NNTesting: Neural Network Fault Attacks Detection Using Gradient-Based Test Vector Generation

Antian Wang
Department of Electrical and Computer Engineering
Clemson University
Clemson, SC, USA
antianw@clemson.edu

Bingyin Zhao
Department of Electrical and Computer Engineering
Clemson University
Clemson, SC, USA
bingyiz@clemson.edu

Weihang Tan
Department of Electrical and Computer Engineering
Clemson University
Clemson, SC, USA
wtan@clemson.edu

Yingjie Lao
Department of Electrical and Computer Engineering
Clemson University
Clemson, SC, USA
ylao@clemson.edu

2023 60th ACM/IEEE Design Automation Conference (DAC) | 979-8-3503-2348-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/DAC56929.2023.10247885

***Abstract*—Recent studies have shown Neural Networks (NNs) are highly vulnerable to fault attacks. This work proposes a novel defensive framework, *NNTesting*, for detecting the fault attack and recovering the model. We first leverage gradient-based optimization to generate a set of high-quality Test Vectors (TVs) that effectively differentiate faulty profile models and further optimize the TV set by reducing the TVs through compression. The selected final TV set is then used to recover the model. The effectiveness of the proposed method is comprehensively evaluated on a wide range of models across various benchmark datasets. For instance, we successfully generate more than thousands of TV candidates using a gradient-based generation method. After compression, we achieve up to 94.76% detection success rate with only 140 TVs on the CIFAR-10 dataset.**

***Index Terms*—Neural Network, Fault Attack, Defense, Test Vector Generation, Model Repairing**

I. Introduction

Neural Networks (NNs) have achieved remarkable success in various fields, including image classification [1], natural language processing [2], and self-driving cars [3]. However, NNs are vulnerable to various attacks. Prior works mainly focus on data-oriented attacks (e.g., adversarial examples and data poisoning) [4, 5]. Orthogonal to these, fault injection attacks [5–7] are model-oriented. Such attacks at the system level compromise the overall accuracy [5–8] or inject a backdoor into victim NNs [9] by only modifying a tiny fraction of model parameters via techniques such as adversarial laser beam [8] and rowhammer [10].

One of the most prevalent fault injection attacks is the Bit-Flip Attack (BFA), where the adversary leverages the DRAM vulnerability and tampers a few bits of the most critical parameters stored in the memory. A recent work [11] demonstrates that the top-1 accuracy of a ResNet-18 model degrades from 69.8% to 0.1% by only flipping 13 out of 93 million bits using the bit progressive search-based BFA, posing a severe threat to safety-critical applications. Existing defenses employ patching-based [12, 13] or detection-based [14, 15] methods as countermeasures. Unfortunately, the robustness improvement of these approaches is limited (e.g., the poor generalizability to adaptive attacks and low detection rate due to false positives, etc.). Moreover, models protected by such methods may sacrifice accuracy or incur significant overhead in time and algorithm complexity. A new honeypot-based approach improves the detection rate and reduces the overhead, where trapdoors are injected into the model to induce BFA to concentrate on particular weights [16]. However, this method still incurs overhead to alter the model, impacting the accuracy.

In this work, we propose *NNTesting*, a novel testing framework that detects and fixes the faults from BFA, attempting to achieve a high detection rate at a low cost without sacrificing model accuracy. Unlike prior works, our approach does not modify any pre-trained parameters nor inspect every single weight in the model indiscriminately. Our main idea is motivated by the conventional digital hardware testing paradigm, where high-quality test patterns are used to detect malfunctions [17, 18]. We likewise utilize carefully selected Test Vectors (TVs) for BFA detection and fixing. NNTesting consists of three phases: i) profile faulty model generation; ii) TV generation; and iii) TV compression. We first perform the state-of-the-art BFA [11] on the benign model to create a certain number of profile faulty models with flipped weights that significantly undermine the model accuracy. We then generate a TV candidate set that distinguishes the maximum number of profile faulty models (i.e., covers as many faults as possible) using a gradient-based algorithm. We exploit the benign and profile faulty models to optimize TV generation concurrently. Finally, we compress all TV candidates to find an optimal set that detects the most faults with the fewest TVs. This compression step is essential to improving testing efficiency. The contributions of this paper are summarized as follows:

- We propose a novel model agnostic hardware testing framework NNTesting against BFA, which detects and fixes flipped bits with a high detection rate at a low cost while preserving the model accuracy.
- We propose a novel gradient-based method to generate high-quality TVs that effectively distinguish fault models from benign ones.
- We develop an efficient compression algorithm to reduce the TV set, which significantly reduces the overhead and complexity of testing.
- We empirically validate the effectiveness of NNTesting in detecting and fixing faults on a wide range of models across various benchmark datasets.

