can give strong resilience to a smaller B.E.R. (1% or 0.5%). We also find that using VGG19 as a surrogate model with UNet-Based generators like UNetL can give better improvements than other combinations. On the other hand, in some cases, we observe that if we transfer between the same source and target models (but with

Table 2: Transferred results trained by 1.5% of random bit error rate on CIFAR-10. Notations: S.M. (source model, used for training generators), T.M. (target model, used for testing generators), B.E.R (the bit error rate of the target model), C.A. (clean accuracy, %), P.A. (perturbed accuracy, %), N.F. (NeuralFuse), and IP (total improvement of P.A.+N.F. *v.s.* P.A.).

S.M.	T.M.	B.E.R.	ConvL			UNetL		
			C.A.+N.F.	P.A.+N.F.	IP	C.A.+N.F.	P.A.+N.F.	IP
ResNet18	ResNet18	1%	89.8	89.0 ± 0.5	50.1	85.8	85.2 ± 0.5	46.3
		0.5%		89.6 ± 0.2	19.6		85.7 ± 0.2	15.6
	ResNet50	1%	89.2	36.1 ± 18	10.0	84.4	38.9 ± 16	12.8
		0.5%		74.1 ± 10	13.1		72.7 ± 4.6	11.7
	VGG11	1%	86.3	59.2 ± 10	17.0	82.3	69.8 ± 7.5	27.6
		0.5%		78.9 ± 4.9	15.2		77.0 ± 4.0	13.4
	VGG16	1%	89.4	62.2 ± 18	26.5	84.7	68.9 ± 14	33.1
		0.5%		83.4 ± 5.5	16.8		80.5 ± 5.9	13.9
	VGG19	1%	89.8	49.9 ± 23	13.9	85.0	55.1 ± 17	19.1
		0.5%		81.8 ± 8.5	17.6		78.5 ± 6.8	14.3
VGG19	ResNet18	1%	88.9	62.6 ± 13	23.7	85.0	72.3 ± 11	33.3
		0.5%		84.2 ± 7.2	14.1		82.1 ± 2.2	12.0
	ResNet50	1%	88.8	37.9 ± 18	11.8	85.2	46.7 ± 17	20.5
		0.5%		76.6 ± 7.8	15.6		78.3 ± 3.7	17.3
	VGG11	1%	88.9	76.0 ± 6.1	33.9	85.5	81.9 ± 3.9	39.7
		0.5%		85.9 ± 2.6	22.3		84.8 ± 0.5	21.2
	VGG16	1%	89.0	76.5 ± 9.0	40.8	85.9	79.2 ± 7.5	43.5
		0.5%		87.7 ± 0.7	21.1		84.7 ± 0.9	18.1
	VGG19	1%	89.1	80.2 ± 12	44.2	86.3	84.3 ± 1.2	48.3
		0.5%		88.8 ± 0.4	24.6		85.9 ± 0.3	21.7

different B.E.R for training and testing), the performance may outperform the original relaxed-access results. For example, when transferring VGG19 with UNetL under 1.5% B.E.R. to VGG19 or VGG11 under 0.5% B.E.R., the results would be 85.9% compared to 85.0% for VGG19 (original), and 84.8% compared to 82.4% for VGG11 (original), respectively. We conjecture that the generators trained on a larger B.E.R. can actually cover the error patterns of a smaller B.E.R., and even help improve generalization under a smaller B.E.R. Consequently, these findings show great promise for improving the accuracy of access-limited-based models in low-voltage settings.

4 Conclusion

In this paper, we proposed *NeuralFuse*, the first non-intrusive post-hoc protection module for model inference against bit errors induced by low voltage. NeuralFuse is particularly suited to practical machine deployment settings involving relaxed or limited access to the base model. The design of NeuralFuse includes a novel loss function and a new optimizer EOPM for handling simulated randomness in perturbed models. Our comprehensive experimental results and analysis show that NeuralFuse can greatly improve the test accuracy (up to 57% increase) while simultaneously enjoying up to 24% reduction in memory access energy). NeuralFuse also demonstrates high transferability to access-restricted models. NeuralFuse provides significant improvements in mitigating the energy-accuracy tradeoff of neural network inference in low-voltage regimes and sheds new insights on green AI technology.

Acknowledgments

Part of this work was done during Lei Hsiung's visit to IBM T. J. Watson Research Center. We thank National Center for High-performance Computing (NCHC) in Taiwan for providing computational and storage resources.

References

[1] 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.

[2] 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.

[3] 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. URL `https://proceedings.mlsys.org/paper/2021/file/a684eceee76fc522773286a895bc8436-Paper.pdf`.

[4] 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.

[5] 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.

[6] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *International conference on machine learning*, pages 284–293. PMLR, 2018.

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

[8] 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*, pages 4820–4828, 2016.

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

[10] 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*, pages 5687–5695, 2017.

[11] 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*, pages 11206–11215, 2019.

[12] 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*, 2019.

[13] 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.

[14] 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.

[15] Shrikanth Ganapathy, John Kalamatianos, Keith Kasprak, and Steven Raasch. On characterizing near-threshold sram failures in finfet technology. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2017. doi: 10.1145/3061639.3062292.

[16] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[17] 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.

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

[19] 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.

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

[21] 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.

[22] 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.

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

Appendix

A Related Work and Background

A.1 Software Based Energy Saving Strategies

Recent studies have proposed reducing energy consumption from a software perspective. For instance, the quantization techniques reduce the precision of storing the model weights and therefore reduce the total memory storage [7–9]. [10] proposes Energy-Aware pruning on each layer and then finetuned the weights to maximize the final accuracy. [11] proposes ECC, a DNN compression framework, that compresses a DNN model to meet the given energy constraint, and [12] serves the energy constraint as the optimization problem during DNN training to reduce the energy consumption and maximize the training accuracy. [13] considers joint pruning and quantization of the DNN models for compression. However, these methods focus on changing either the model architectures or model weights to reduce the energy consumption, which are orthogonal to our NeuralFuse that serves as an add-on module at the data input to any given model.

A.2 Hardware Based Energy Saving Strategies

Existing works have studied improving energy efficiency by designing specific hardware. Several works study undervolting of the DNN accelerators and propose methods to preserve accuracy with bit errors. Minerva [1] proposes an SRAM fault mitigation technique that rounds the faulty weights into zeros to avoid the degradation of the prediction accuracy. [14] proposes to store the sensitive MSBs in robust SRAM cells to preserve accuracy. [2] proposes dynamic supply voltage boosting to a higher voltage to improve the resilience of the memory access operation. [3] considers a learning-based method that tries to find robust models against the bit errors. The paper discusses several techniques to improve the robustness, such as different quantization methods, weight clipping, random bit error training, and adversarial bit error training. The authors observe that the combination of quantization, weight clipping, and adversarial bit error training achieves excellent performance in their experiments. However, the authors have mentioned that the training process they proposed is very sensitive to the settings of hyperparameters and hence it might be difficult to train. We argue that the methods mentioned above are not easy to implement or not suitable for real-world scenarios in access-limited settings. For example, the weights of DNN models packed on embedded systems may not be configurable or updatable. Therefore, model retraining such as in [3] is not a viable option. Moreover, model training of DNNs is already a tedious and time-consuming task. Adding error-aware training during training may further increase the training complexity and introduce challenges in hyperparameter search as identified in [3]. Our proposed *NeuralFuse* spares the need for model retraining by attaching a trainable input transformation function parameterized by a relatively small DNN as an add-on module to any DNN model as is.

