GENIE: GENetIc Algorithm-Based REliability Assessment Methodology for Deep Neural Networks

Samira Nazari
University of Zanjan
Zanjan, Iran
samira.nazari@znu.ac.ir

Mahdi Taheri
Tallinn University of Technology
Brandenburgische Technische Universität Cottbus
Cottbus, Germany
mahdi.taheri@taltech.ee
*Corresponding author

Ali Azarpeyvand
University of Zanjan
Tallinn University of Technology
Zanjan, Iran
azarpeyvand@znu.ac.ir

Mohsen Afsharchi
University of Zanjan
Zanjan, Iran
afsharchi@znu.ac.ir

Christian Herglotz
Brandenburgische Technische Universität Cottbus
Cottbus, Germany
christian.herglotz@b-tu.de

Maksim Jenihhin
Tallinn University of Technology
Tallinn, Estonia
maksim.jenihhin@taltech.ee

***Abstract*—As deep neural networks (DNNs) are becoming vital to numerous safety-critical applications, ensuring their fault reliability is crucial. This paper presents a hybrid framework that combines genetic algorithms with fault injection and analytical methods to identify vulnerable neurons and layers in DNNs. Validated on LeNet-5, AlexNet, and VGG-11, our approach significantly enhances model accuracy in harsh environments by protecting critical components, achieving reliability improvements (accuracy drop of the model compared to the golden network after fault injection) of 69.86% for LeNet, and 99.65% for AlexNet at BER=1E-4, and 91.37% improvement for VGG at BER=1E-6. Furthermore, our framework reduces computational costs by requiring fewer inferences than traditional analytical methods, highlighting its potential to improve DNN accelerators' reliability and contribute to their safety.**

***Index Terms*—deep neural networks, parallel processing, memory overhead, reliability, DNN accelerator**

I. Introduction

Deep Neural Networks (DNNs) are increasingly used in safety-critical applications, e.g., perception in automotive [1], and their hardware reliability assessment is necessary during the design and development phase [2]. The size of DNNs, particularly Convolutional Neural Networks (CNNs), in terms of memory footprint and number of operations, is rapidly growing [3], [4]. With the rapid growth of emerging DNNs in terms of their memory and computation requirements, their reliability analysis becomes prohibitively complex, time-consuming, and resource-intensive [5].

It is shown in the recent literature that the majority of papers studying the reliability of DNNs exploit a Fault Injection (FI) approach [6], [7]. Software-level simulation FI is widely adopted in reliability analysis due to its low design time and fast execution [8]. Multiple open-source tools for software-level simulation-based reliability analysis are proposed in the literature, with the most adopted ones including PytorchFI [9] and TensorFI [10], [11], enabling FI simulations in PyTorch and TensorFlow respectively. Any FI simulation requires multiple simulations while a random set of parameters/activations are faulty and propagated through the forward execution of a DNN. The number of simulations affects the accuracy of the obtained metrics. In exhaustive FI, all possible bitflips in the fault space are flipped individually, leading to a huge simulation space, which is impractical since DNNs possess millions of parameters and activations. Statistical Fault Injection (SFI) attempts to reduce the number of simulations based on statistical analysis while guaranteeing a certain maximum error margin [12]. However, it is shown that these statistics do not result in accurate FI results [13]. Analytical methods for reliability analysis of DNNs are proposed to address its scalability issue. Throughout the literature, multiple techniques are proposed to obtain the resilience of CNNs based on non-FI approaches [6]. In [14], authors proposed DeepVigor, an analytical methodology to study the resilience of CNNs against faults that provides a Layer Vulnerability Factor (LVF). Although this methodology is demonstrated to be faster than FI, it requires a high execution time to obtain LVFs. The reason is that DeepVigor attempts to analyze all neurons in a CNN as well as to find the vulnerability values in a huge space of numbers. DeepVigor faces several critical limitations when assessing fault vulnerability in deep neural networks. First, they are unable to account for the impact of multiple faults, which introduces significant uncertainty in real-world scenarios. Second, the error propagation is

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

approximated, leading to inaccuracies in resilience analysis and fault vulnerability factor (VF) calculations. Additionally, DeepVigor incorrectly assumes that large value bit-flips are always critical, overlooking cases where such errors do not cause misclassification. Finally, its exhaustive fault analysis across all neurons is computationally expensive and time-consuming, making the method less practical for large-scale networks.