The rest of the paper is organized as follows: In Section II, we introduce the related research to our work. We then define the threat model and elaborate on details of the proposed NNTesting framework in Section III and Section IV, respectively. In Section V, we demonstrate the experiment results and analysis. Finally, we conclude the paper in Section VI.

II. Related Works

A. Bit-Flip Attacks

Recent works developed progressive bit search and flipping to maximize the adversarial effect [11, 19]. The goal of BFA is to degrade the model's accuracy by performing a series of bit-flips. The gradient-based progressive bit search iteratively identifies the most vulnerable bits. It consists of intra-layer and cross-layer search, finding the bits to be flipped within the layer, selecting the most detrimental bit-flip across the layers, and flipping the bit which causes the most significant loss that individual bit-flip contributes to the overall model. Some recent works also improve BFA technique [20, 21].

B. Defenses against BFA

Existing countermeasures against BFA are broadly categorized as patching-based, detection-based, and honeypot-based defense. The patching-based approaches exploit binarization [13] and block reconstruction [12] to patch the vulnerable parts in NN models and improve the robustness against BFA. However, such defenses sacrifice model inference accuracy due to modifying the parameters in the patching process. The detection-based approaches aim to categorize, locate and repair the tampered bits by comprehensively inspecting model parameters. For example, the self-test framework proposed in [22] identifies bit-flip types, namely random, worst-case bit-flip on selected weights' Most Significant Bits (MSB), imprecise programming, and drifting faults. A primary drawback of detection-based approaches is the high overhead caused by exhaustive inspection of large models (e.g., modern NN models have billion to trillion-level parameters). Methods using hash functions to protect NN's critical weights have also been investigated [23]. The honeypot-based approach [16] embeds honey neurons into the model to lure the attacker's attention. It employs retraining-based and one-shot embedding to inject trapdoors into the model and induce the attacker to flip particular bits pre-defined by the defender. However, this method alters model parameters and leads to an accuracy drop.

C. Digital Hardware Testing

NNTesting is partially inspired by the prior works of digital hardware testing. In particular, digital hardware testing finds input sequences to discover the circuit incorrectness [24]. It exploits Test Patterns Generation (TPG) to generate qualified Test Patterns (TPs) that distinguish a good circuit model and its various faulty counterparts. In this work, we adopt the concept and analogous benign NNs as good circuit models, faulty NNs as the faulty circuit models, and TVs that differentiate benign and faulty models as TPs. We embraced the deterministic TPG approach (i.e., an optimized TV set) for our TV generation for detection effectiveness. We also incorporated the concept of TP compression, which is a part of TPG, to obtain efficient TPs, and design a highly efficient TV compression algorithm to enhance the performance of the approach further.

III. Threat Model

In this paper, we consider real-world scenarios where model owners provide a warranty to detect and fix failures on deployed models caused by fault attacks. We adopt a consistent threat model defined in prior works [10, 11] where the attacker aims to undermine a NN model by modifying a small number of model parameters. We assume a strong adversary to have the necessary capability (e.g., model knowledge and physical access) to perform the fault attack. The location and numbers of tampered bits are unknown to the defender (i.e., the model provider). However, the attacker is assumed to keep the attack as stealthy as possible by only changing a minimal amount of bits. After the attack, the defender tries to efficiently repair the model without retraining. Note that the attack detection is straightforward since the model accuracy is dropped significantly. The defender's goal is to locate and fix the flipped bits efficiently.