A.3 Memory bit errors

Low-voltage-induced memory bit cell failures cause bit flips from 0 to 1 and vice-versa. As supply voltage is scaled down below V_{min}, the minimum rated voltage at which no bit flips are guaranteed to occur, SRAM memory bit errors exponentially increase. This phenomenon is well-studied in literature [2, 15]. Figure 2 illustrates the increase in bit errors when voltage is scaled down for an SRAM array of size 512x64 bits and 14nm technology node. The corresponding dynamic energy per reading access of the SRAM is shown in the figure on the right, where energy is measured for each voltage at a constant frequency. In both cases, the supply voltages are normalized to V_{min}, the lowest voltage at which there are no measured bit errors for this SRAM. For example, compared to V_{min}, accessing the SRAM at $0.83V_{min}$ leads to a 1% bit error rate. However, dynamic energy per access is reduced by approximately 30%. Accelerator memories store DNN weights, and bit errors, particularly at most significant bits, can lead to unacceptable losses in test accuracy. However, improving the robustness to bit errors allows us to lower the voltage and exploit the energy savings that come with it. Such bit cell failures for a given memory array are randomly distributed and independent of each other. The spatial distribution of bit errors is usually different across different arrays, even within a chip as well as different chips, and can be assumed random. In this work, we simulate bit errors in a memory array of a given size, similar to [2], by generating a random distribution of bit cell failures, resulting in the equal likelihood of 0-to-1 and 1-to-0 bit flips. Weights are quantized to N=8

bit precision by symmetrically mapping to the range $(-2^{N-1}-1, 2^{N-1}-1)$. Randomly distributed bit errors are then injected in the quantized 2's complement representation of weights to generate perturbed models.

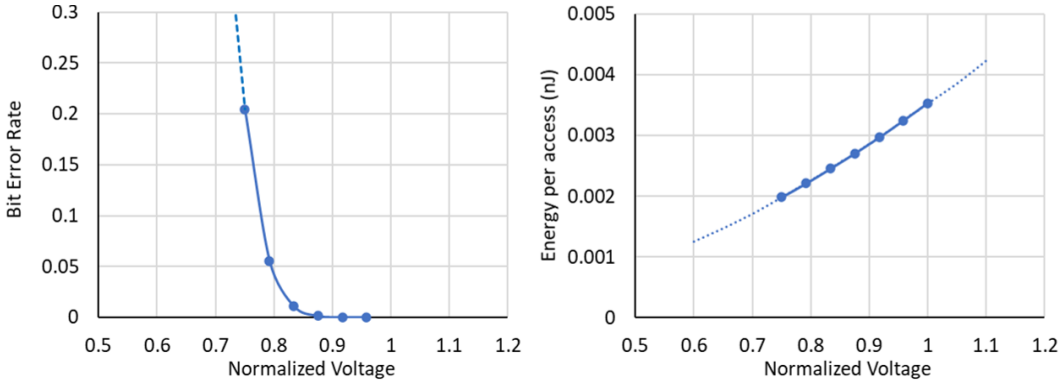


Figure 2: (left) Bit error rate (right) Dynamic energy per memory access versus voltage. x-axis shows voltages normalized with respect to the lowest bit error-free voltage

B Training Algorithm for NeuralFuse

Algorithm 1 summarizes our training steps for NeuralFuse. We split 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 get the transformed inputs. Also, we simulate N perturbed models using $p\%$ random bit error rate, denoted by $M_{p_1}, ..., M_{p_N}$, from $\mathcal{M}_p$. Then, the transformed inputs are fed into these N perturbed models and the clean model M_0 to calculate their losses and gradients. Finally, the NeuralFuse parameters $\mathcal{W}_\mathcal{G}$ are updated based on the gradient obtained by EOPM.

Algorithm 1 Training steps for NeuralFuse

Input: Based 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 $\mathbf{x}_t = \mathcal{F}(\mathbf{x}) = \mathbf{x} + \mathcal{G}(\mathbf{x})$.
    Sample N perturbed models $\{M_{p_1}, ..., M_{p_N}\}$ from $\mathcal{M}_p$ under $p\%$ random bit error rate.
    for all $M_{pi} \sim \{M_{p_1}, ..., M_{p_N}\}$ **do**
      Calculate the loss $loss_{pi}$ based on the output of the perturbed model M_{pi}. Then calculate the gradients g_{pi} for $\mathcal{W}_\mathcal{G}$ based on $loss_{pi}$.
    end for
    Calculate the loss $loss_0$ based on the output of the clean model M_0. Then calculate the gradients g_0 for $\mathcal{W}_\mathcal{G}$ based on $loss_0$.
    Calculate the mean gradients g_{mean} based on g_0 and $g_{p1}...g_{pN}$.
    Update $\mathcal{W}_\mathcal{G}$ by using g_{mean} defined in Equation (1).
  end for
end for

C Experiment Setups

Datasets: We evaluate NeuralFuse on three different datasets: CIFAR-10 [16], CIFAR-100 [16], and GTSRB [17]. CIFAR-10 consists of 60000 32*32*3 images. There are 10 classes for CIFAR-10 with 50000 images for training and another 10000 for testing. CIFAR-100 is a dataset similar to CIFAR-10, but it has 100 classes. It contains 60000 32*32*3 images. For each class, there have 500

images for training and 100 for testing. GTSRB (German Traffic Sign Recognition Benchmark) is a dataset that contains 43 classes with 39209 images for training, and 12630 images for testing. We resize GTSRB into 32*32*3 in our experiments.

Base Model Details: We choose several common architectures for our base models, such as VGG11, VGG16, VGG19 [18], ResNet18, and ResNet50 [19]. In order to satisfy the settings for deploying the models on chips, all of our models are trained by using quantization-aware training on weights. Therefore, we did not adopt the pre-trained models in our experiments.

Details for Generator: The architectures of the generators for NeuralFuse are based on the Encoder-Decoder structure. We design and compare three types of generators, namely Convolution-Based, Deconvolution-Based, and UNet-Based. For each type, we also consider large/small network sizes. Both Convolution-Based and Deconvolution-Based will follow similar architecture for ease of comparison. We train the generators based on EOPM under quantization-aware training. More details are given in Appendix D.

- **Convolution-Based**: We use Convolution with MaxPool layers to create the encoder, and Convolution with UpSample layers for the decoder. The architecture is inspired by [20].
- **Deconvolution-Based**: We use Convolution with MaxPool layers to create the encoder, and Deconvolution layers for the decoder.
- **UNet-Based**: UNet [21] has outperformed performance on several image segmentation tasks.

  We use Convolution with MaxPool layers to create the encoder, and Deconvolution layers for the decoder.