This paper presents GENIE, a fast hybrid methodology based on a Genetic Algorithm (GA) and FI to assess the reliability of DNNs based on the layer vulnerability factors. The contributions of this paper are as follows:

- **Reliability Assessment Methodology:** Developed a hybrid Methodology combining GA with fault injection for accurate and fast Layer Vulnerability analysis in DNN.
- **Fast Vulnerability Detection:** Achieved rapid detection of critical neurons and layers using GA, significantly reducing required reliability assessment time, power, and efficiency.
- **Open Source GitHub Repository:** Released the implementation as an open-source framework to facilitate further research and collaboration. https://github.com/nilay1400/Genie
- **Testing and Validation on Benchmarks:** Tested and validated the methodology on a variety of state-of-the-art DNN architectures such as LeNet, AlexNet, and VGG, demonstrating its effectiveness.

The remainder of this paper is structured as follows: Section II presents the proposed methodology. Section III discusses experimental results and Section IV concludes the paper.

II. Proposed Methodology

To evaluate the reliability of DNNs using GA, the first step is to define the number of faults to be injected into the model, along with the population size (number of individuals), individual size, and the number of generations for the GA (A_step 1, Figure 1). The number of faults corresponds to the number of neurons the GA will evaluate for vulnerability. This fault count is determined as a percentage of the total number of neurons, denoted as α. For instance, if $\alpha = 0.0001$, the GA will identify $\alpha \times N$ of the most vulnerable weights, where N represents the total number of neurons in the DNN.

Next, the GA generates the initial population randomly (A_step 2, Figure 1). Each individual in the population represents a set of neurons to be perturbed for reliability analysis. For instance, if the population size is 100 and the individual size is 5, the GA generates 100 sets of 5 neurons each. Faults are then injected into the corresponding neurons in all sets, and the resulting loss in model performance is evaluated. Based on the loss values, the GA evolves the population by generating new generations, aiming to maximize the loss. This allows the identification of the most vulnerable set of neurons. Notably, after the first generation, not all individuals in subsequent generations are evaluated; only a subset of the population undergoes evaluation to optimize computational resources.

A. Genetic Algorithm steps

The GA used for identifying the most critical weights in a DNN involves the following detailed steps:

1. Initialization

The initial population is randomly generated, where each individual represents a set of neurons, and each neuron is identified by the layer it belongs to and its index within that layer. Key parameters for the initialization include:

Population size: Denoted as P, this defines the total number of individuals (solutions) in the population. In this implementation, $P = 100$ is chosen.

Individual size: Each individual represents a set of k neurons to be perturbed. For example, if $k = 5$, then each individual contains 5 neuron indices. The neuron indices are sampled randomly from all layers that contain trainable weights (excluding biases). The minimum individual size is determined by 1, where k is the minimum value the individual size can take and N_l is the number of layers in DNN. This equation ensures that neurons from all layers can participate in the GA, allowing us to comprehensively analyze the vulnerability of each layer.

$$k = N_l + 1 \tag{1}$$

2. Fitness Evaluation

The fitness function evaluates the impact of perturbing selected neurons in each individual on the overall DNN performance. The steps are:

Fault Injection: For each individual, the algorithm selects the weights corresponding to the neurons and injects faults in them by a random amount, e.g., flipping a random bit of the weight values. Let $w_{i,j}$ represent the weight of the $i-th$ neuron in layer j. The perturbation can be modeled as:

$$w_{i,j}^{'} = w_{i,j} + \epsilon \tag{2}$$

Evaluation: After fault injection, the modified DNN is evaluated on a validation dataset using a loss function, typically cross-entropy loss for classification tasks. The fitness score of the individual is proportional to the increase in loss. Denote the original loss as $L_{original}$ and the loss after perturbation as $L_{perturbed}$. The fitness F is defined as:

$$F = L_{perturbed} - L_{original} \tag{3}$$

A higher fitness value indicates that the individual caused a significant degradation in model accuracy, suggesting that the perturbed neurons are more critical.

3. Selection

Selection is performed using a tournament selection mechanism [15]:

- A subset of individuals is randomly selected from the population (with size defined by the tournament size, typically 2-5 individuals).
- The individual with the highest fitness within the tournament is chosen to pass on its genetic material to the next generation. This ensures that fitter individuals have a higher chance of being selected, while still maintaining some genetic diversity.

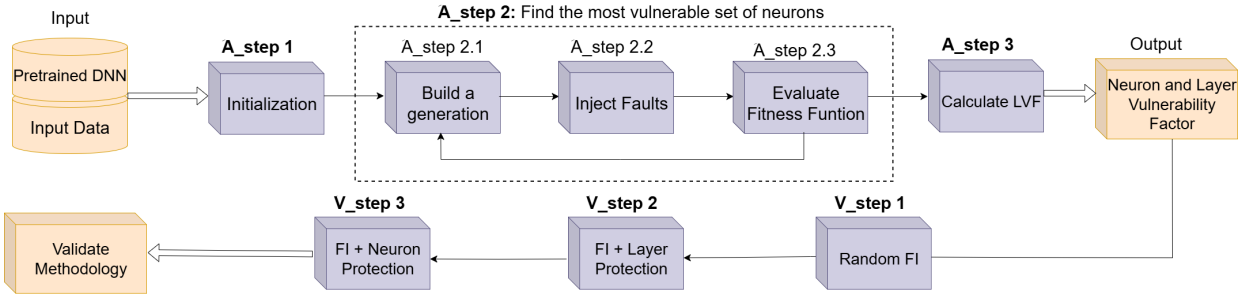


Fig. 1: Proposed assessment methodology and methodology validation steps for evaluating the reliability of DNNs. A_step and V_step refer to the Assessment and Validation phases, respectively.