IV. NNTesting

The proposed NNTesting is designed to efficiently defend against BFA as shown in Fig. 1, demonstrating the high-level overview. A model provider is responsible for model training. In completion of training and before delivering the model to the user for deployments, the model provider derives a set of TV candidates based on profile faulty models and then compresses them to obtain an optimized TV set. The TV set locates faults and recovers the faulty model after deployment in case of BFA occurrence. By using the optimized TV set, the user promptly locates the flipped bits and fixes them rather than reloading all parameters or retraining the model. In this section, We present the detailed flow of NNTesting.

A. Profile Faulty Model Generation

The first step in NNTesting involves generating profile faulty models, which serve as surrogate models for actual victim models. These profile models cover the most vulnerable bits in a model, enabling the TV generation algorithm to find the most valuable fault patterns and generate high-quality TVs covering most faults caused by BFA. The primary theoretical basis of our approach is that different parameters have different vulnerabilities against fault attacks. Particular parameters are critical to the inference accuracy of the NN model, which are the targets to tamper. In contrast, other parameters are much less vital, which in fact is consistent with the findings in model pruning [25]. Thus, although the exact parameters being attacked are unpredictable, the attacker tends to select these more critical ones over others. To this end, we argue that different attack algorithms would search for a similar set of "important" and "vulnerable" parameters to attack, which is empirically verified by our experiments. We use one representative BFA algorithm, namely bit progressive

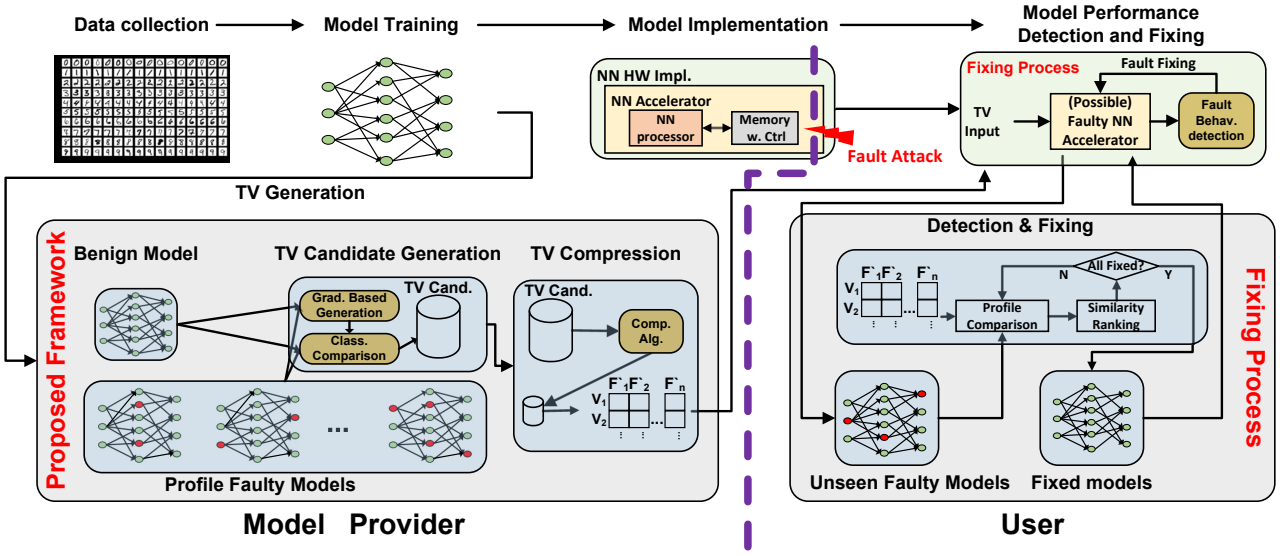

Fig. 1: Overview of NNTesting. The process is situated after the training of an NN. We first generate profile faulty models by existing fault attacks, which will then be used to derive TV candidates that will differentiate parameter changes in this model. The compressed TV sets and the corresponding classifications can be used to locate and repair the parameters, which are compromised by future fault attacks.

search [11], to generate profile faulty models and derive TVs upon these models, which can then be generalized to unseen faulty models attacked by different attack algorithms. Note that we want the profile models to encompass a wide range of vulnerable bits. Therefore, we utilize various input images to generate different profile models, which enrich the diversity of profile models and hence improve vulnerable bits coverage.