Energy Consumption Calculation: Reported energy in Figure 1 is calculated as the product of total number of SRAM memory accesses in a systolic-array-based CNN accelerator and the dynamic energy per reading access at a given voltage. Our work focuses on resilience to low-voltage bit errors in model weights, therefore, we report the reduction in overall weight memory energy. It needs to be noted that reporting the overall DNN accelerator energy, including computation for a specific hardware architecture, is not relevant to the scope of this work. The number of memory accesses is obtained from SCALE-SIM simulator [22], and our chosen configuration simulates an output-stationary dataflow and a 32x32 systolic array, with 256KB of weight memory. The dynamic energy per reading access of the SRAM at V_{min}, the bit error-free voltage, and V_{ber}, the voltage corresponding to 1% bit error rate ($0.83V_{min}$), are obtained at the same clock frequency from Cadence ADE Spectre simulations.

Relaxed and Restricted Access Settings: We consider two scenarios (relaxed/restricted access) in our experiments. For relaxed access, the information of the base model is not entirely transparent, but it allows obtaining gradients from the black-box model through backpropagation. Therefore, this setting allows direct training of NeuralFuse with the base model using EOPM. On the other hand, for restricted access, only the inference function is allowed for the base model. Therefore, we train NeuralFuse by using a white-box surrogate model and then transfer the generator to the access-restricted model.

Computing Resources and Training Process: For the base models, we take 100 epochs for training both CIFAR-10 and GTSRB and 200 epochs for training CIFAR-100. For the generators trained by EOPM, we take different epochs for each experiment until the generator converges.

For both base models and generators, we apply several data augmentation techniques to ensure the accuracy is close to state-of-the-art. For instance, we take random crops, random horizontal flip for all three datasets, and random affine only for GTSRB. All of the datasets would be normalized to $[-1, 1]^d$. Our experiments are conducted on Nvidia Tesla V100 GPUs with Pytorch for implementation.

D Implementation Details for NeuralFuse Generator

The architecture of the generators is based on the Encoder-Decoder structure. In this paper, we design three types of generators with large/small network sizes, namely Convolution-Based (ConvL, ConvS), Deconvolution-Based (DeConvL, DeConvS), and Unet-Based (UNetL, UNetS) to evaluate

the performance. ConvL is inspired by [20], and then we design three similar generators to ConvL, such as ConvS, DeConvL, and DeConvS. The model architectures are shown in Table 3. Besides, UNet-Based is different from other architectures; we follow the architecture in [21], then show the architecture in Table 4. 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, batch size=256, lambda=5, and Adam optimizer.

Table 3: Model architecture for both Convolution-Based and Deconvolution-Based models. Each ConvBlock consists of a Convolution (kernel=3x3, padding=1, stride=1), a Batch Normalization, and a ReLU layer. Each DeConvBlock consists a Deconvolution (kernel=4x4, padding=1, stride=2), a Batch Normalization, and a ReLU layer.

ConvL		ConvS		DeConvL		DeConvS	
Layers	#Channels	Layers	#Channels	Layers	#Channels	Layers	#Channels
(ConvBlock)x2, MaxPool	32	ConvBlock, Maxpool	32	(ConvBlock)x2, MaxPool	32	ConvBlock, Maxpool	32
(ConvBlock)x2, MaxPool	64	ConvBlock, Maxpool	64	(ConvBlock)x2, MaxPool	64	ConvBlock, Maxpool	64
(ConvBlock)x2, MaxPool	128	ConvBlock, Maxpool	64	(ConvBlock)x2, 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		

Table 4: Model architecture for UNet-Based models. Each ConvBlock consists of a Convolution (kernel=3x3, padding=1, stride=1), a Batch Normalization, and a ReLU layer. Other layer like deconvolution layer (kernel=2x2, padding=1, stride=2) is used in UNet-Based model. For the final Convolution layer, the kernel size is set to 1.

UNetL		UNetS	
Layers	#Channels	Layers	#Channels
L1: (ConvBlock)x2	16	L1: (ConvBlock)x2	8
L2: Maxpool, (ConvBlock)x2	32	L2: Maxpool, (ConvBlock)x2	16
L3: Maxpool, (ConvBlock)x2	64	L3: Maxpool, (ConvBlock)x2	32
L4: Maxpool, (ConvBlock)x2	128	L4: Maxpool, (ConvBlock)x2	64
L5: DeConv	64	L5: DeConv	32
L6: Concat[L3, L5]	128	L6: Concat[L3, L5]	64
L7: (ConvBlock)x2	64	L7: (ConvBlock)x2	32
L8: DeConv	32	L8: DeConv	16
L9: Concat[L2, L8]	64	L9: Concat[L2, L8]	32
L10: (ConvBlock)x2	32	L10: (ConvBlock)x2	16
L11: DeConv	16	L11: DeConv	8
L12: Concat[L1, L11]	32	L12: Concat[L1, L11]	16
L13: (ConvBlock)x2	16	L13: (ConvBlock)x2	8
L14: Conv	3	L14: Conv	3

E Visualization

To better understand how our proposed NeuralFuse works, we visualize the output distribution from the final linear layer of the based models and project the results onto the 2D space using TSNE [23]. Figure 3 shows the output distribution from ResNet18 trained by CIFAR-10 under 1% of bit error rate. We choose 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 1% of 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. (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.