4. Crossover
Once a set of parents is selected, crossover is applied to create new offspring. In GA, offspring refers to the new individuals (solutions) created by combining or modifying the characteristics of parent individuals through operations like crossover and mutation. These offspring represent potential solutions in the next generation of the algorithm. The uniform crossover operator is used: For each neuron in the individual, a random decision is made (with a probability of 0.5) to inherit the neuron from either parent. This results in offspring that have a mix of neurons from both parents. Let p_1 and p_2 be two parent individuals:

$$o_i = \begin{cases} p_{1,i} & \text{with probability } 0.5 \\ p_{2,i} & \text{with probability } 0.5 \end{cases} \tag{4}$$

where o_i is the $i-th$ neuron of the offspring.
5. Mutation
Mutation introduces random changes to individuals to maintain diversity and avoid premature convergence. The mutation operator selects random neurons from the individual and replaces them with new, randomly selected neurons from the network. This occurs with a small mutation probability p_m (e.g., 0.01): For each neuron in the individual, with probability p_m, the neuron is replaced by another randomly chosen neuron from the same layer or another layer.
Mathematically, for an individual o:

$$o_i \rightarrow random(N), \tag{5}$$

where N is the total number of neurons, and o_i is a randomly selected neuron index in the individual.
6. Survivor Selection and Elitism
To maintain the best solutions across generations, the algorithm employs elitism, where a small percentage of the best-performing individuals from the current generation are carried over to the next generation unchanged. This ensures that high-performing solutions are not lost due to random crossover or mutation. The rest of the population is filled with offspring generated through crossover and mutation.

7. Termination
The GA continues to run for multiple generations until the final result stabilizes, indicating that the methodology's objective—determining the layer vulnerability factor (as explained in Section II-B)—remains nearly constant across the last few generations.
8. Output
At the end of the evolution process, the GA identifies the set of neurons that, when perturbed, cause the largest loss in model accuracy. These neurons are deemed the most critical for the DNN's performance and provide insights into the network's vulnerability to faults.

B. Layer Vulnerability Factor

At the end of the GA, the output is a set of the most vulnerable neurons in the DNN. These are the neurons whose perturbation resulted in the largest degradation of model performance. Based on this output, we can calculate the Layer Vulnerability Factor (LVF) for each layer in the network (A_step 3, Figure 1). The LVF quantifies how critical each layer is to the overall network performance by measuring the proportion of vulnerable neurons in that layer.

The LVF for a given layer l can be defined as:

$$LVF(l) = \frac{N_{vulnerable}(l)}{N_{total}(l)}, \tag{6}$$

where $N_{vulnerable}(l)$ is the number of neurons in layer l identified as the most vulnerable by the GA and $N_{total}(l)$ is the total number of neurons in layer l.
This metric provides insight into the relative importance of each layer within the DNN. A higher LVF value indicates that a larger proportion of the neurons in that layer are critical to the model's robustness, making it more susceptible to faults. By calculating the LVF for each layer, we can identify which layers contribute most to the network's vulnerability and focus on techniques to enhance their reliability.

C. Methodology Validation

To validate the performance of the GA, we use multiple-bit fault injection to simulate real-world faults in the DNN

(V_step 1, Figure 1). In this process, random bit-level perturbations are applied to the weights based on the Bit Error Rate (BER). The BER defines the percentage of bits across all weights in the DNN that will be flipped in each iteration of fault injection.

At each iteration, several bits are randomly selected according to the BER. These bits are flipped (from 0 to 1 or 1 to 0) within the weight representations, introducing perturbations into the model. This simulates scenarios where faults affect multiple bits during time in the memory of the model's weights.

The fault injection process helps validate the results of the GA by assessing how the identified critical neurons respond to multiple-bit perturbations. By applying different BER values, we can evaluate the resilience of the DNN and verify that the neurons identified by the GA are the most vulnerable to such faults.