B. Test Vectors Generation

Next, we generate TV candidates based on these profile faulty models. We propose a novel gradient-based approach for TV generation. The idea is partially inspired by the prior work of adversarial examples [4], which aims to cause a classification discrepancy between benign and perturbed inputs. In contrast to adversarial examples, we seek to generate TVs that cause a classification discrepancy between benign and profile faulty models. These TVs distinguish faulty models from benign ones. We achieve this by solving the optimization objective defined as:

$$\begin{aligned}\mathcal{L}_{\text{all}} = &\alpha \cdot \mathcal{L}\Big(F, \mathbf{x} + g[\nabla_{\mathbf{x}}\mathcal{L}(F, \mathbf{x}, y_t)] \\ &+ (1-\alpha)\cdot \mathcal{L}(F', \mathbf{x} + g[(\nabla_{\mathbf{x}}\mathcal{L}(F', \mathbf{x}, y'_t))]\Big)\end{aligned} \tag{1}$$

where $\mathcal{L}$ is cross-entropy loss, F is the benign model, and F' as the profile faulty model. Our approach involves minimizing the loss of F and that of F' with respect to the TV $\mathbf{x}$ and target class label y_t, y'_t, so that the generated TV effectively differentiates these two models. α controls the mixing strengths between the two loss functions. We explore three different approaches to select y_t and y'_t, namely soft, medium, and hard selection. For the hard selection, we enumerate all possible different classes for y_t and y'_t. While for medium and soft selection, they are chosen by logit value ranking and current iterations' classification result of F', F, respectively. The motivation is to improve the efficiency of the TV generation algorithm as we empirically find that soft selection takes significantly less time, while only introducing trivial performance degradation. Consequently, we use the soft selection approach in our final implementation.

The function $g[\cdot]$ in our approach refers to the gradient-based generation method. The high-level concept is illustrated in Fig. 2. The TV generation process moves initial seed TVs sampled from the dataset to the decision boundary, following the gradient directions (i.e., dotted arrows) to achieve the TV generation objective expressed in Equation (1). The generated TVs that successfully cross the decision boundary are marked as stars in Fig. 2 and are considered TV candidates. To accomplish this, we have adapted techniques from FGSM [4] and PGD [26] in our approach.

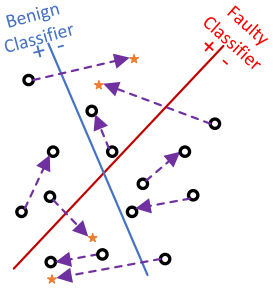

Fig. 2: Illustration of the TV generation process. Circles represent the seed TVs, dotted arrows represent the gradient-based search process, and stars indicate the generated TVs that can differentiate parameter changes in the model.

C. Test Vectors Compression

The TV generation described in Section IV-B generates massive TV candidates to cover as many faults as possible for each profile faulty model with redundancy. We then propose a

TV compression algorithm to find an optimal set that detects and locates maximum amount of faults with a minimum number of TVs. The compression reduces the size of the TV set by iteratively removing redundant candidates. We utilize collision to evaluate the performance after compression and guide the removal process. Our algorithm uses a coarse-to-fine-grained strategy, as summarized in Algorithm 1.

Algorithm 1 Probabilistic Test Vector Compression

Require: Initial TV mapping M_{org}, Compression method C, initial TV removal size S_{init}.
Ensure: Compressed TV mapping M_{comp}.
1: CNT_{succ}=0, CNT_{fail}=0, $S_{\text{comp}} = S_{\text{init}}$
2: OrgCNT=CollisionCNT(M_{org}), $M_{\text{comp}} = M_{\text{org}}$
3: **while** Stopping criteria do not satisfy **do**
4: Remove S_{comp} TVs using C, resulting M_{temp}.
5: **if** CollisionCNT(M_{temp})$\leq$ OrgCNT **then**
6: $M_{\text{comp}} = M_{\text{temp}}$, CNT_{succ}++
7: **else**
8: CNT_{fail}++
9: S_{comp}=Comp_Size_Mod($S_{\text{comp}}, CNT_{\text{succ}}, CNT_{\text{fail}}$)
10: **for** Remaining TVs going backward **do**
11: Iterative delete each TV resulting M_{temp}.
12: **if** CollisionCNT(M_{temp})$\leq$ OrgCNT **then**
13: $M_{\text{comp}} = M_{\text{temp}}$