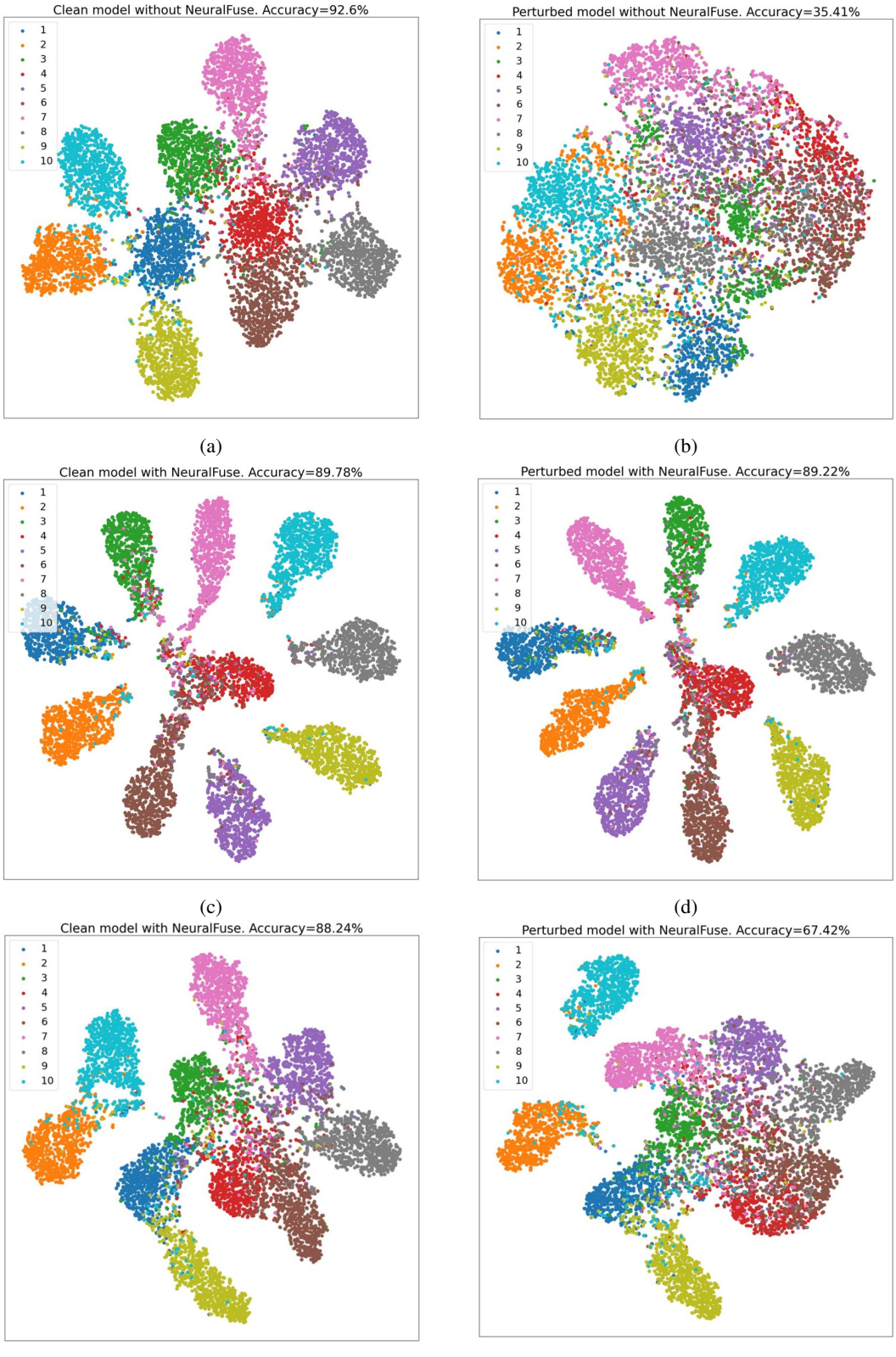


Figure 3: TSNE results for ResNet18 trained by CIFAR-10 under 1% of bit error rate. (a) Clean model. (b) Perturbed model. (c) Clean model with ConvL. (d) Perturbed model with ConvL. (e) Clean model with ConvS. (f) Perturbed model with ConvS

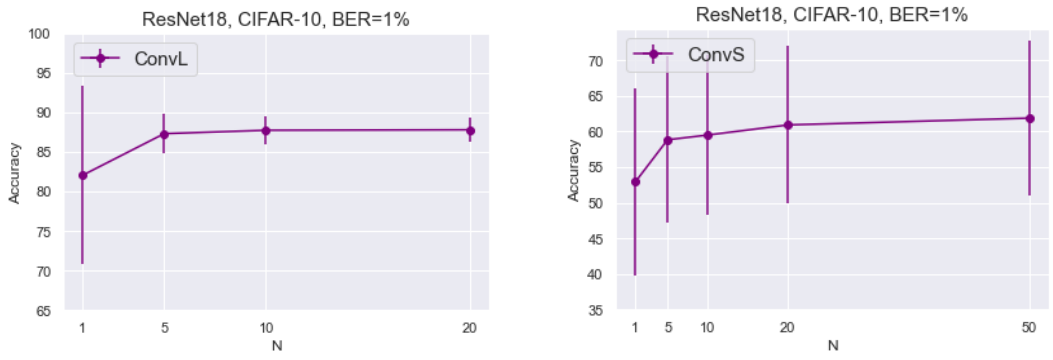
F Additional Analysis

Efficiency Ratio: To provide a full performance characterization of NeuralFuse, we analyze the relationship between the final improvement of each based model and generators of varying parameter counts or MACs (Multiply–Accumulate Operations). The efficiency ratio is defined as the improvement in perturbed accuracy divided by parameter counts or MACs.For fairness, we should only compare the efficiency ratio between two similar sizes of generators. We provide the efficiency results in Table 5 for ConvL and UNetL. We observe the UNet-Based generators have better efficiency than Convolution-Based for both improvements per Million-Parameters (M.Param) and improvements per 100-Million-MACs(100-M.MACs).

Table 5: The efficiency ratio for CIFAR-10 with ConvL and UNetL. Notations: B.E.R. (bit error rate).

Model	B.E.R.	ConvL		UNetL	
		Improve. per M.Param.	Improve. per 100-M.MACs	Improve. per M.Param.	Improve. per 100-M.MACs
ResNet18	1%	67.481%	60.63%	94.478%	110.145%
	0.5%	24.660%	22.157%	33.633%	39.21%
ResNet50	1%	37.448%	33.646%	102.297%	119.261%
	0.5%	35.187%	31.615%	47.447%	55.315%
VGG11	1%	62.313%	55.986%	92.019%	107.278%
	0.5%	32.277%	29%	39.94%	45.397%
VGG16	1%	69.556%	62.494%	98.8%	115.185%
	0.5%	30.291%	27.216%	40.584%	47.315%
VGG19	1%	57.646%	51.794%	95.362%	111.176%
	0.5%	33.026%	29.673%	43.054%	50.193%

Study for N in EOPM: Here, we study the effect of N used in EOPM. The results are shown in Figure 4. We report the results for ConvL and ConvS on ResNet18 trained by CIFAR-10 under 1% bit error rate. We can find that if we set larger N, the performance increases until saturation. On the other hand, in the experimental results for ConvL, larger N empirically has a lower standard deviation. This means larger N gives better stability but at the cost of increased training time. In contrast, for the small generator ConvS, we can find that the standard deviation is still large even trained by larger N. The reason can be that the ability of the representation learning of the small generator is weaker than the large generator. Therefore, there exists a trade-off between the stability of the performance of the generators and the total training time. In our implementation, choosing $N = 5$ or 10 is a good balance.



(a) (b)

Figure 4: The experimental results on different sizes of N. (a) Using ConvL. (b) Using ConvS.

G Implementation Details for SCALE-SIM

SCALE-SIM [22] 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 output and input of 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 for further energy calculation.

H Energy-Accuracy Tradeoff with 1% Random Bit Error Rate

As mentioned in [3], the total dynamic energy consumption can be calculated by using the total number of SRAM accesses times the dynamic energy of single SRAM access. Therefore, in Table 6, we first report the total weight memory access calculated by SCALE-SIM. We note the total weights memory access as T.W.M.A. In Table 7, we report the energy saving percentage of each combination of the based models and generators at a voltage corresponding to 1% of the random bit error rate. The formula of the energy saving percentage can be defined as:

$$\text{Energy Saving } (\%) = \frac{\text{Original Energy} - \left(\text{Energy}_{\text{low-voltage-regime}} + \text{Energy}_{\text{NeuralFuse}}\right)}{\text{Original Energy}} \times 100\% \quad (2)$$

Table 6: The total weights memory access calculated by SCALE-SIM. Notations: T.W.M.A. (total weight memory access)

Model	T.W.M.A.
ResNet18	2755968
ResNet50	6182144
VGG11	1334656
VGG16	2366848
VGG19	3104128
ConvL	320256
ConvS	41508
DeConvL	259264
DeConvS	86208
UNetL	180894
UNetS	45711

Table 7: The energy saving percentage for different combination of base models and NeuralFuse.