1) Validation with Layer Protection: After identifying the most critical neurons and layers through the GA, a comparative analysis between different fault injection scenarios is performed (V_step 2, Figure 1). The goal is to assess how protecting the most vulnerable layers compares to protecting the least vulnerable layers in terms of the accuracy drop caused by faults. The comparison is based on two key scenarios:

- Scenario 1: Protection of the Most Vulnerable Layers. In this scenario, a percentage of the most vulnerable layers, as identified by the GA, is considered protected. The remaining layers are exposed to faults. The number of faults injected into the DNN is determined by the BER. For instance, a portion of the bits in the network's weights are randomly selected and flipped, but none of the bits in the protected, critical layers are perturbed. The protected layers are considered fault-free, representing scenarios with enhanced error correction mechanisms or hardware-level protections.
- Scenario 2: Protection of the Least Vulnerable Layers. In this scenario, instead of protecting the most vulnerable layers, the same percentage of the least vulnerable layers is shielded from faults. The number of faults injected is still governed by the BER, meaning that the same percentage of bits (e.g., 1% for $BER = 0.01$) is flipped across the unprotected layers, including the critical ones. This helps to evaluate how the model performs when non-critical layers are prioritized for protection instead of the more vulnerable ones.

After fault injection, the drop in accuracy is measured for both scenarios. The difference in accuracy degradation between protecting the most vulnerable layers and protecting the least vulnerable layers helps validate the importance of the critical layers identified by the GA.

By comparing these two fault injection strategies, The effectiveness of the proposed methodology can be assessed by targeting the most vulnerable layers for protection versus using a random or least-vulnerable protection scheme. A smaller accuracy drop in Scenario 1 would indicate that protecting the most critical layers yields a more resilient network.

TABLE I: THE NUMBER OF WEIGHTS AND LAYERS OF DNNS STUDIED IN THIS WORK

Model	Accuracy (%)	Total #of Weights	Conv	FC	BN
LeNet	98.90	61,470	2	3	0
AlexNet	95.52	58,289,504	5	3	0
VGG-11	92.39	28,135,808	8	3	8

2) Validation with Neuron Protection: To further validate the effectiveness of the GA in identifying critical neurons, the outcomes of random fault injections are compared under two scenarios: (1) when the most vulnerable neurons are excluded from the fault list and (2) when the most vulnerable neurons are included in the fault list (V_step 3, Figure 1). In both cases, the number of faults injected is governed by the BER.

- Scenario 1: Excluding Vulnerable Neurons from Fault. Injection. In this scenario, the most vulnerable neurons identified by the GA are protected from fault injection. The faults are injected only into the remaining, less critical neurons. The number of injected faults is determined by the BER.
- Scenario 2: Including Vulnerable Neurons in Fault Injection. In this scenario, the same number of faults, determined by the BER, is injected across the entire network, including the most vulnerable neurons. This scenario helps to evaluate the accuracy degradation when critical neurons are exposed to faults and demonstrates how the network behaves without any protection.

After injecting faults in both scenarios, the resulting accuracy of the DNN is compared. By measuring the accuracy drop, it can be assessed how much the exclusion of vulnerable neurons from fault injection improves the model's resilience.

This comparison provides clear validation of the GA's effectiveness. If the accuracy drop is significantly smaller when vulnerable neurons are excluded from fault injection (Scenario 1), it confirms that the GA successfully identifies neurons that are most sensitive to bit-level faults.

III. EXPERIMENTAL RESULTS

A. Experimental Setup

In this study, the GA's capability to identify the most critical neurons in DNNs across three widely used architectures is shown, each trained on distinct datasets as follows: LeNet-5 on MNIST, AlexNet of Fashion MNIST, and VGG-11 on CIFAR-10. Each network is trained to achieve near state-of-the-art accuracy on its respective dataset. Different BERs are used to inject random faults into the network's weights during evaluation, simulating accumulated memory faults.

The base accuracy, the number of weights, as well as the count of convolutional, fully connected, and batch normalization layers for each network, are displayed in Table I. This information provides a clear overview of the complexity of each model and serves as a reference for interpreting the results of the GA in identifying critical weights and layers. All experiments are conducted on an NVIDIA GeForce GTX 1050 Ti GPU, utilizing the PyTorch framework for model training, testing, and fault injection.

TABLE II: EXECUTION TIME (SECONDS) OF GA FOR DIFFERENT BENCHMARKS WITH DIFFERENT VALUES OF α

α	1E-6	1E-5	1E-4	1E-3	1E-2
LeNet	N/A	N/A	7575.60	7808.02	7990.09
AlexNet	32425.11	32375.82	37471.47	50009.82	N/A
VGG-11	N/A	12888.32	16087.27	20683.54	N/A

B. GA Execution

The GA selects a subset of weights for evaluation based on the parameter α, which represents a portion of the total weights in the model. In this study, α is chosen from a range of $1E-6$ to $1E-2$, depending on the total number of weights in each neural network. This adaptive selection ensures that the GA scales appropriately for networks of different sizes.