The total collision of TV set $\mathbf{x}$ for a set of faulty NN $S_{F'}$ with individual model F'_j is defined as:

$$\text{collision} = \sum_{j_1}^{|S_{F'}|} \sum_{j_2=j_1+1}^{|S_{F'}|} \mathbb{1}_{\text{colsum}(j_1,j_2,,\mathbf{x})=|\mathbf{x}|} \quad (2)$$

$$\text{colsum}(j_1, j_2, \mathbf{x}) = \sum_i \mathbb{1}_{F'_{j_1}(\mathbf{x})=F'_{j_2}(\mathbf{x})}, \quad (3)$$

where colsum is the sum of indicator function for all available $\mathbf{x}$ in the set whether they have identical classification results for F'_{j_1} and F'_{j_2} as in Equation (2). The collision in Equation (3) checks whether all pairs of F'_{j_1}, F'_{j_2} have a total amount of collision TVs as the number of available $\mathbf{x}$ using indicator function $\mathbb{1}_{\text{colsum}(j_1,j_2,\mathbf{x})=|\mathbf{x}|}$. All available $\mathbf{x}$ cannot distinguish F' if colsum$(j_1, j_2, \mathbf{x}) = |\mathbf{x}|$, resulting a collision between faulty model F'_{j_1} and F'_{j_2} over $\mathbf{x}$.

The proposed compression algorithms in Algorithm 1 reduces the remaining TVs without an increase in collisions iteratively. The majority run-time of the algorithm is consumed by the collision computation (i.e., CollisionCNT). To this end, we use a universal hash function to accelerate the process by hashing all individual profile model's classification results with respect to the TVs within M, which is the TV classification results of profile faulty models. Then we enumerate all key-value pairs to accumulate the collisions within the pair to compute the indicator function Equation (3), significantly reducing the algorithm's time complexity.

For the compression process (Step 3 to Step 9 in Algorithm 1), we assign TV candidates with a probability of being removed from the current set of TV candidates based on a randomized strategy in Step 4, where all TVs have an equal probability of being removed in the experiment. Note that the traditional integer-programming-based TV compression method in [27] is time-consuming for such a large problem size. Using such a randomized strategy significantly improves the run-time. If the remaining TV keeps an identical collision property as defined in Equation (2), the compression is successful. We check whether collisions within the updated TV mapping deteriorate the collisions or not at Step 5.

At the end of randomized TV compression, we perform a final round of exhaustive TV reduction by enumerating the remaining TVs in Step 10 to Step 13. The result M_{comp} is the mapping between $\mathbf{x}$'s and profile faulty model's fault locations annotated as $F'_1, F'_2, \ldots F'_n$ in Fig. 1.

D. Detection and Repairing

We finally perform faulty model detection and repairing process using the optimal set of TVs obtained from Section IV-C, which only relies on the model inference without model weight inspection. The procedure is shown in the bottom right part of Fig. 1 to monitor the performance of NN hardware implementations by comparing the current TV's classification results with the benign model's ground truth classification results. We fed TVs to potential under-attack models (unseen profile models in the experiment). Then, we use the known TV classification results for the profile faulty models and profile faulty models' fault locations to fix errors of the unseen faulty models. The fault locations of the profile faulty models with the highest classification similarity with the current model are identified as the possible fault locations. Once the possible locations of faults are identified, they are iteratively refreshed back to the correct values. The success of repairing each bit is assessed by monitoring the model performance on a small set of held-out validation data. The process stops once the model achieves an acceptable level of TVs' classification results.

Overall, this approach allows for detecting and repairing faults without relying on model weight inspection, making it practical for real-world implementations.

V. Experiment and evaluation

A. Experimental Settings

The NN training and TV generation are implemented in Pytorch [28] based on datasets of MNIST [29], CIFAR10, and CIFAR100 [30]. One fully-connected layer (1FC) model and LeNet5 [29] are used for the MNIST dataset. VGG11 [1] and VGG19 with Batch Normalization (BN) as well as ResNet [31] are used for CIFAR10 and CIFAR100 datasets. All models are quantized in 8-bit. We use the bit-flip progressive search [11] for the $1,000$ distinct profile and $3,000$ distinct unseen faulty model generation. All faulty models are generated randomly within 50 iterations of the attack algorithm.