Model	Energy Saving Percentage					
	ConvL	ConvS	DeConvL	DeConvS	UNetL	UNetS
ResNet18	18.98%	29.09%	21.19%	27.47%	24.04%	28.94%
ResNet50	25.42%	29.93%	26.41%	29.21%	27.67%	29.86%
VGG11	6.6%	27.49%	11.17%	24.14%	17.05%	27.18%
VGG16	17.07%	28.85%	19.65%	26.96%	22.96%	28.67%
VGG19	20.28%	29.26%	22.25%	27.82%	24.77%	29.13%

We especially visualize the energy-accuracy tradeoff with 1% bit error rate for ResNet-18 (CIFAR-10) in Figure 5. The full result is shown in Figure 6.

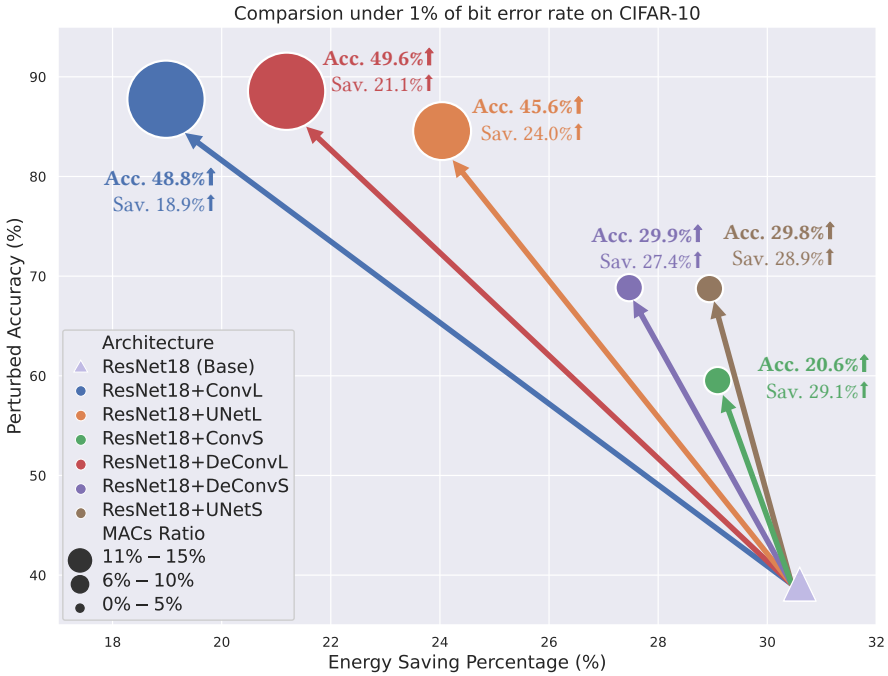


Figure 5: The energy-accuracy tradeoff with 1% bit error rate for ResNet-18 (CIFAR-10) and different NeuralFuse implementations. Bit errors are injected into model weights. The X-axis shows the percentage reduction in dynamic memory access energy for NeuralFuse implementations in a low voltage setting with 1% bit error rate, compared to the lowest bit error-free voltage. The Y-axis shows the perturbed model accuracy with 1% bit error rate. The circle marker area reflects the relative size of NeuralFuse to the base model.

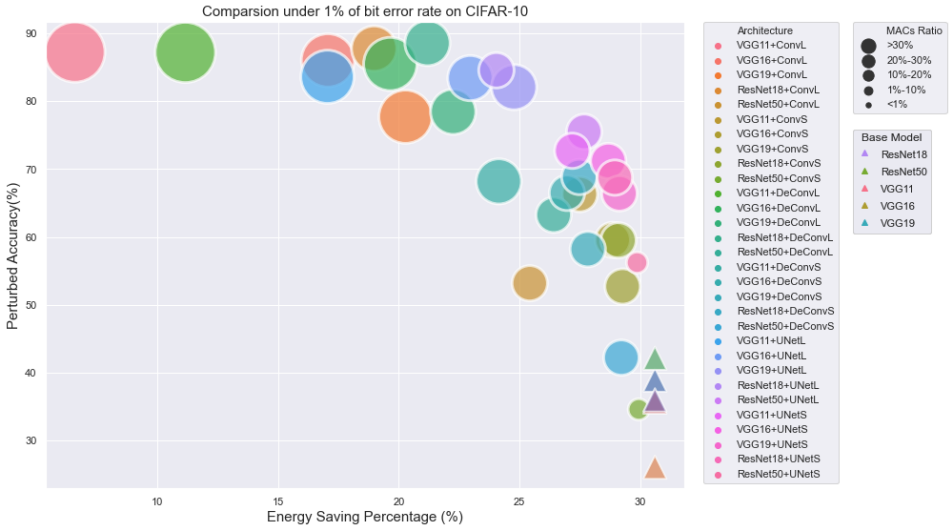


Figure 6: The trend between MACs rate and transformed accuracy for all of the based models with NeuralFuse on CIFAR-10 under 1% random bit error. When the MACs rate is large, it can achieve better accuracy. Therefore, there exists a tradeoff between the amount of energy saving and the perturbed model accuracy.

I Details for Model Parameters and MACs Value

We show the total numbers of parameters and MACs value in Table 8. We can find that all of our generators are smaller than based models either on total model parameters or MACs values.

Table 8: The parameter counts and MACs for all based models and all generators used in this paper.

Model	Parameter	MACs
ResNet18	$11,173,962$	557.14M
ResNet50	$23,520,842$	1.31G
VGG11	$9,231,114$	153.5M
VGG16	$14,728,266$	314.43M
VGG19	$20,040,522$	399.47M
ConvL	$723,273$	80.5M
ConvS	$113,187$	10.34M
DeConvL	$647,785$	64.69M
DeConvS	$156,777$	22.44M
UNetL	$482,771$	41.41M
UNetS	$121,195$	10.58M

J Additional Experimental Results on Relaxed Access

We conducted more experiments on Relaxed Access to show that our NeuralFuse can protect the models under different bit error rates. Here we show the results under 0.5% of bit error rate for all three datasets in Table 9, and 0.35% for CIFAR-100 in Table 10. We can find that for both CIFAR-10 and GTSRB, larger generators like ConvL, DeConvL, and UNetL have better performance than small generators, as we have seen in the previous section. In addition, although the improvements are less obvious on CIFAR-100 (the more difficult dataset), we can still conclude that our NeuralFuse is applicable to different datasets.

Table 9: Testing accuracy under 0.5% of random bit error rate. Notations: B.M. (based model), C.A. (clean accuracy), P.A. (perturbed accuracy), N.F. (NeuralFuse), and IP (total improvement of P.A.+N.F. *v.s.* P.A.).