For all experiments, the generation size is set to 40, meaning the algorithm runs for 40 generations, and the population size is fixed at 100, meaning each generation consists of 100 individuals.

As observed in the experiments, larger values of α result in longer execution times due to the increased number of weights being evaluated. Additionally, networks with a higher total number of parameters, such as AlexNet and VGG-11, naturally have longer execution times. However, for AlexNet, it is observed that reducing α below a certain threshold (e.g., $1E-5$) does not significantly decrease the execution time. This suggests that the overhead of GA operations, such as population management and fitness evaluations, plays a larger role at smaller scales of α.

The execution time of each run for the respective models, including LeNet-5, AlexNet, and VGG-11, is summarized in Table II, showing how execution time scales with both α and the number of parameters in the model.

C. Neuron Protection

The results of the neuron protection experiments are illustrated for LeNet-5 as an example in Figure 2. The findings reveal that there is consistently less accuracy drop across most BERs and α values when vulnerable neurons are excluded from the fault injection list. This reduction in accuracy degradation underscores the effectiveness of the GA in accurately identifying critical neurons that are most susceptible to faults.

By protecting these vulnerable neurons during fault injection, it is shown that a significant improvement in the model's resilience against random bit flips. This outcome confirms that the GA successfully detects and isolates the neurons that, when compromised, would lead to greater accuracy loss in the neural network.

D. Layer Protection

The LVF is calculated for all layers of the tested models using Equation 6. LVF provides an indication of the vulnerability of each layer by measuring how critical the weights within the layer are to the overall accuracy of the model when perturbed. The LVF values for the layers of each model are displayed in Table III for LeNet-5, Table IV for AlexNet, and Table V for VGG-11, across different α values. These tables show how the vulnerability of each layer varies with the portion of weights considered by the GA.

TABLE III: LVF OF LENET-5 LAYERS FOR DIFFERENT α VALUES

α	1E-4	1E-3	1E-2
features.0	2.00E-2	9.33E-2	9.17E-1
features.3	4.17E-4	5.83E-32	4.79E-2
classifier.0	2.08E-5	1.67E-4	2.81E-3
classifier.2	0.00E-0	1.39E-3	1.21E-2
classifier.4	1.19E-3	1.31E-2	1.25E-1

TABLE IV: LVF OF ALEXNET LAYERS FOR DIFFERENT α VALUES

α	1E-6	1E-5	1E-4	1E-3
conv1	5.17E-4	6.63E-3	6.29E-2	6.25E-1
conv2	1.79E-5	1.03E-4	1.22E-3	1.18E-2
conv3	4.52E-6	8.02E-5	8.17E-4	8.22E-3
conv4	3.77E-6	6.25E-5	5.30E-4	5.56E-3
conv5	7.91E-6	7.91E-5	8.14E-4	8.13E-3
fc1	2.12E-7	1.99E-6	1.86E-5	1.90E-4
fc2	5.96E-7	3.93E-6	4.37E-5	4.35E-4
fc3	1.71E-4	1.88E-3	1.87E-2	1.82E-1

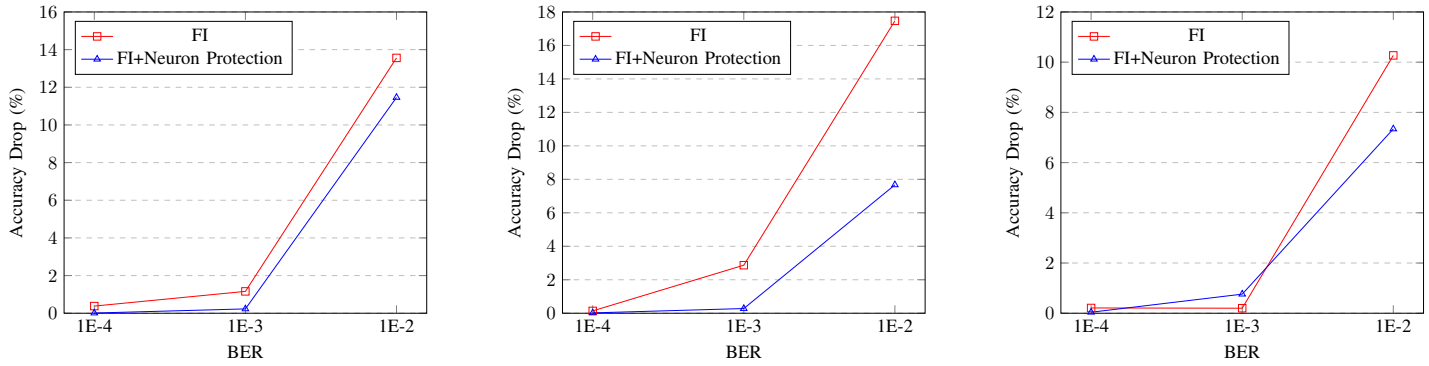
The results show that the order of vulnerable layers according to LVF is almost the same between LeNet-5 and AlexNet across different α values. However, for VGG-11, the order of vulnerable layers differs slightly at different α values, indicating some variability in sensitivity across layers.