B. Experimental Results and Discussions

1) Profile model flip location distribution: We first examine the occurrence of bit-flip locations of generated 4,000 distinct

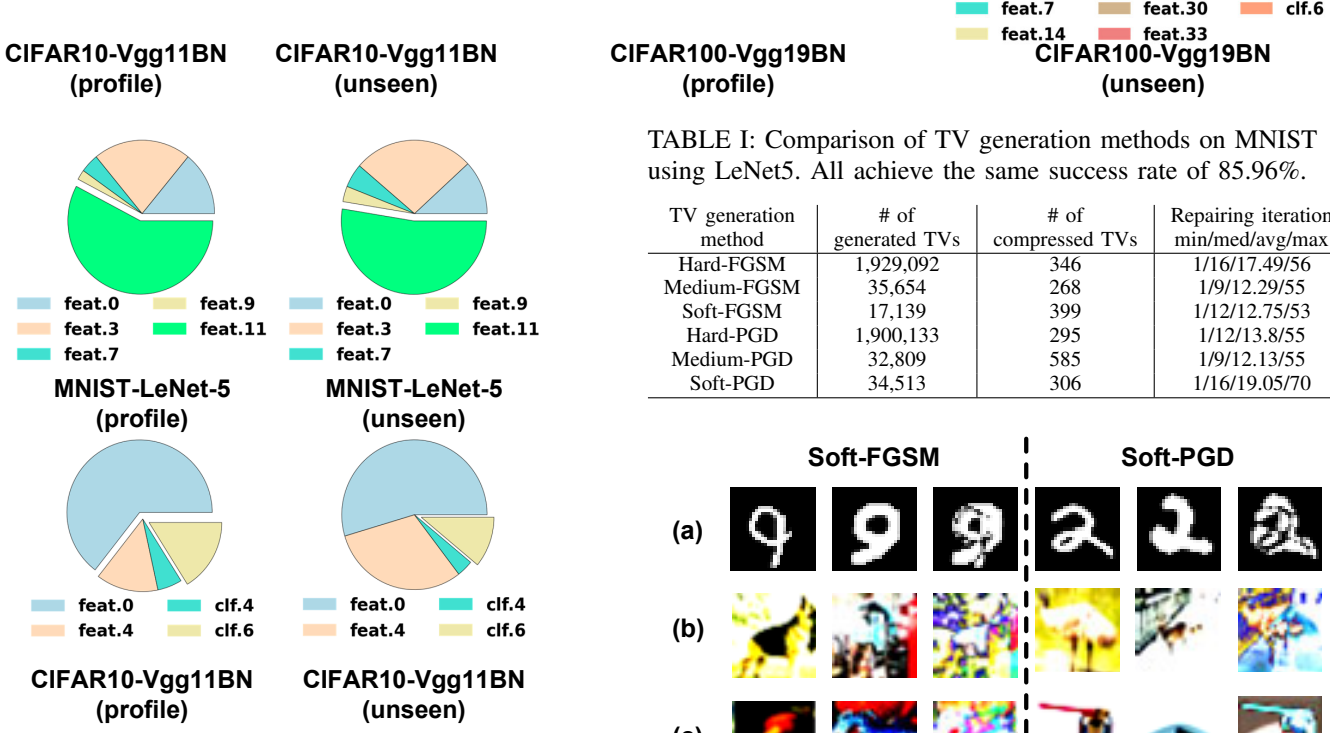

Fig. 3: Faulty model bit-flip layer statistics with more than 10 flips in faulty models (feat: feature; clf: classifier layer).

faulty models. From Fig. 3, a limited number of bit locations (all in MSBs), which mostly lie in a few layers of the entire NN, are flipped among the profile and unseen distinct faulty models. In the meantime, the profile and unseen distinct faulty models share similar bit-flip layer locations. Such an observation validates the idea of using the profile models to repair the unseen faulty models. However, we notice the difference in distribution between profile and unseen faulty models that unseen faulty models contain more diverse bit-flip locations given their larger sizes.