B.M.	N.F.	CIFAR-10					GTSRB					CIFAR-100				
		C.A.	P.A.	C.A.+N.F.	P.A.+N.F.	IP	C.A.	P.A.	C.A.+N.F.	P.A.+N.F.	IP	C.A.	P.A.	C.A.+N.F.	P.A.+N.F.	IP
ResNet18	ConvL	92.6	70.079 ±11.565	90.43	87.915 ± 2.16	17.836	95.51	75.236 ±12.665	93.38	89.549 ± 1.89	14.313	73.69	20.898 ±7.406	65.18	38.951 ± 7.137	18.053
	ConvS			91.72	78.414 ± 8.292	8.335			94.79	87.676 ± 4.247	12.44			69.97	24.48 ± 7.587	3.582
	DeConvL			90.15	90.018 ± 0.157	19.939			95.392	93.379 ± 1.125	18.143			66.26	38.247 ± 6.876	17.349
	DeConvS			84.14	79.881 ± 3.56	9.802			95.76	90.086 ± 3.28	14.85			68.16	25.863 ± 6.757	4.965
	UNetL			89.67	86.316 ± 2.401	16.237			96.231	93.515 ± 1.563	18.279			66.23	40.129 ± 6.447	19.231
	UNetS			90.93	80.743 ± 5.767	10.664			95.519	91.434 ± 2.753	16.198			67.07	28.826 ± 6.82	7.928
ResNet50	ConvL	92.55	61 ±10.286	90.26	86.45 ± 3.204	25.45	94.96	73.982 ±13.027	94.58	90.55 ± 3.74	16.568	73.45	21.25 ±6.961	68.39	28.845 ± 6.675	7.595
	ConvS			90.82	73.263 ± 8.69	12.263			95.424	84.509 ± 8.532	10.527			71.86	23.172 ± 6.876	1.922
	DeConvL			89.54	87.212 ± 2.514	26.212			94.743	91.601 ± 2.874	17.619			68.1	28.628 ± 6.983	7.378
	DeConvS			90.3	75.459 ± 8.061	14.459			94.576	87.441 ± 5.88	13.459			70.295	24.953 ± 6.743	3.703
	UNetL			89.88	83.906 ± 3.632	22.906			96.548	93.718 ± 2.321	19.736			66.56	36.507 ± 6.217	15.257
	UNetS			89.68	76.084 ± 7.151	15.084			95.899	90.642 ± 4.793	16.66			69.09	26.083 ± 6.627	4.833
VGG11	ConvL	88.35	63.617 ±9.259	89.84	86.962 ± 1.344	23.345	91.86	64.866 ±10.787	93.864	92.59 ± 0.71	27.724	64.82	23.868 ±9.364	63.1	38.821 ± 9.29	14.953
	ConvS			88.2	74.492 ± 5.693	10.875			90.934	80.533 ± 3.547	15.667			62.67	27.854 ± 10.2	3.986
	DeConvL			89.6	86.864 ± 1.094	23.247			93.595	91.926 ± 0.643	27.06			63.58	40.046 ± 9.024	16.178
	DeConvS			88.27	75.686 ± 4.624	12.069			92.272	83.081 ± 3.68	18.215			61.88	29.778 ± 9.9	5.91
	UNetL			88.03	82.416 ± 1.793	18.799			94.79	90.612 ± 1.739	25.746			61.83	37.771 ± 9.007	13.903
	UNetS			88.07	75.767 ± 4.323	12.15			94.64	88.938 ± 2.204	24.072			61.65	29.838 ± 9.653	5.97
VGG16	ConvL	90.31	66.56 ±8.051	90.21	88.469 ± 0.88	21.909	95.23	58.797 ±8.885	95.558	93.232 ± 1.84	34.435	67.77	22.373 ±7.029	61.83	41.058 ± 5.553	18.685
	ConvS			89.93	77.756 ± 4.848	11.196			94.307	82.158 ± 6.154	23.361			63.79	27.484 ± 6.834	5.111
	DeConvL			89.71	88.221 ± 0.957	21.661			95.59	93.107 ± 2.034	34.31			62.75	42.132 ± 5.511	19.759
	DeConvS			90.01	78.374 ± 4.732	11.814			95.139	84.038 ± 5.26	25.241			62.07	29.862 ± 6.671	7.489
	UNetL			88.96	86.153 ± 1.469	19.593			95.978	92.777 ± 1.964	33.98			61.74	41.26 ± 5.049	18.887
	UNetS			89.03	80.238 ± 3.526	13.678			95.384	87.768 ± 3.553	28.971			61.58	31.286 ± 6.26	8.913
VGG19	ConvL	90.48	64.201 ±12.418	90.36	88.088 ± 1.81	23.887	95.47	69.148 ±11.142	95.614	93.36 ± 2.13	24.212	67.84	34.023 ±9.551	65.61	46.485 ± 6.795	12.462
	ConvS			89.6	74.516 ± 8.963	10.315			94.901	86.979 ± 4.42	17.831			66.57	38.255 ± 6.795	4.232
	DeConvL			90.43	88.515 ± 1.384	24.314			95.455	92.445 ± 2.222	23.297			65.74	46.876 ± 7.059	12.853
	DeConvS			89.71	75.24 ± 8.61	11.039			95.511	88.826 ± 3.68	19.678			66.53	39.037 ± 3.654	5.014
	UNetL			89.14	84.986 ± 2.718	20.785			94.909	91.737 ± 2.507	22.589			65.46	46.944 ± 6.456	12.921
	UNetS			89.22	77.14 ± 7.256	12.939			96.453	90.766 ± 3.366	21.618			66.31	40.149 ± 7.962	6.126

Table 10: Testing accuracy under 0.35% of random bit error rate on CIFAR-100 only.

B.M.	N.F.	CIFAR-100				
		C.A.	P.A.	C.A.+N.F.	P.A.+N.F.	IP
ResNet18	ConvL	73.69	31.441 ± 7.602	69.36	42.888 ± 6.226	11.447
	ConvS			72.14	35.117 ± 7.268	3.676
	DeConvL			69.2	42.876 ± 5.498	11.435
	DeConvS			71.64	35.829 ± 5.498	4.388
	UNetL			70.31	46.311 ± 5.541	14.87
	UNetS			70.94	38.309 ± 6.434	6.868
ResNet50	ConvL	73.45	35.674 ± 8.635	71.98	40.819 ± 7.525	5.145
	ConvS			72.95	37.389 ± 7.962	1.715
	DeConvL			71.65	41.748 ± 7.672	6.074
	DeConvS			72.81	38.938 ± 7.927	3.264
	UNetL			70.81	45.281 ± 6.691	9.607
	UNetS			72.59	39.555 ± 7.806	3.881
VGG11	ConvL	64.82	31.278 ± 10.021	63.93	42.355 ± 8.988	11.077
	ConvS			63.94	41.767 ± 8.296	10.489
	DeConvL			64.03	42.818 ± 9.071	11.54
	DeConvS			63.53	36.056 ± 10.069	4.778
	UNetL			63.45	40.877 ± 9.306	9.599
	UNetS			63.8	35.746 ± 9.925	4.468
VGG16	ConvL	67.77	31.108 ± 7.15	64.93	44.925 ± 5.275	13.817
	ConvS			65.96	36.255 ± 6.067	5.147
	DeConvL			64.98	46.576 ± 5.175	15.468
	DeConvS			64.87	38.064 ± 6.333	6.956
	UNetL			64.79	46.766 ± 4.613	15.658
	UNetS			65.01	39.808 ± 5.920	8.7
VGG19	ConvL	67.84	42.171 ± 9.382	66.91	49.151 ± 7.376	6.98
	ConvS			67.72	45.337 ± 8.539	3.166
	DeConvL			67.32	49.811 ± 7.631	7.64
	DeConvS			67.74	45.736 ± 8.429	3.565
	UNetL			67.42	50.022 ± 7.478	7.851
	UNetS			67.53	46.639 ± 8.413	4.468