Interestingly, as summarized in Table VI, where colorful cells highlight the shifts in layer rankings, even with a small value of α, the correct order of LVF values can be obtained for all layers. This allows for faster identification of critical layers, as the layer vulnerability order differs by only one or two positions across the varying α values. This suggests that using a small α can significantly speed up the LVF calculation while still accurately identifying the most vulnerable layers.

TABLE V: LVF OF VGG-11 LAYERS FOR DIFFERENT α VALUES

α	1E-5	1E-4	1E-3
features.0	9.26E-3	8.10E-02	8.50E-01
features.1	1.88E-1	2.13E+00	2.34E+01
features.4	1.63E-4	2.08E-03	2.02E-02
features.5	1.41E-1	1.18E+00	1.16E+01
features.8	6.10E-5	4.41E-04	5.30E-03
features.9	5.47E-2	5.35E-01	5.74E+00
features.11	3.22E-5	2.58E-04	2.53E-03
features.12	4.30E-2	6.29E-01	5.66E+00
features.15	1.27E-5	1.30E-04	1.24E-03
features.16	2.54E-2	3.07E-01	2.89E+00
features.18	5.51E-6	6.27E-05	6.35E-04
features.19	3.71E-2	3.11E-01	2.85E+00
features.22	3.81E-6	6.06E-05	6.35E-04
features.23	3.91E-2	2.81E-01	3.00E+00
features.25	3.93E-6	6.10E-05	6.41E-04
features.26	4.49E-2	3.09E-01	2.86E+00
classifier.0	7.15E-6	6.39E-05	6.64E-04
classifier.3	5.96E-7	8.76E-06	8.42E-05
classifier.6	3.91E-4	4.05E-03	3.67E-02

Once the vulnerable layers are identified, a threshold can be set to determine what portion of these layers should be employed for validating the algorithm through fault injection.



(a) α=1E-4 (b) α=1E-3 (c) α=1E-2

Fig. 2: Random fault injection along with neuron protection in LeNet

TABLE VI: SORTING VGG-11 LAYERS ACCORDING TO VULNERABILITY FACTOR WITH DIFFERENT α VALUES

1E-5	1E-4	1E-3
features.1	features.1	features.1
features.5	features.5	features.5
features.9	features.12	features.9
features.26	features.9	features.12
features.12	features.19	features.23
features.23	features.26	features.16
features.19	features.16	features.26
features.16	features.23	features.19
features.0	features.0	features.0
classifier.6	classifier.6	classifier.6
features.4	features.4	features.4
features.8	features.8	features.8
features.11	features.11	features.11
features.15	features.15	features.15
classifier.0	classifier.0	classifier.0
features.18	features.18	features.25
features.22	features.25	features.22
features.25	features.22	features.18
classifier.3	classifier.3	classifier.3

In this study, this percentage is set to 20-25%, meaning that only the most vulnerable 20-25% of the layers are protected for further validation.

This approach allows us to focus on the most critical layers, significantly reducing the amount of fault injection required, while still effectively validating the GA's ability to identify vulnerable neurons and layers.

Figures 3, 4, and 5 illustrate the layer protection results for LeNet-5, AlexNet, and VGG-11, respectively. In these experiments, the most vulnerable layers identified by the GA are protected. Specifically, in LeNet-5, the most vulnerable layer is protected; in AlexNet, the two most vulnerable layers are protected; and in VGG-11, the five most vulnerable layers, detected with $\alpha = 1E-5$, are protected.

The results show that this protection mechanism significantly reduces the accuracy drop caused by faults. For instance, in VGG-11, protecting the most vulnerable layers leads to noticeable mitigation of the impact of faults, confirming the effectiveness of the GA in identifying the critical layers.

To further validate the results, protecting the least vulnerable

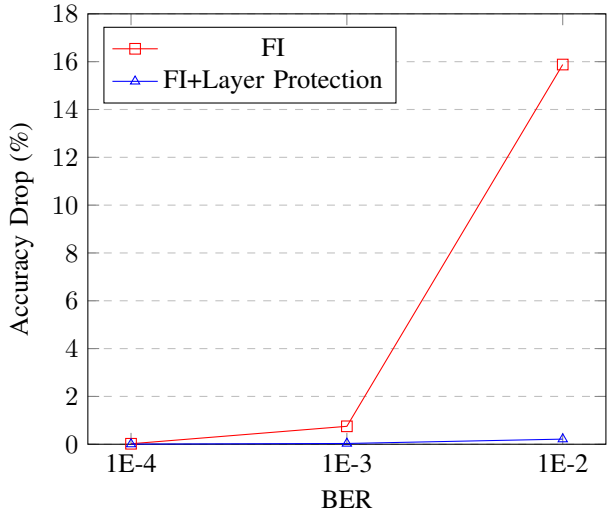


Fig. 3: Random fault injection along with layer protection in LeNet

layers is also tested. In VGG-11, for example, protecting the five least vulnerable layers shows marginal improvement in accuracy drop compared to the scenario without any protection. Due to the negligible difference, these figures are excluded from the paper.