2) TV generation and compression performance: From the results in Table I, we observe that hard selection of classes during the TV generation using the gradient-based generation method generates a significantly larger amount of TV candidates with similar collision counts compared with the gradient-based generation method using soft or medium methods. In order to reduce the complexity of the method, especially for larger NNs, the soft method is the best option. Thus, we only present the results for soft methods in the following. In addition, we observe that the FGSM and PGD methods achieve similar performance.

Table I shows the effectiveness of the compression algorithm proposed in Section IV-C to discover the high-quality TVs among the TV candidates. The number of compressed TVs for the models used in the experiments has a manageable size for regular NN hardware implementation performance monitoring, testing, and fixing in practice. From the visualization of the generated TV in Fig. 4, the generated TVs are mostly visually recognizable.

3) Detection and repairing: The performances of the detection for different models are shown in Table II. We present the minimum, medium, maximum, and average iterations needed for repairing the faults. It can be observed that the proposed method achieves a success rate of 70-90% in general. Note that in our experiments, we divide the profile faulty models

TABLE I: Comparison of TV generation methods on MNIST using LeNet5. All achieve the same success rate of 85.96%.

TV generation method	# of generated TVs	# of compressed TVs	Repairing iteration min/med/avg/max
Hard-FGSM	1,929,092	346	1/16/17.49/56
Medium-FGSM	35,654	268	1/9/12.29/55
Soft-FGSM	17,139	399	1/12/12.75/53
Hard-PGD	1,900,133	295	1/12/13.8/55
Medium-PGD	32,809	585	1/9/12.13/55
Soft-PGD	34,513	306	1/16/19.05/70

Fig. 4: Generated TV examples with original seed TV on the left, generated TV in the middle, and the absolute difference between them on the right. (a) MNIST-LeNet-5, (b) CIFAR10-VGG11BN, (c) CIFAR100-VGG11BN.

and unseen faulty models randomly, especially for the large model with more diverse possible harmful bit-flip locations. Thus, some bit-flip patterns may only appear in the unseen faulty models, which will lower the success rate. In addition, there is a trade-off between the TV set size and success rate. We significantly reduce the number of TVs during the TV compression process, which also contributes success rate loss.

VI. Conclusion

This work proposed a novel defensive framework against fault attacks, which utilizes gradient-based optimization to produce a set of TVs. The set is then compressed to enhance efficiency. We have empirically verified that the selected TVs obtained using our methods can successfully detect and repair the models compromised by unseen fault patterns from attacks.

Acknowledgments

This work is partially supported by the National Science Foundation award 2047384.

References

[1] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[2] J. Hirschberg *et al.*, "Advances in natural language processing," *Science*, 2015.

[3] M. Bojarski *et al.*, "End to end learning for self-driving cars," *arXiv:1604.07316*.

[4] I. J. Goodfellow *et al.*, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations*, 2015.

TABLE II: Performance of the proposed NNTesting

Dataset-NN	TV generation method	# of generated TVs	# of compressed TVs	Pairwise collision using Equation (2)	Repairing iteration min/med/avg/max	Success rate (%)
MNIST-1FC	Soft-FGSM	9,666	207	0	1/8/9.03/37	76.83%
	Soft-PGD	11,871	202	0	1/7/8.07/34	76.83%
MNIST-LeNet5	Soft-FGSM	17,139	399	634	1/12/12.75/53	85.96%
	Soft-PGD	34,513	306	589	1/16/19.05/70	85.96%
CIFAR10-VGG11BN	Soft-FGSM	129,926	147	7	1/9/14.99/78	75.06%
	Soft-PGD	76,813	109	5	1/9/12.82/61	77.03%
CIFAR10-VGG19BN	Soft-FGSM	110,583	85	5	1/9/11.49/58	76.83%
	Soft-PGD	89,641	101	5	1/9/11.49/58	76.83%
CIFAR10-ResNet20	Soft-FGSM	103,663	126	136	1/6/10.52/78	88.34%
	Soft-PGD	72,808	140	132	1/1/5.7/57	94.76%
CIFAR100-VGG11BN	Soft-FGSM	95,109	45	0	1/6/6.81/28	80.50%
	Soft-PGD	127,943	41	0	1/5/6.11/27	80.48%
CIFAR100-VGG19BN	Soft-FGSM	98,806	20	0	1/8/8.83/25	69.03%
	Soft-PGD	146,032	25	0	1/10/10.28/26	69.03%
CIFAR100-ResNet20	Soft-FGSM	102,678	119	11	1/8/9.83/53	70.96%
	Soft-PGD	129,164	134	9	1/9/11.41/60	70.96%

[5] Y. Liu *et al.*, "Fault injection attack on deep neural network," in *IEEE/ACM International Conference on Computer-Aided Design*, 2017.