K Additional Experimental Results on Transferability

We conduct more experiments on Restricted Access to show that our NeuralFuse can transfer to protect different black-box models. In Table 11, we show the results on CIFAR-10 in which NeuralFuse is trained under 1% bit error rate. In Table 12, 13 we show the results on GTSRB in which NeuralFuse is trained under both 1.5% and 1% bit error rate separately. In Table 14, 15, we show the results on CIFAR-100 in which NeuralFuse is trained under 1%, and 0.5% bit error rate separately. The results show that VGG19 as the white-box surrogate model can have better transferability than ResNet18 for all datasets. On the other hand, in some cases, We can find that if we try to transfer the generators to the target models, which have similar architectures to the surrogate model under a smaller bit error rate than what is used in training, then the results after transferring can be better than the original results with NeuralFuse. For example. in Table 11, if we use VGG19 as a source-based model with ConvL as a generator, and then transfer to VGG19 with ConvL under 0.5% bit error rate, then the accuracy after transferring is 88.913%, which is slightly better than original result 88.469% on VGG16. Besides, VGG19 under 0.5% can attain similar results for both ConvL and UNetL. We conjecture that this is because the generators trained on a larger bit error rate can cover the error patterns of a smaller bit error rate, and thus they can have better generalizability under a smaller bit error rate.

Table 11: Transferred results trained by 1% of random bit error rate on CIFAR-10 dataset. Notations: S.M. (source model, used for training generators), T.M. (target model, used for testing generators), B.E.R (the bit error rate of the target model), C.A. (clean accuracy, %), P.A. (perturbed accuracy, %), N.F. (NeuralFuse), and IP (total improvement of P.A.+N.F. *v.s.* P.A.).

S.M.	T.M.	B.E.R.	ConvL			UNetL		
			C.A.+N.F.	P.A.+N.F.	IP	C.A.+N.F.	P.A.+N.F.	IP
ResNet18	ResNet18	0.5%	89.78	89.545 ± 0.205	19.466	86.55	86.165 ± 0.251	16.086
	ResNet50	1%	89.51	36.041 ± 19.244	9.93	85.18	38.825 ± 19.254	12.714
		0.5%		75.11 ± 10.737	14.11		77.097 ± 5.047	16.097
	VGG11	1%	88.37	62.516 ± 8.417	20.349	76.75	61.057 ± 8.492	18.89
		0.5%		80.957 ± 4.558	17.34		73.69 ± 2.982	10.073
	VGG16	1%	89.55	63.343 ± 18.49	27.639	85.17	59.935 ± 16.423	24.231
		0.5%		84.956 ± 3.386	18.396		80.226 ± 4.49	13.666
	VGG19	1%	89.57	50.675 ± 22.355	14.654	85.34	51.088 ± 15.969	15.067
		0.5%		80.203 ± 8.67	16.002		76.526 ± 7.81	12.325
VGG19	ResNet18	1%	89.83	61.042 ± 17.158	22.099	87.03	69.698 ± 11.199	30.755
		0.5%		86.107 ± 6.943	16.028		84.207 ± 2.985	14.128
	ResNet50	1%	89.89	33.986 ± 19.04	7.875	87.02	44.204 ± 17.216	18.093
		0.5%		76.526 ± 10.411	15.526		80.649 ± 4.194	19.649
	VGG11	1%	89.73	76.531 ± 7.02	34.364	87.05	79.942 ± 5.623	37.775
		0.5%		88.024 ± 2.12	24.407		85.448 ± 0.777	21.831
	VGG16	1%	89.6	75.522 ± 12.179	39.818	87.18	78.857 ± 7.819	43.153
		0.5%		88.913 ± 0.616	22.353		86.212 ± 0.347	19.652
	VGG19	0.5%	89.84	89.55 ± 8.67	25.349	87.35	86.756 ± 0.375	22.555

Table 12: Transferred results trained by 1.5% of random bit error rate on GTSRB dataset.

S.M.	T.M.	B.E.R.	ConvL			UNetL		
			C.A.+N.F.	P.A.+N.F.	IP	C.A.+N.F.	P.A.+N.F.	IP
ResNet18	ResNet18	1%	95.693	93.927 ± 1.886	57.006	94.861	94.397 ± 0.432	57.476
		0.5%		95.709 ± 0.184	20.473		94.846 ± 0.246	19.61
	ResNet50	1%	94.434	37.024 ± 22.471	7.49	94.41	47.072 ± 23.398	17.538
		0.5%		77.463 ± 13.311	3.481		84.809 ± 9.482	10.827
	VGG11	1%	92.835	45.155 ± 10.303	10.291	91.409	50.527 ± 13.309	15.663
		0.5%		79.39 ± 5.75	14.524		83.878 ± 4.23	19.012
	VGG16	1%	95.439	31.057 ± 12.555	15.941	94.64	36.77 ± 11.638	21.654
		0.5%		84.548 ± 8.322	25.751		86.025 ± 8.607	27.228
	VGG19	1%	94.988	56.366 ± 14.591	19.794	94.315	60.777 ± 15.476	24.205
		0.5%		86.915 ± 3.385	17.767		87.745 ± 3.766	18.597
VGG19	ResNet18	1%	88.401	50.328 ± 12.454	13.407	92.787	63.717 ± 15.521	26.796
		0.5%		77.88 ± 7.388	2.644		87.482 ± 3.915	12.246
	ResNet50	1%	87.546	29.707 ± 17.205	0.173	92.455	40.408 ± 20.5	10.874
		0.5%		67.939 ± 16.567	-6.043		77.51 ± 14.717	3.528
	VGG11	1%	89.66	47.053 ± 10.687	12.189	93.5	60.002 ± 12.248	25.138
		0.5%		76.314 ± 5.064	11.448		85.955 ± 3.81	21.089
	VGG16	1%	92.985	29.22 ± 14.5	14.104	93.048	38.398 ± 15.606	23.282
		0.5%		75.676 ± 12.417	16.879		79.903 ± 8.327	21.106
	VGG19	1%	95.115	87.409 ± 5.969	50.837	94.608	88.65 ± 5.003	52.078
		0.5%		92.448 ± 2.426	23.3		92.376 ± 2.193	23.228

Table 13: Transferred results trained by 1% of random bit error rate on GTSRB dataset.