These findings confirm that the LVF values and vulnerable layers identified by the GA are accurate, as protecting the layers with higher LVF values improves model resilience while protecting the least vulnerable layers provides no benefit.

E. Comparison with State-of-the-Art

The proposed hybrid framework is compared with the state-of-the-art (SOTA) analytical method [14], which is used for identifying vulnerable layers and neurons in DNNs. While the reference method focuses primarily on the Neuron Vulnerability Factor (NVF) which also is used to obtain LVFs, GENIE emphasizes obtaining LVFs, offering insights that are particularly valuable for layer-level resilience analysis.

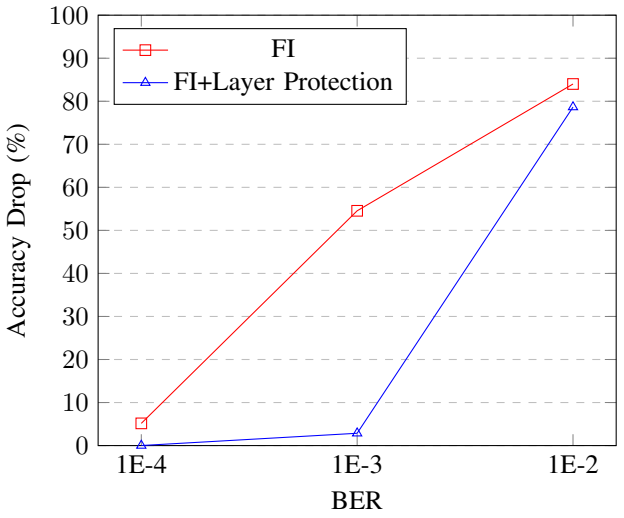


Fig. 4: Random fault injection along with layer protection in AlexNet

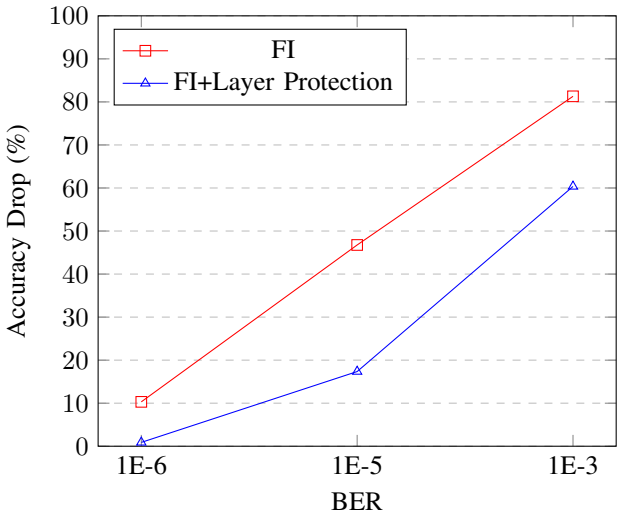


Fig. 5: Random fault injection along with layer protection in VGG-11

GENIE combines fault injection with analytical assessment, allowing verification of vulnerability factors through actual fault injections. This provides exact results that complement SOTA methods, which approximate fault-induced changes without direct injections. Additionally, while SOTA methods effectively analyze single-bit faults, GENIE supports multi-bit faults at varying levels determined by the BER, enabling a more comprehensive fault tolerance analysis.

In terms of efficiency, GENIE demonstrates faster execution time, particularly in larger models. Table VII compares the two approaches, showing inference counts in the second and third columns, with the final column highlighting the speedup achieved by GENIE relative to SOTA analytical methods. This gain becomes more pronounced as model size increases. While SOTA methods perform an inference per parameter to estimate vulnerability factors, GENIE maintains a consistent

TABLE VII: COMPARING EXECUTION TIME OF GA AND [14] ($\times no.inferences$) TO EXTRACT LVF

Model	[14]	GA	Speedup
LeNet	61,470	2,568	23.93
AlexNet	58,289,504	2,444	23,850.04
VGG-11	28,135,808	2,556	11007.74

requirement of approximately 2,500 inferences across models, as the execution time depends primarily on the GA algorithm itself rather than the parameter count. This efficiency makes it suitable for large networks.