[6] P. Zhao *et al.*, "Fault sneaking attack: A stealthy framework for misleading deep neural networks," in *ACM/IEEE Design Automation Conference*, 2019.

[7] S. Hong *et al.*, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," in *USENIX Security*, 2019.

[8] J. Breier *et al.*, "Practical fault attack on deep neural networks," in *ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[9] M. C. Tol *et al.*, "Toward realistic backdoor injection attacks on DNNs using rowhammer," *arXiv:2110.07683*.

[10] F. Yao *et al.*, "DeepHammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips," in *USENIX Security*, 2020.

[11] A. S. Rakin *et al.*, "Bit-flip attack: Crushing neural network with progressive bit search," in *ICCV*, 2019.

[12] J. Li *et al.*, "Defending bit-flip attack through DNN weight reconstruction," in *ACM/IEEE Design Automation Conference*, 2020.

[13] A. S. Rakin *et al.*, "RA-BNN: Constructing robust & accurate binary neural network to simultaneously defend adversarial bit-flip attack and improve accuracy," *arXiv:2103.13813*.

[14] H. Chen *et al.*, "Deepattest: an end-to-end attestation framework for deep neural networks," in *ACM/IEEE Annual International Symposium on Computer Architecture*, 2019.

[15] F. S. Hosseini *et al.*, "Safeguarding the intelligence of neural networks with built-in light-weight integrity marks (LIMA)," in *IEEE International Symposium on Hardware Oriented Security and Trust*, 2021.

[16] Q. Liu *et al.*, "NeuroPots: Realtime proactive defense against Bit-Flip attacks in neural networks," in *USENIX Security*, 2023.

[17] P. Agrawal *et al.*, "On monte carlo testing of logic tree networks," *IEEE Transactions on Computers*, 1976.

[18] J. Carter *et al.*, "ATPG via random pattern simulation," in *IEEE International Symposium on Circuits and Systems*, 1985.

[19] A. S. Rakin *et al.*, "T-bfa: Targeted bit-flip adversarial weight attack," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[20] B. Ghavami *et al.*, "Blind data adversarial bit-flip attack against deep neural networks," in *25th Euromicro Conference on Digital System Design*, 2022, pp. 899–904.

[21] K. Lee *et al.*, "SparseBFA: Attacking sparse deep neural networks with the worst-case bit flips on coordinates," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2022.

[22] F. Meng *et al.*, "A self-test framework for detecting fault-induced accuracy drop in neural network accelerators," in *Asia and South Pacific Design Automation Conference*, 2021.

[23] M. Javaheripi *et al.*, "AccHashtag: Accelerated hashing for detecting fault-injection attacks on embedded neural networks," *ACM Journal on Emerging Technologies in Computing Systems*, 2022.

[24] Z. Navabi, *Digital System Test and Testable Design: Using HDL Models and Architectures*, 2010.

[25] T. Zhang *et al.*, "A systematic dnn weight pruning framework using alternating direction method of multipliers," in *European Conference on Computer Vision*, 2018.

[26] A. Madry *et al.*, "Towards deep learning models resistant to adversarial attacks," in *International Conference on Learning Representations*, 2018.

[27] N. Yogi *et al.*, "N-model tests for VLSI circuits," in *Southeastern Symposium on System Theory*, 2008.

[28] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019.

[29] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[30] A. Krizhevsky *et al.*, "Cifar-10," *http://www.cs.toronto.edu/kriz/cifar.html*, 2010.

[31] K. He *et al.*, "Deep residual learning for image recognition," in *IEEE conference on computer vision and pattern recognition*, 2016.