S.M.	T.M.	B.E.R.	ConvL			UNetL		
			C.A.+N.F.	P.A.+N.F.	IP	C.A.+N.F.	P.A.+N.F.	IP
ResNet18	ResNet18	0.5%	95.717	95.267 ± 0.454	20.031	96.247	95.707 ± 0.303	20.471
	ResNet50	1%	94.497	35.566 ± 21.197	6.032	95.645	42.626 ± 23.398	13.092
		0.5%		78.805 ± 13.447	4.823		87.305 ± 8.971	13.323
	VGG11	1%	93.143	45.792 ± 11.276	10.928	93.991	47.109 ± 14.781	12.245
		0.5%		81.84 ± 5.027	16.974		84.162 ± 4.768	19.296
	VGG16	1%	95.455	26.512 ± 11.962	11.396	95.527	32.382 ± 11.286	17.266
		0.5%		82.215 ± 9.026	23.418		85.394 ± 6.749	26.597
	VGG19	1%	94.909	53.22 ± 14.544	16.648	95.614	60.905 ± 15.12	24.333
		0.5%		85.441 ± 4.549	16.293		87.492 ± 3.714	18.344
VGG19	ResNet18	1%	93.69	53.1 ± 15.58	16.179	95.004	63.439 ± 18	26.518
		0.5%		83.854 ± 7.631	8.618		89.656 ± 4.842	14.42
	ResNet50	1%	92.827	30.591 ± 18.33	1.057	95.432	38.904 ± 21.458	9.37
		0.5%		74.69 ± 17.721	0.708		81.546 ± 15.819	7.564
	VGG11	1%	93.729	50.581 ± 11.212	15.717	95.052	58.884 ± 14.84	24.02
		0.5%		82.341 ± 5.134	17.475		87.466 ± 3.747	22.6
	VGG16	1%	95.218	27.794 ± 14.628	12.678	95.218	33.474 ± 14.384	18.358
		0.5%		78.995 ± 11.512	20.198		81.813 ± 7.79	23.016
	VGG19	0.5%	96.033	93.998 ± 2.197	24.85	95.368	93.881 ± 2.115	24.733

Table 14: Transferred results trained by 1% of random bit error rate on CIFAR-100 dataset.

S.M.	T.M.	B.E.R.	ConvL			UNetL		
			C.A.+N.F.	P.A.+N.F.	IP	C.A.+N.F.	P.A.+N.F.	IP
ResNet18	ResNet18	0.5%	54.77	35.81 ± 5.159	14.912	50.56	39.287 ± 2.797	18.389
		0.35%		41.706 ± 3.711	10.265		43.279 ± 1.407	11.838
	ResNet50	1%	44.94	2.243 ± 2.048	-0.801	41.53	2.362 ± 1.912	-0.682
		0.5%		15.918 ± 8.249	-5.332		17.106 ± 7.139	-4.144
		0.35%		23.663 ± 7.122	-12.011		26.15 ± 5.582	-9.524
	VGG11	1%	41.18	9.828 ± 5.579	1.596	37.54	10.159 ± 5.127	1.927
		0.5%		24.172 ± 5.891	0.304		24.504 ± 4.675	0.636
		0.35%		29.034 ± 5.375	-2.244		28.229 ± 4.514	-3.049
	VGG16	1%	44.03	7.94 ± 3.71	1.341	39.48	10.131 ± 4.47	3.532
		0.5%		22.416 ± 7.588	0.043		26.33 ± 5.317	3.957
		0.35%		28.083 ± 5.906	-3.025		30.64 ± 3.615	-0.468
	VGG19	1%	44.16	13.519 ± 6.058	2.949	40.74	15.61 ± 6.209	5.04
		0.5%		27.85 ± 4.827	-6.173		29.181 ± 4.622	-4.842
		0.35%		33.24 ± 48.19	-8.931		32.816 ± 3.894	-9.355
VGG19	ResNet18	1%	55.45	5.761 ± 3.686	1.173	57.29	6.757 ± 4.405	2.169
		0.5%		24.578 ± 6.331	3.68		28.143 ± 5.948	7.245
		0.35%		31.124 ± 5.011	-0.317		36.354 ± 4.502	4.913
	ResNet50	1%	56.1	2.765 ± 2.09	-0.279	56.07	3.721 ± 2.381	0.677
		0.5%		18.89 ± 8.591	-2.36		22.818 ± 8.468	1.568
		0.35%		28.732 ± 8.195	-6.942		33.65 ± 7.026	-2.024
	VGG11	1%	52.84	12.312 ± 8.371	4.08	53.86	15.35 ± 9.395	7.118
		0.5%		30.02 ± 9.282	6.152		33.28 ± 7.222	9.412
		0.35%		36.517 ± 7.723	5.239		38.769 ± 6.536	7.491
	VGG16	1%	53.63	11.2 ± 4.364	4.601	55.24	13.563 ± 5.241	6.964
		0.5%		32.43 ± 7.329	10.057		35.857 ± 6.175	13.484
		0.35%		39.433 ± 6.263	8.325		42.389 ± 4.877	11.281
	VGG19	0.5%	59.4	50.211 ± 3.057	16.188	58.73	49.066 ± 3.511	15.043
		0.35%		53.074 ± 2.828	10.903		51.989 ± 3.145	9.818

Table 15: Transferred results trained by 0.5% of random bit error rate on CIFAR-100 dataset.

S.M.	T.M.	B.E.R.	ConvL			UNetL		
			C.A.+N.F.	P.A.+N.F.	IP	C.A.+N.F.	P.A.+N.F.	IP
ResNet18	ResNet18	0.35%	65.18	47.706 ± 4.874	16.265	66.23	49.249 ± 4.139	17.808
	ResNet50	0.5%	62.51	24.005 ± 9.896	2.755	63.53	26.397 ± 9.101	5.147
		0.35%		36.271 ± 8.942	0.597		39.384 ± 8.05	3.71
	VGG11	0.5%	59.21	33.042 ± 9.848	9.174	61.08	34.151 ± 9.78	10.283
		0.35%		40.382 ± 8.666	9.104		41.384 ± 9.047	10.106
	VGG16	0.5%	59.49	34.704 ± 8.015	12.331	61.4	37.54 ± 6.763	15.167
		0.35%		42.911 ± 5.955	11.803		45.261 ± 4.948	14.153
	VGG19	0.5%	61.56	43.672 ± 6.202	9.649	62.01	45.036 ± 6.313	11.013
		0.35%		49.014 ± 5.452	6.843		50.472 ± 5.325	8.301
VGG19	ResNet18	0.5%	66.07	24.916 ± 6.659	4.018	67.77	27.687 ± 6.827	6.789
		0.5%		34.4 ± 5.434	2.959		38.103 ± 5.619	6.662
	ResNet50	0.5%	66.16	22.657 ± 7.751	1.407	66.7	25.432 ± 8.019	4.182
		0.35%		35.513 ± 7.678	-0.161		38.824 ± 7.5	3.15
	VGG11	0.5%	59.89	29.262 ± 10.064	5.394	61	31.219 ± 9.831	7.351
		0.35%		36.574 ± 9.533	5.296		38.101 ± 9.014	6.823
	VGG16	0.5%	62.49	30.775 ± 7.311	8.402	62.62	33.03 ± 7.285	10.657
		0.35%		39.988 ± 6.454	8.88		42.455 ± 5.924	11.347
	VGG19	0.35%	65.61	51.979 ± 6.198	9.808	65.46	52.573 ± 6.06	10.402

L 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 [3], 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 16. After careful tuning of hyperparameters, we find that we are not able to obtain satisfactory improvements when adopting adversarial training. Empirically, we argue that adversarial training may not be suitable for training generator-based methods.

Table 16: 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 1% bit error rate. Notation: C.A. (clean accuracy, %), P.A. (perturbed accuracy, %), N.F. (NeuralFuse), and IP (total improvement of P.A.+N.F. *v.s.* P.A.).

K-bits	C.A.+N.F.	P.A.+N.F.	IP
100	92.37	38.307 ± 12.145	-0.636
500	92.14	38.653 ± 12.458	-0.29
5000	92.57	38.851 ± 12.513	-0.092
20000	60.11	22.967 ± 8.119	-15.976
100000	71.08	23.571 ± 6.587	-15.372