Lastly, GENIE operates independently of specific datasets, whereas SOTA methods may rely on dataset characteristics when evaluating vulnerabilities. In summary, both SOTA analytical methods and our hybrid approach have unique strengths suited to distinct applications, such as neuron-focused versus layer-focused vulnerability analysis. By supporting multi-bit fault injection and ensuring adaptability to various datasets, GENIE complements SOTA methods, offering a flexible alternative for analyzing DNN robustness.

IV. CONCLUSION

In this paper, we propose a hybrid framework based on a GA for detecting the most vulnerable layers in DNNs. By combining both fault injection and analytical techniques, our approach improves upon purely analytical methods. Our experiments on three different networks—LeNet-5, AlexNet, and VGG-11—demonstrate the effectiveness of the proposed method. The GA efficiently identifies vulnerable layers by leveraging a limited number of inferences, significantly reducing the computational cost compared to SOTA. GENIE significantly enhances model accuracy in harsh environments by protecting critical components, achieving reliability improvements (accuracy drop of the model compared to the golden network after fault injection) of 69.86% for LeNet, and 99.65% for AlexNet at BER=1E-4, and 91.37% improvement for VGG at BER=1E-6. These achievements are made with faster vulnerable layer detection compared to pure fault injection and analytical approaches. The proposed framework proves not only to be more accurate due to its real fault injection capability but also significantly faster, especially for larger models.

REFERENCES

[1] H. Y. Yatbaz, M. Dianati, and R. Woodman, "Introspection of dnn-based perception functions in automated driving systems: State-of-the-art and open research challenges," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[2] M. Taheri, "Dnn hardware reliability assessment and enhancement," in *27th IEEE European Test Symposium (ETS)*, 2022.

[3] H. Hussain, P. Tamizharasan, and C. Rahul, "Design possibilities and challenges of dnn models: a review on the perspective of end devices," *Artificial Intelligence Review*, p. 1–59, 2022.

[4] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, "Trends in ai inference energy consumption: Beyond the performance-vs-parameter laws of deep learning," *Sustainable Computing: Informatics and Systems*, vol. 38, p. 100857, 2023.

[5] M. Taheri, M. Daneshtalab, J. Raik, M. Jenihhin, S. Pappalardo, P. Jimenez, B. Deveautour, and A. Bosio, "Saffira: a framework for assessing the reliability of systolic-array-based dnn accelerators," in *2024 27th International Symposium on Design & Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE, 2024, pp. 19–24.

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

[7] S. Nazari, M. Taheri, A. Azarpeyvand, M. Afsharchi, T. Ghasempouri, C. Herglotz, M. Daneshtalab, and M. Jenihhin, "Fortune: A negative memory overhead hardware-agnostic fault tolerance technique in dnns," in *2024 IEEE 33rd Asian Test Symposium (ATS)*. IEEE, 2024, pp. 1–6.

[8] M. H. Ahmadilivani, A. Bosio, B. Deveautour, F. F. Dos Santos, J.-D. Guerrero-Balaguera, M. Jenihhin, A. Kritikakou, R. L. Sierra, S. Pappalardo, J. Raik *et al.*, "Special session: Reliability assessment recipes for dnn accelerators," in *2024 IEEE 42nd VLSI Test Symposium (VTS)*. IEEE, 2024, pp. 1–11.

[9] A. Mahmoud, N. Aggarwal, A. Nobbe, J. R. S. Vicarte, S. V. Adve, C. W. Fletcher, I. Frosio, and S. K. S. Hari, "Pytorchfi: A runtime perturbation tool for dnns," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2020, p. 25–31.

[10] Z. Chen, N. Narayanan, B. Fang, G. Li, K. Pattabiraman, and N. DeBardeleben, "Tensorfi: A flexible fault injection framework for tensorflow applications," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 2020, p. 426–435.

[11] S. Laskar, M. H. Rahman, and G. Li, "Tensorfi+: a scalable fault injection framework for modern deep learning neural networks," in *2022 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2022, p. 246–251.

[12] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation & Test in Europe Conference & Exhibition*, 2009, p. 502–506.

[13] A. Ruospo, G. Gavarini, C. D. Sio, J. Guerrero, L. Sterpone, M. S. Reorda, E. Sanchez, R. Mariani, J. Aribido, and J. Athavale, "Assessing convolutional neural networks reliability through statistical fault injections," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, p. 1–6.

[14] M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, "Deepvigor: Vulnerability value ranges and factors for dnns' reliability assessment," in *2023 IEEE European Test Symposium (ETS)*, 2023, p. 1–6.

[15] B. L. Miller, D. E. Goldberg *et al.*, "Genetic algorithms, tournament selection, and the effects of noise," *Complex systems*, vol. 9, no. 3, pp. 193–212, 1995.